

Ensemble Methods

Что такое ансамбли

Ансамбль можно собрать как угодно, хоть случайно нарезать в тазик классификаторы и залить регрессией. За точность, правда, тогда никто не ручается. Потому есть три проверенных способа делать ансамбли - бэггинг, стеккинг и бустинг.

Bootstrapping and bagging

<https://youtu.be/Xz0x-8-cgaQ>

<https://youtu.be/N4ZQQqyIf6k>

[https://ru.wikipedia.org/wiki/Бутстрэп_\(статистика\)](https://ru.wikipedia.org/wiki/Бутстрэп_(статистика))

Для начала вспомним статистику.

Пусть есть датасет X размера m . Возьмём из него случайным образом объект. Проделаем это m раз, допуская повторение объектов. Получим новый датасет X_j того же размера.

Сгенерируем N таких датасетов из исходного.

В статистике такой подход используют для построения оценок, в машинном обучении - для построения ансамблей.

Допустим, мы решаем задачу регрессии, b_j - алгоритм, обученный на сгенерированном бутстррапом датасете X_j . Посчитаем ошибку каждого алгоритма (предсказание - истинное значение) и MSE:

Error of model trained on X_j : $\varepsilon_j(x) = b_j(x) - y(x)$, $j = 1, \dots, N$,

Then $\mathbb{E}_x(b_j(x) - y(x))^2 = \mathbb{E}_x \varepsilon_j^2(x)$.

Средняя ошибка всех моделей:

$$E_1 = \frac{1}{N} \sum_{j=1}^N \mathbb{E}_x \varepsilon_j^2(x)$$

Сделаем теперь предположение, что ошибка каждой модели несмещенная и что ошибки двух моделей независимы:

$$\begin{aligned}\mathbb{E}_x \varepsilon_j(x) &= 0; \\ \mathbb{E}_x \varepsilon_i(x) \varepsilon_j(x) &= 0, \quad i \neq j.\end{aligned}$$

Тогда вводится новая модель:

$$a(x) = \frac{1}{N} \sum_{j=1}^N b_j(x).$$

Новый объект, который мы подаем на вход этой модели, пройдет по всем N моделям, после чего все предсказания усредняются. Эта модель называется ансамблем.

Средняя ошибка ансамбля:

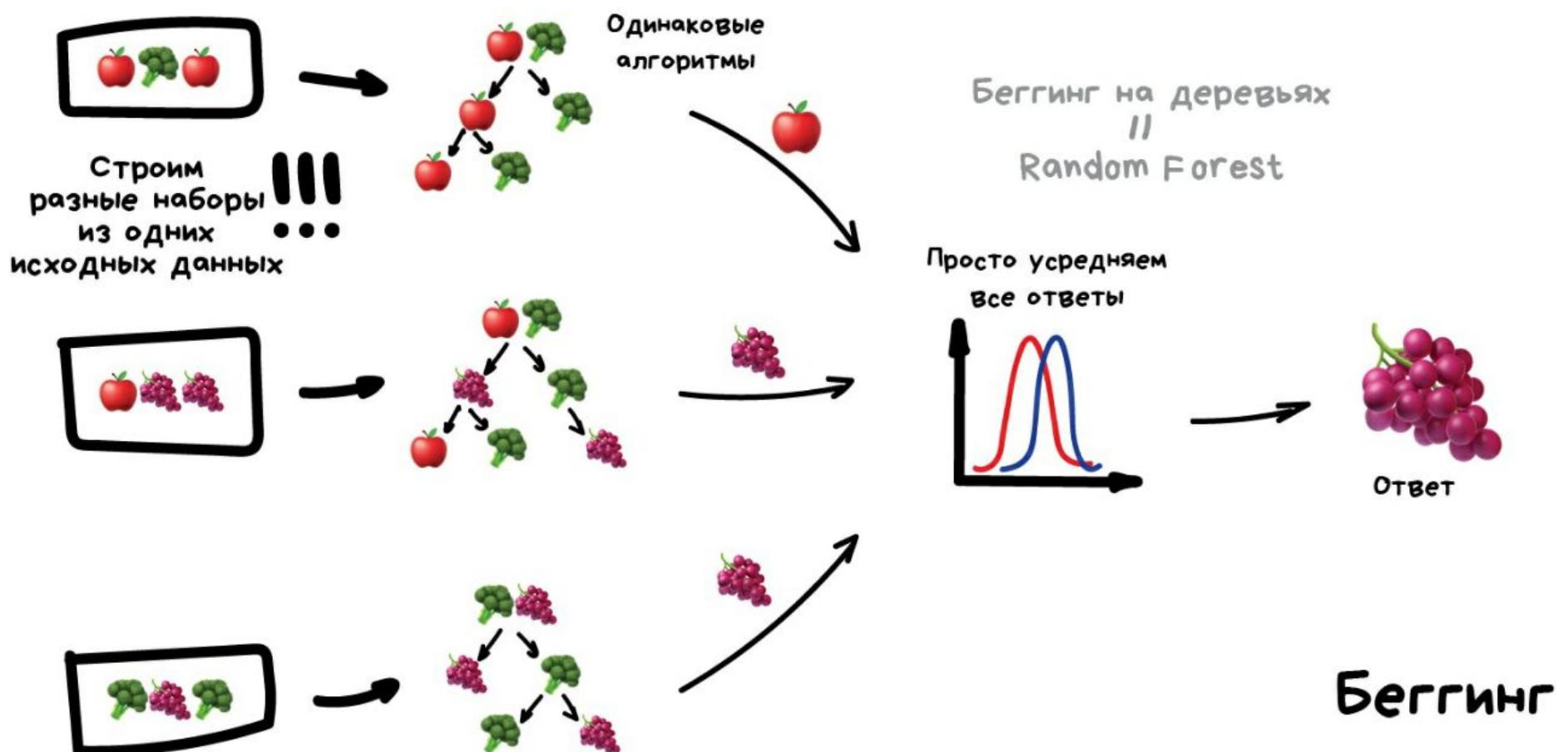
$$\begin{aligned}E_N &= \mathbb{E}_x \left(\frac{1}{N} \sum_{j=1}^N b_j(x) - y(x) \right)^2 = \\ &= \mathbb{E}_x \left(\frac{1}{N} \sum_{j=1}^N \varepsilon_j(x) \right)^2 = \\ &= \frac{1}{N^2} \mathbb{E}_x \left(\sum_{j=1}^N \varepsilon_j^2(x) + \underbrace{\sum_{i \neq j} \varepsilon_i(x) \varepsilon_j(x)}_{=0} \right) = \\ &= \frac{1}{N} E_1.\end{aligned}$$


Получается, что она уменьшилась в N раз по сравнению с единичной моделью. Разумеется, это мечта, которой в реальной жизни быть не может. Всему виной наши допущения о несмещённости и нескоррелированности.

Нескоррелированность означает, что модели сильно не похожи друг на друга. Но здесь можно вспомнить про решающие деревья.

Дело в том, что переобученные решающие деревья, особенно на бутстррапированных датасетах, сильно не похожи друг на друга (чего не скажешь, например, про линейные модели), а значит близки к выполнению нашего условия нескоррелированности. Это место, где примитивные деревья превращаются в самый популярный ансамбль и одну из самых мощных моделей - в случайный лес. Про эту модель подробно написано в заметке Decision Tree and Random Forest.

Бэггингом (bootstrap aggregating) как раз и называется метод, который бутстрепит датасеты, переобучает на каждом из них модели и усредняет результаты.

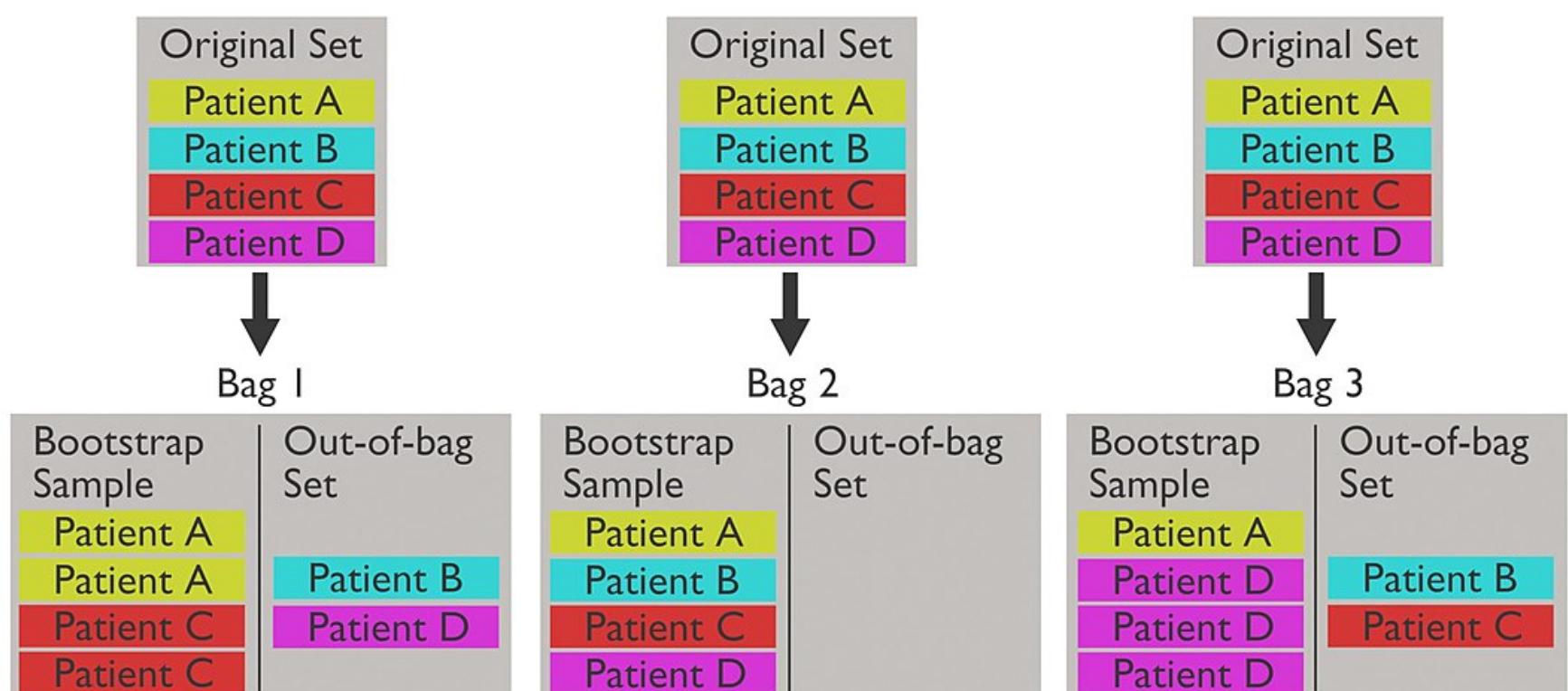


* Бэггинг на деревьях + RSM = Random Forest

Бэггинг можно параллелизировать и он позволяет использовать метод валидации out-of-bag (OOB).

Bagging uses subsampling with replacement to create training samples for the model to learn from. OOB error is the mean prediction error on each training sample x_i , using only the trees that did not have x_i in their bootstrap sample.[1]

Bootstrap aggregating allows one to define an out-of-bag estimate of the prediction performance improvement by evaluating predictions on those observations that were not used in the building of the next base learner.



The patients in each out-of-bag set can be used to test their respective models. The OOB sets can be aggregated into one dataset, but each sample is only considered out-of-bag for the trees that do not include it in their bootstrap sample.

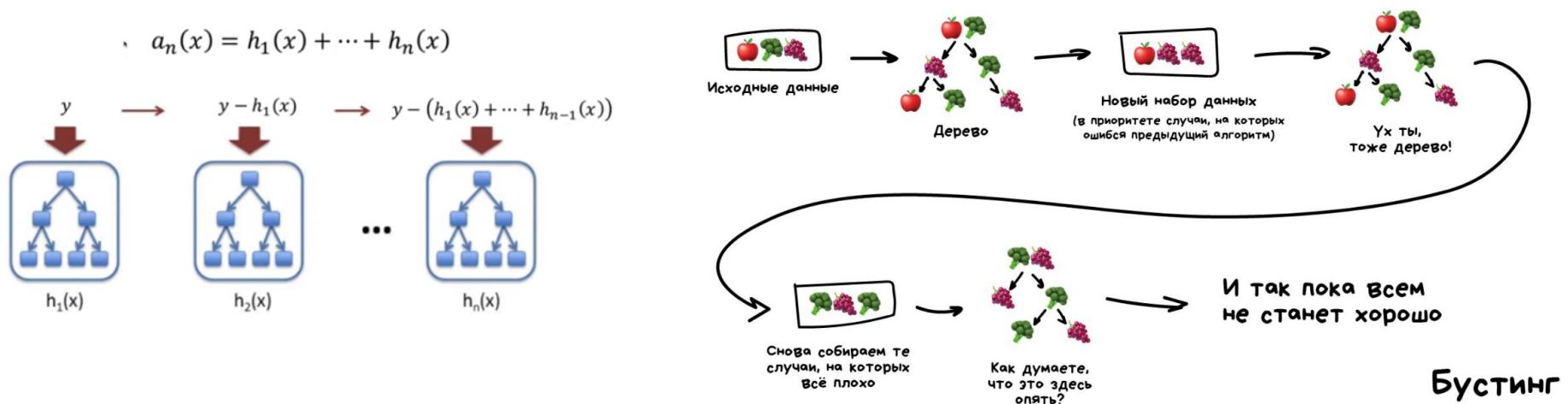
$$\text{OOB} = \sum_{i=1}^{\ell} L \left(y_i, \frac{1}{\sum_{n=1}^N [x_i \notin X_n]} \sum_{n=1}^N [x_i \notin X_n] b_n(x_i) \right)$$


Boosting

Этот метод точнее бэггинга, но не параллелируется на уровне всего ансамбля (модели теперь зависимые). Хотя можно распараллелить на уровне дерева: вспомним, что дерево строится жадно путем перебора всех признаков и порогов. Многие библиотеки сейчас умеют строить деревья на GPU.

Бустинг, а точнее рассмотренный далее градиентный бустинг, является одной из сильнейших моделей классического машинного обучения, когда речь заходит о работе с табличными данными.

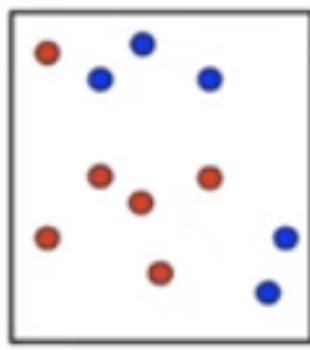
Бустинг - это ансамбль, в котором вместо независимого обучения моделей на каждом датасете модели обучаются последовательно так, чтобы каждая последующая модель исправляла ошибки предыдущих (уделяя особое внимание тем случаям, на которых ошиблась предыдущая).



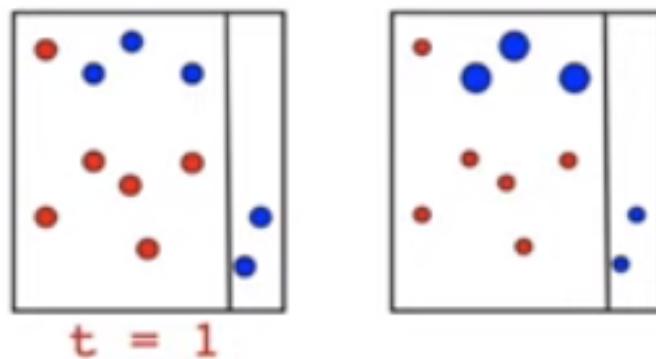
Как в бэггинге, мы делаем выборки из исходных данных, но теперь не совсем случайно. В каждую новую выборку мы берём часть тех данных, на которых предыдущий алгоритм отработал неправильно. То есть как бы доучиваем новый алгоритм на ошибках предыдущего.

Рассмотрим бинарную классификацию на решающих пнях (деревья с одним сплитом):



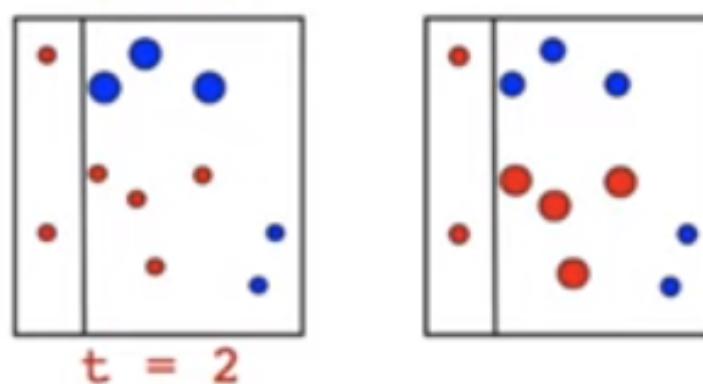


Первая модель (пень) задала некоторые веса каждому объекту и показала такой результат:

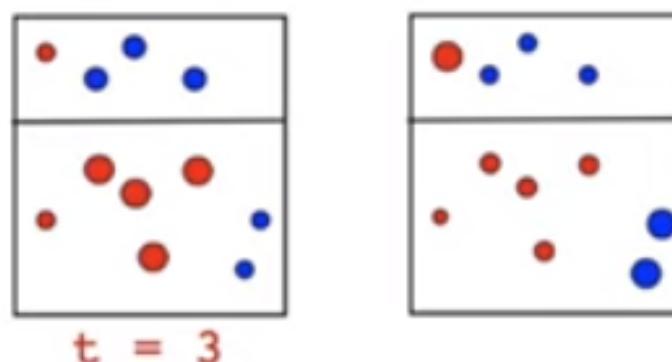


Как сказать следующему дереву, что три синие точки слева классифицированы неверно?

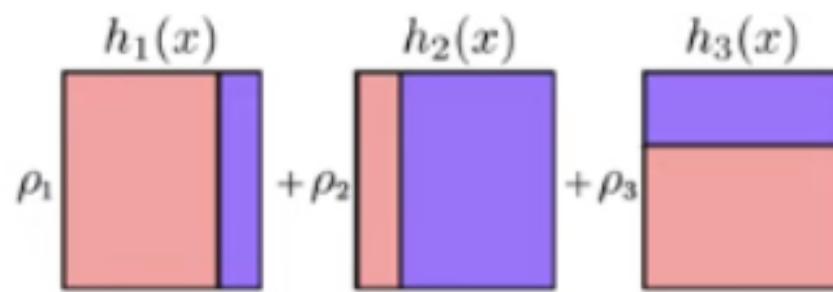
Дадим этим точкам больший вес при обучении. То есть, ошибка на них для следующей модели будет иметь больший штраф. Тогда вторая модель решит, что эти объекты обязательно нужно правильно классифицировать и сделает такой сплит:



Теперь классифицированы неправильно четыре красных объекта. Продолжаем:



Теперь объединяем эти модели с некоторыми коэффициентами:



$$\hat{f}_T(x) = \sum_{t=1}^T \rho_t h_t(x) = \begin{array}{|c|c|c|} \hline & \text{Red} & \text{Purple} \\ \hline \text{Red} & \bullet & \bullet \\ \hline \text{Red} & \bullet & \bullet \\ \hline \text{Red} & \bullet & \bullet \\ \hline \end{array}$$

Все объекты классифицированы верно, хотя мы применяли самые примитивные деревья с одним сплитом. Каждый пень внёс свой вклад - кто-то большой, кто-то малый, но все они зависели от решений предыдущих.

Представленный алгоритм бустинга (с увеличением весов) получил название **AdaBoost** (адаптивный бустинг). Он плохо подвержен переобучению, поэтому раньше применялся повсюду, пока не изобрели градиентный бустинг. Последний исправил проблему с выбросами и шумом: AdaBoost сильно чувствителен к ним, так как имеет экспоненциальную функцию потерь.

Пример работы AdaBoost на деревьях:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

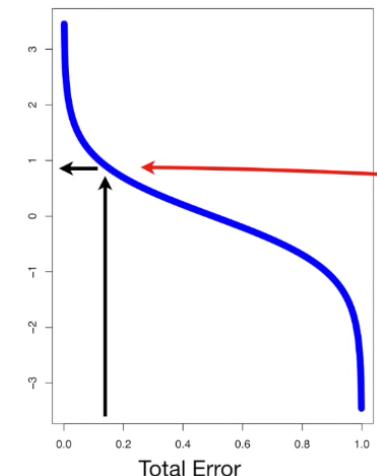
В начале каждый объект имеет одинаковый вес. Строится первый пень:



Посмотрим на его ошибку: просуммируем веса объектов, которые были классифицированы неверно. В нашем случае такой объект только один - в четвертой строке. Следовательно, ошибка пня - 1/8.

Теперь посчитаем вклад (выше обозначали буквой ρ) модели в общую классификацию:

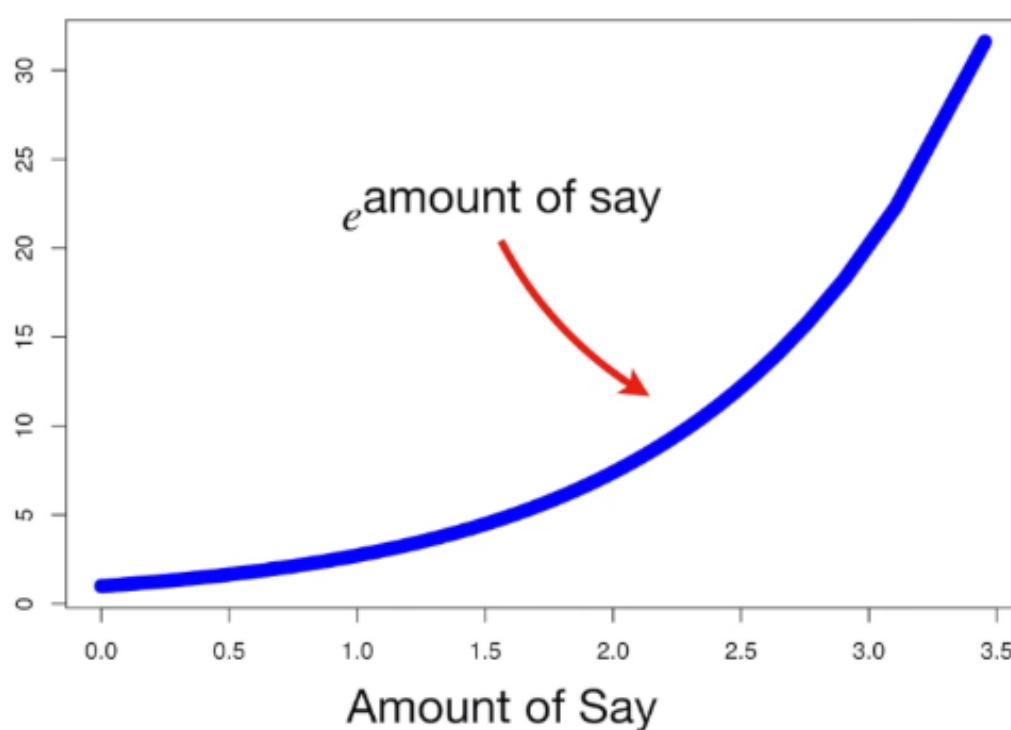
$$\text{Amount of Say} = \frac{1}{2} \log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right) = 0.97$$



Теперь, как уже было сказано, веса нужно обновить. Для неправильно классифицированного объекта вес должен быть увеличен:

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{\text{amount of say}}$$

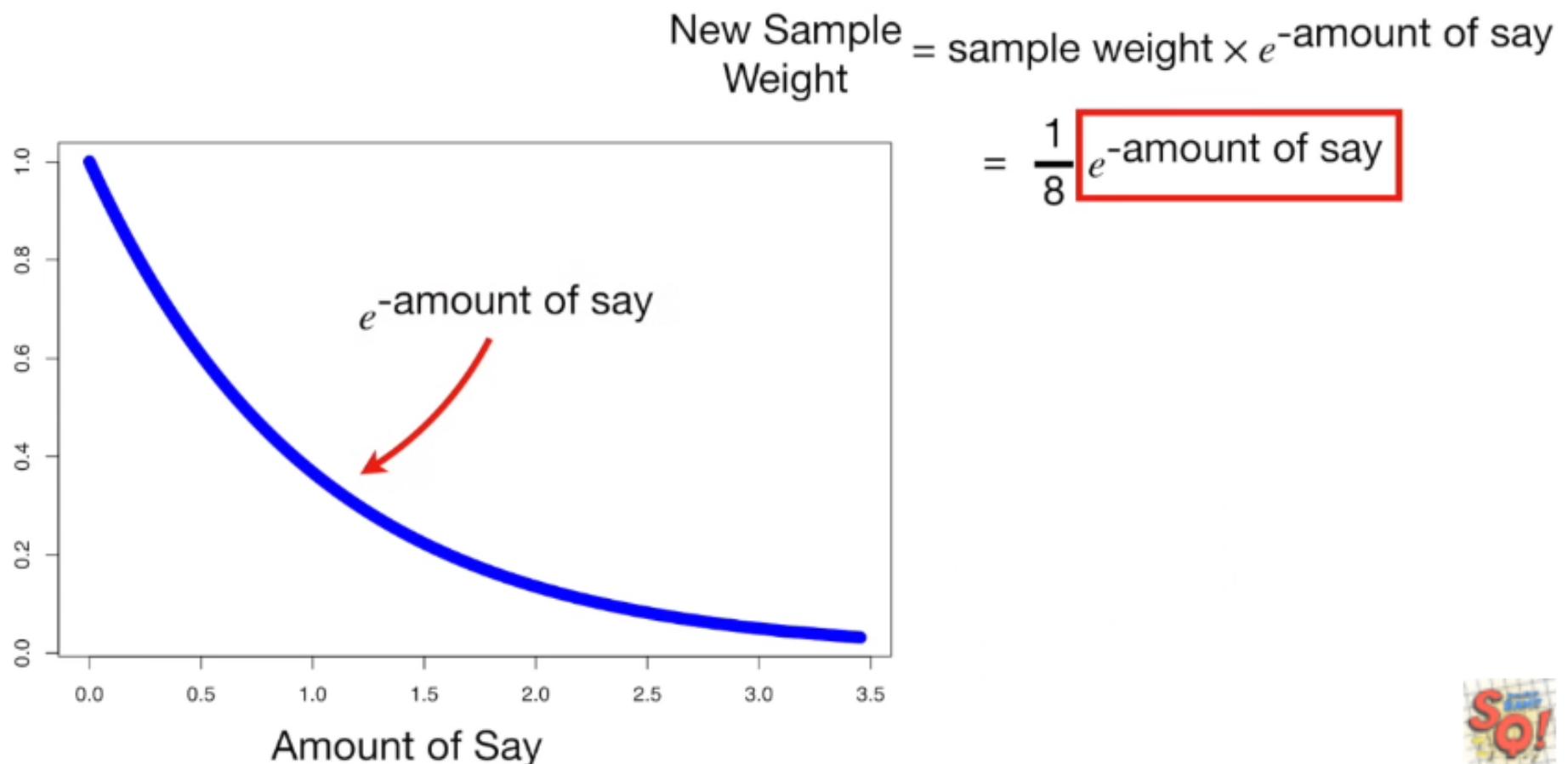


Если модель показывает плохой результат классификации, то вес увеличится слабо, ведь будет умножен на небольшое число. В нашем случае, для объекта в четвертой строке, вес обновится так:

$$\frac{1}{8} e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$

Новый вес больше старого, то есть мы говорим следующему пню обратить больше внимания на этот объект.

Для остальных объектов вес нужно уменьшить:



Здесь наоборот: если модель показывает плохой результат классификации, то вес объекта уменьшится слабо.

$$\frac{1}{8} e^{-0.97} = \frac{1}{8} \times 0.38 = 0.05$$

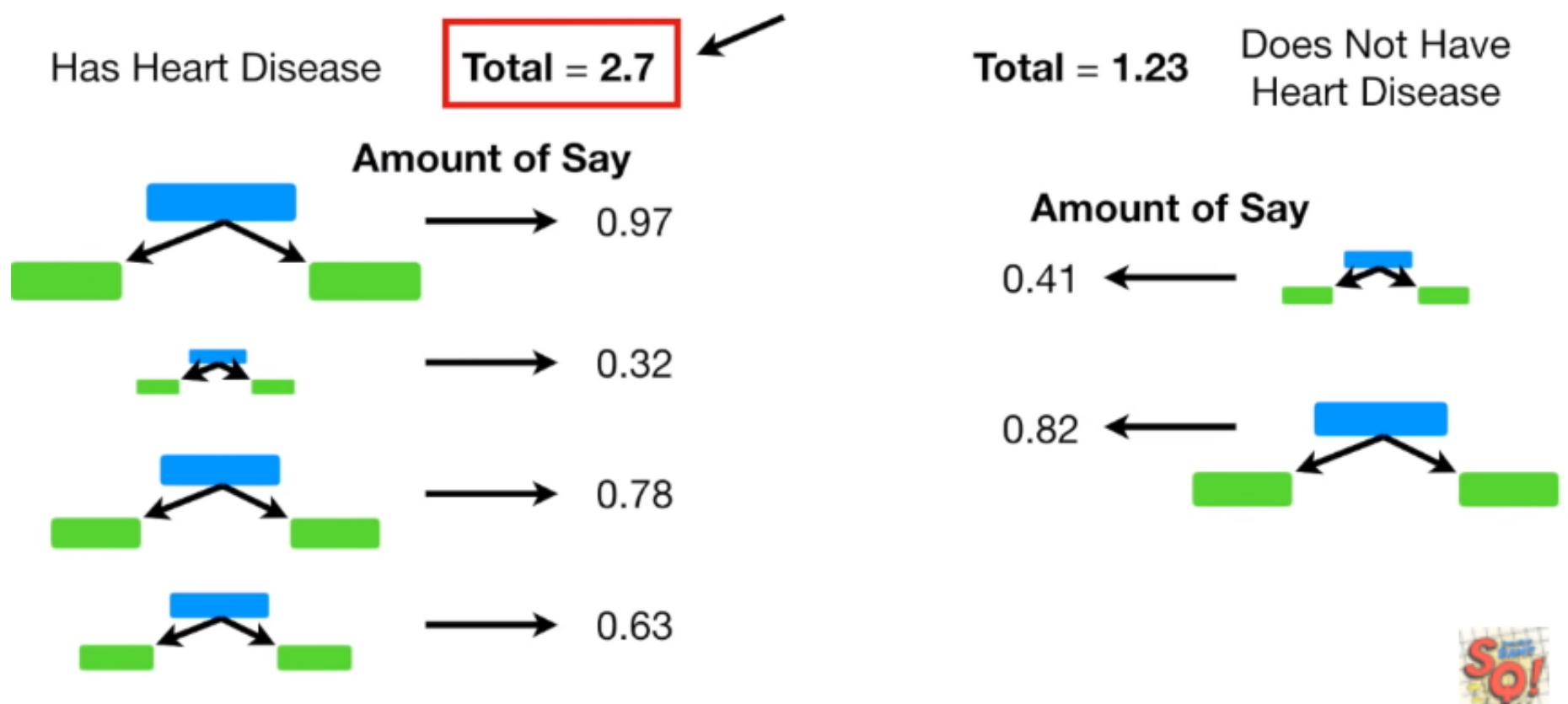
Новый вес меньше старого, то есть мы говорим следующему пню обращать меньше внимания на этот объект.

Складываем: $0.05 \times 7 + 0.33 = 0.68$

Нормализуем путём деления всех новых весов на 0.68:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Теперь строим следующий пень и продолжаем процесс. Когда все модели построены, разделяем их две группы и складываем коэффициенты α в каждой группе:



Результат классификации - первая группа, так как она имеет большую сумму голосов моделей.

<https://ru.wikipedia.org/wiki/Бустинг>

https://neerc.ifmo.ru/wiki/index.php?title=Бустинг,_AdaBoost

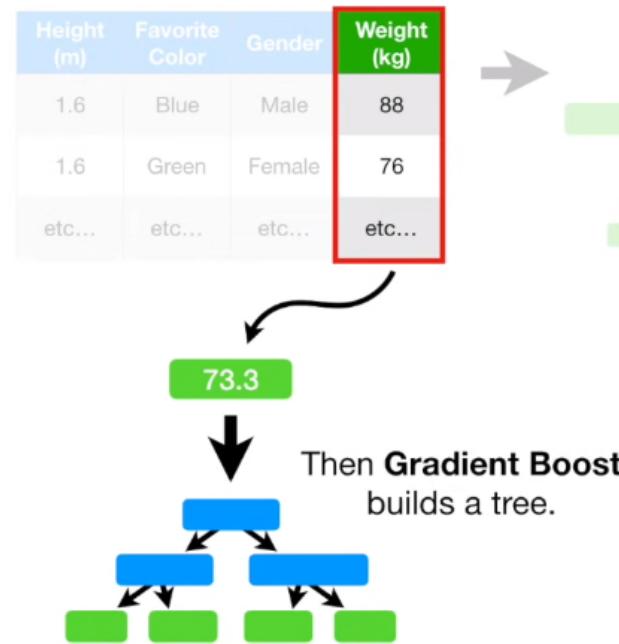
http://www.machinelearning.ru/wiki/index.php?title=Алгоритм_AdaBoost

Gradient boosting

Regression

В отличии от AdaBoost, градиентный бустинг на деревьях начинает работу с создания не дерева, а одного листа, который является initial guess для весов всех объектов. Когда мы

предсказываем вещественные значения, первый guess - это, обычно, среднее:



Как в AdaBoost, деревья основаны на ошибках предыдущего. Глубина всех деревьев в градиентном бустинге одинакова и задается заранее. Как правило, количество листьев задается в диапазоне от 8 до 32 в зависимости от данных. Ещё одно отличие - в AdaBoost все деревья имеют разный вес (коэффициент ро), а в градиентном бустинге все деревья изменяют свой вес на одно определенное значение (learning rate).

Разбираем на примере. Итак, есть датасет и дифференцируемая функция потерь:

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, and a differentiable **Loss Function** $L(y_i, F(x))$

Договоримся сразу, что мы задали для деревьев максимальное количество листьев - 4.

Самая популярная функция потерь для регрессии - это MSE, мы будем использовать MSE с коэффициентом 1/2. Коэффициент нужен для удобства дифференцирования - чтобы

избавиться от двойки:

$$\frac{d}{d \text{ Predicted}} \frac{1}{2} (\text{Observed} - \text{Predicted})^2$$

$$= \frac{2}{2} (\text{Observed} - \text{Predicted}) \times -1$$

$$= -(\text{Observed} - \text{Predicted})$$

Инициализируем константную модель:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$$

y_i - реальные наблюдения, гамма - предсказанные значения

Все лосс-функции для каждого наблюдения суммируются, затем берется такое предсказанное значение, которое минимизирует сумму.

Минимизацией занимается градиентный спуск, но на данном шаге нетрудно получить оптимальный параметр вручную, посчитав среднее: **71.2**. Сложите лоссы для всех объектов, возьмите производную, приравняйте к нулю - получите, что не удивительно, то же самое.

$F_0(x)$ - это начальная попытка предсказать вес всех людей (initial guess).

Теперь начинаем цикл по M деревьев и на каждом шаге строим дерево на основе ошибок предсказаний:

Step 2: for $m = 1$ to M :

(A) Compute $r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \dots, n$

Под ошибками здесь понимаются residuals: разности реальных (вес в таблице) и предсказанных весов:

Взгляните сами: дифференцируется лосс (т.е. вычисляется градиент), , что мы уже сделали выше, затем минус убирается - остается $y - F(x)$. Вместо $F(x)$ подставляется $F_{m-1}(x)$.

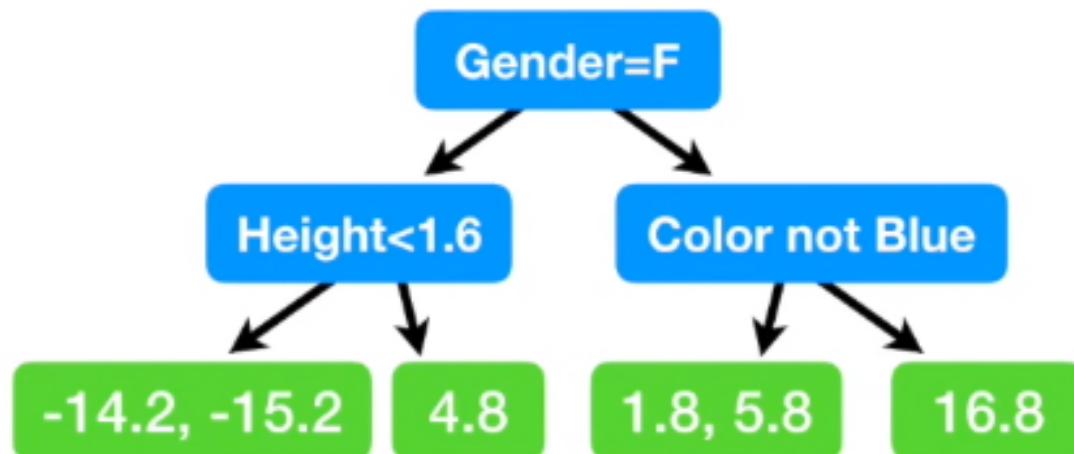
В нашем случае, на первом шаге $F_{m-1}(x) = F_0(x) = 71.2$. Выпишем residuals в отдельный столбец:

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

Таким образом, residual строится для каждого объекта і в каждом дереве m.

Теперь строим первое ($m = 1$) дерево регрессии, которое будет предсказывать Residuals по нашим синим признакам:

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm} , for $j = 1 \dots J_m$



В дереве $J_1 = 4$, имеем четыре terminal regions ($R_{11}, R_{21}, R_{31}, R_{41}$)

Далее необходимо определить output каждого листа:

(C) For $j = 1 \dots J_m$ compute $\gamma_{jm} = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

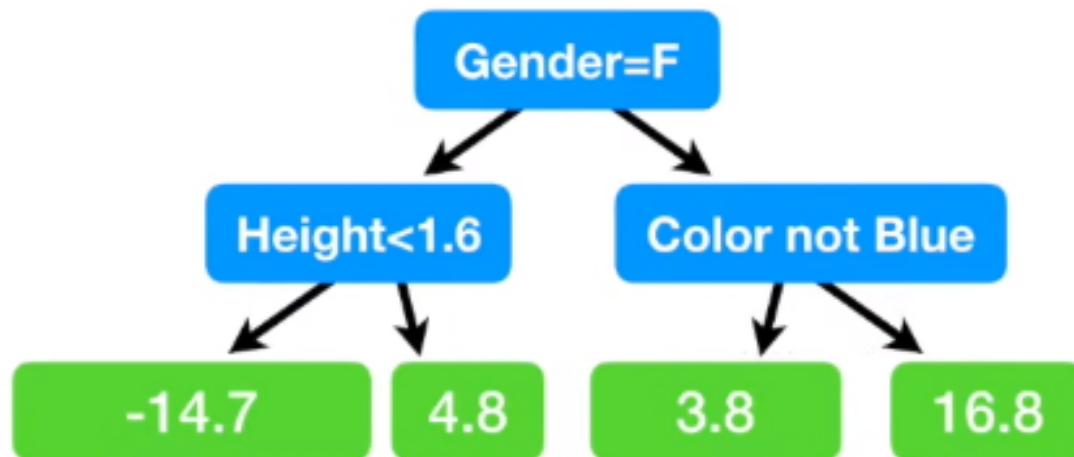
(опечатка: не R_{ij} , а R_{jm})

Для каждого листа считаем output, равный значению гаммы, которая минимизирует сумму аналогичным образом, единственное отличие здесь - мы принимаем во внимание предыдущее предсказание, прибавляя его к гамме, а суммирование происходит по значениям R_{ij} .

Посчитаем output для R31:

```
gamma_31 = argmin[
1/2 ( 73 - (71.2 + gamma) )^2 +
1/2 ( 77 - (71.2 + gamma) )^2
] = 3.8
```

Обратите внимание, output листов - это всегда среднее его residuals в случае с лоссом $1/2 * \text{MSE}$:



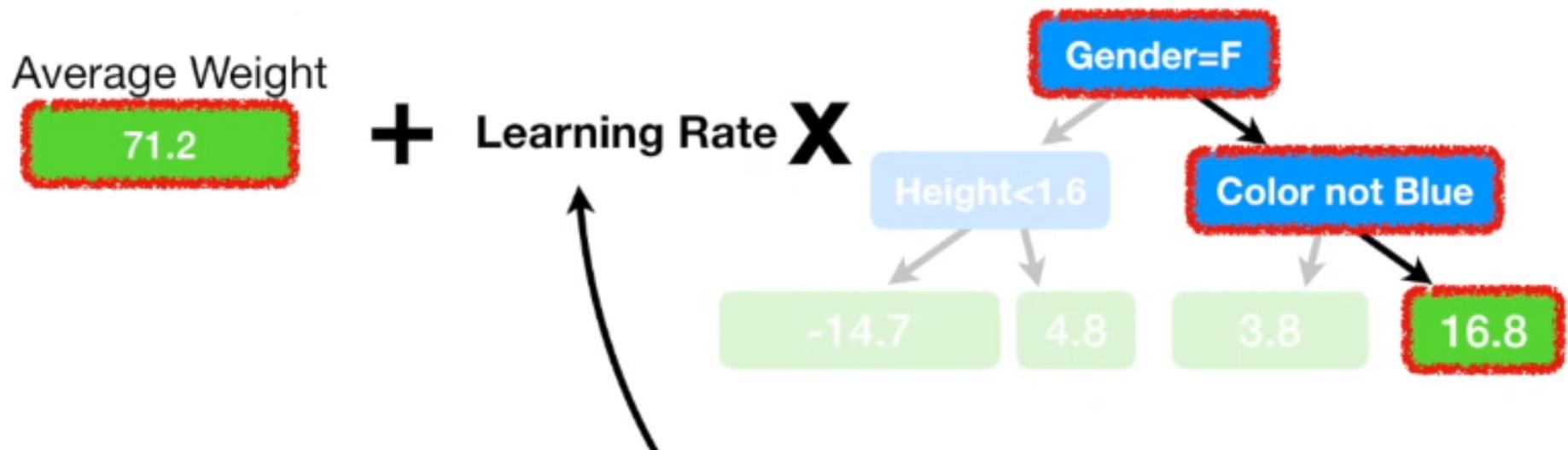
Теперь переходим к шагу внутри итерации цикла, на котором предсказания обновляются:

$$\nabla \quad (\mathbf{D}) \text{ Update } F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

ν - learning rate

В этой формуле мы складываем все output γ_{jm} 's для всех листов R_{jm} , в которых найден объект x , т.е. к каждому листу прибавляем предсказание на предыдущем шаге. На данном шаге для всех объектов предыдущее предсказание - это 71.2.

Возьмём последний лист. Его значение станет равным 88, что соответствует изначальному весу и указывает на переобучение. Чтобы исправить это, появляется learning rate, масштабирующий вклад дерева:



A small **Learning Rate** reduces the effect each tree has on the final prediction, and this improves accuracy in the long run.

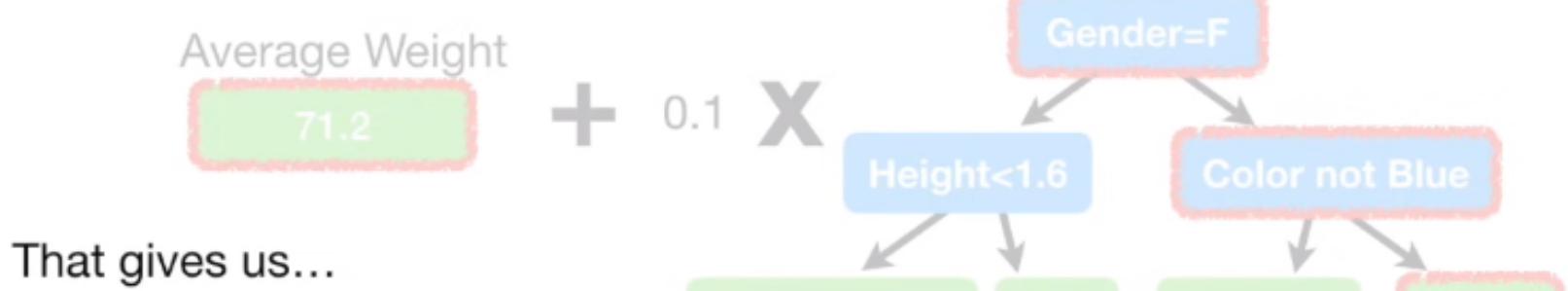
Пусть learning rate, например, 0.1 (он задается в самом начале). Тогда предсказание для первого объекта в датасете слегка улучшится в сравнении с предыдущим (71.2):

Now the **Predicted Weight** = $71.2 + (0.1 \times 16.8) = 72.9$

Новая модель $F_1(x)$ позволяет пройти по всем объектам и улучшить предсказания. Посчитаем residuals и сравним с теми, что были получены на предыдущем шаге:

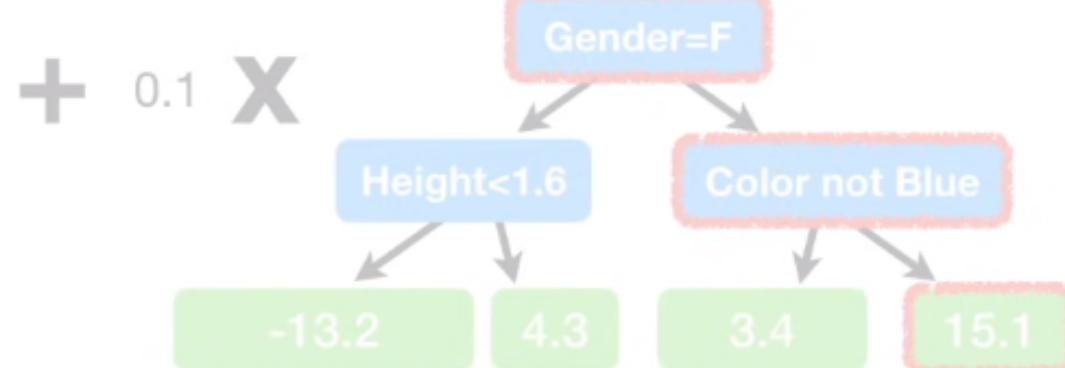
Residual	Residual
16.8	15.1
4.8	4.3
-15.2	-13.7
1.8	1.4
5.8	5.4
-14.2	-12.7

Они уменьшились, то есть дерево внесло свой вклад в улучшение качества модели. Завершилась первая итерация. Цикл переходит к $t = 2$ и строит дерево на новых residuals. Затем новое дерево добавляется в цепочку и таким образом создается модель $F_2(x)$:



$$71.2 + (0.1 \times 16.8) + (0.1 \times 15.1)$$

$$= 74.4$$



Получаем новые значения residuals для этой модели:

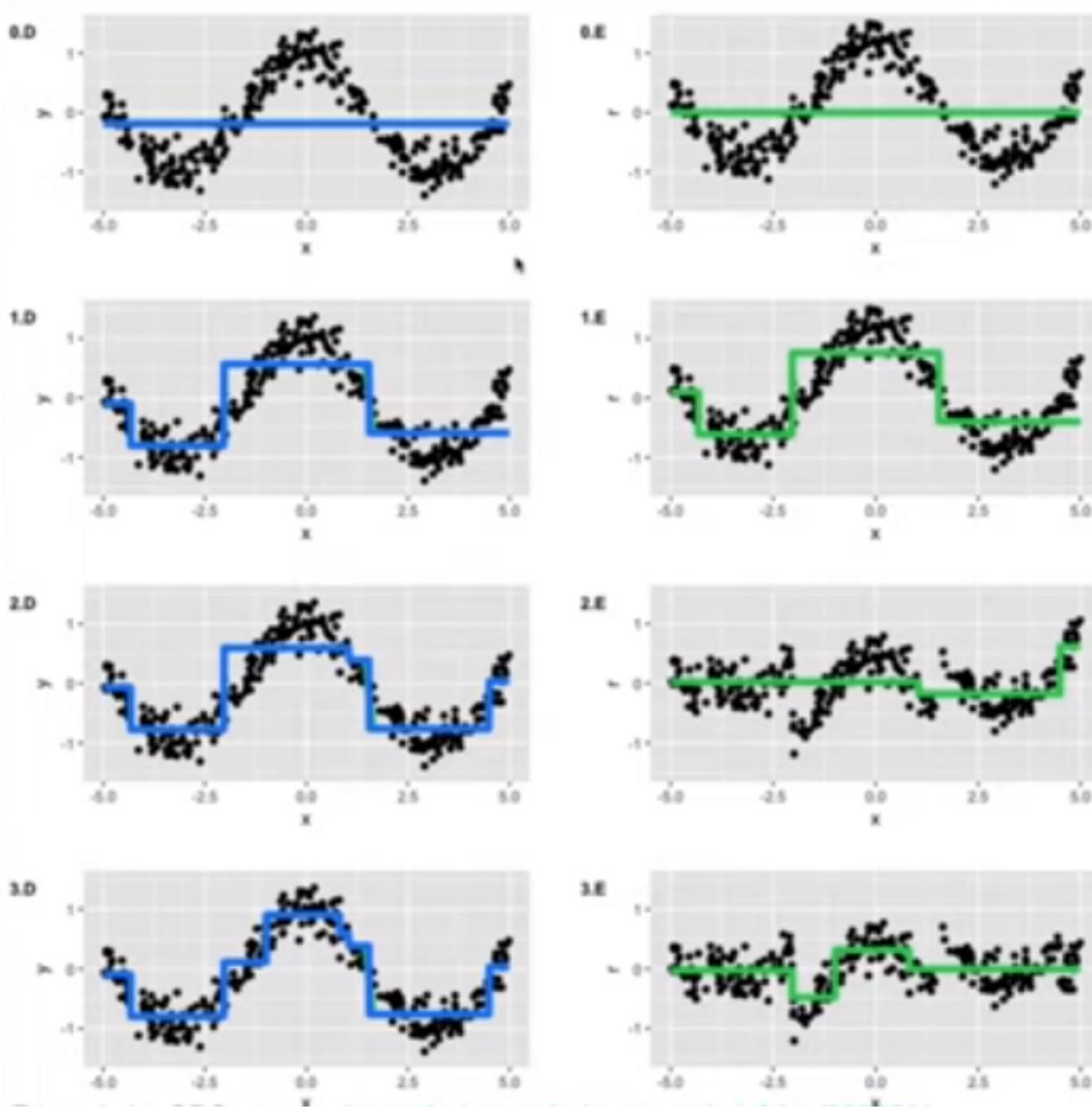
Residual	Residual	Residual
16.8	15.1	13.6
4.8	4.3	3.9
-15.2	-13.7	-12.4
1.8	1.4	1.1
5.8	5.4	5.1
-14.2	-12.7	-11.4

Продолжаем до тех пор, пока residuals не перестанут сильно изменяться или пока не будет достигнут заранее заданный максимум деревьев.

В итоге получим модель-ансамбль, которая, при получении новых данных, пройдет по всей этой цепочке деревьев с заранее определенным шагом и выдаст предсказания для регрессии. Решающей будет модель $F_M(x)$.

Таким образом, градиентный бустинг - это ансамбль, оптимизируемый с помощью градиентного спуска в пространстве моделей.

Вот так выглядит попытка описать функцию косинуса с шумом при помощи градиентного бустинга с MSE на деревьях глубины 2 и initial guess в виде среднего:



Справа показаны предсказания (residuals) последнего элемента в ансамбле, на которых обучалась каждая последующая модель.

В отличии от бэггинга, где мы переобучали независимые деревья, здесь нужна небольшая глубина деревьев. Дело в том, что переобученное дерево показывает маленькую ошибку, а алгоритм обучается именно на антиградиенте функции ошибки. Если функция ошибки близка к нулю, то у неё нет градиента. Если мы переобучим какую-либо модель в ансамбле, то получим маленькие ошибки и всем последующим моделям будет не на чем учиться.

Обратите внимание: бустинг может использовать любое семейство моделей, а не только деревья. Как правило, разные семейства не смешиваются, ведь так придется решать несколько оптимизационных задач. Деревья остаются самым популярным семейством.

Classification

Замечание: для понимания $\log(\text{odds})$ и $\log(\text{likelihood})$ см. заметку ‘Classification & Logistic Regression’.

Алгоритм градиентного бустинга в классификации ничем не отличается от регрессии.

Формулы на каждом шаге остаются прежними, изменяется лишь метод подсчёта `output` листов и лосс-функция.

Рассмотрим пример:

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Разберёмся с тем, как будет выглядеть лосс в случае классификации.

Вспомним, как в логистической регрессии мы считали log(likelihood):

Log(Likelihood of the Observed Data given the Prediction) =

$$\sum_{i=1}^N y_i \times \log(p) + (1 - y_i) \times \log(1 - p)$$

NOTE: The better the prediction, the larger the **log(likelihood)**, and this is why, when doing **Logistic Regression**, the goal is to *maximize* the **log(likelihood)**.

That means that if we want to use the **log(likelihood)** as a **Loss Function**, where smaller values represent better fitting models, then we need to multiply the **log(likelihood)** by **-1**.

$$-\left[y \times \log(p) + (1 - y) \times \log(1 - p) \right]$$



...and since a **Loss Function** sometimes only deals with one sample at a time, we can get rid of the summation...

Now we need to transform this equation, the **negative log(likelihood)**, so that it is a function of the predicted **log(odds)** instead of the predicted probability, p .

Выполним преобразования:

$$-\left[\text{Observed} \times \log(p) + (1 - \text{Observed}) \times \log(1 - p) \right]$$

1) $-\text{Observed} \times \log(p) - (1 - \text{Observed}) \times \log(1 - p)$

2) $-\text{Observed} \times \log(p) - \log(1 - p) + \text{Observed} \times \log(1 - p)$

3) $-\text{Observed} \times [\log(p) - \log(1 - p)] - \log(1 - p)$

4) $-\text{Observed} \times \log(\text{odds}) - \log(1 - p)$

5) $-\text{Observed} \times \log(\text{odds}) + \log(1 + e^{\log(\text{odds})})$

$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$

$$\begin{aligned} \log(1 - p) &= \log\left(1 - \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}\right) = \log\left(\frac{1 + e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}} - \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}\right) = \log\left(\frac{1}{1 + e^{\log(\text{odds})}}\right) \\ &= \log(1) - \log(1 + e^{\log(\text{odds})}) = -\log(1 + e^{\log(\text{odds})}) \end{aligned}$$

...into a function of the predicted **log(odds)**.

Получили лосс-функцию в финальном виде. Проверим дифференцируемость:

$$\frac{d}{d \log(\text{odds})} -\text{Observed} \times \log(\text{odds}) + \log(1 + e^{\log(\text{odds})}) =$$

As we will soon see, sometimes it's easier to use the function of the **log(odds)** and sometimes it's easier to use the function of the probability, p .

$$= -\text{Observed} + \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

$$= -\text{Observed} + p$$

Теперь у нас есть простой, удобный и дифференцируемый лосс, как в регрессии.

Переходим к начальной модели $F_0(x)$. Аналогично просуммируем лосс-функции для каждого объекта, возьмём производную ($d/d \log(\text{odds})$):

$$\begin{aligned}
 -1 \times \log(\text{odds}) + \log(1 + e^{\log(\text{odds})}) &\rightarrow -1 + \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}} \\
 -1 \times \log(\text{odds}) + \log(1 + e^{\log(\text{odds})}) &\rightarrow -1 + \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}} \\
 -0 \times \log(\text{odds}) + \log(1 + e^{\log(\text{odds})}) &\rightarrow -0 + \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}
 \end{aligned}$$

$\frac{d}{d \log(\text{odds})}$

Заменяем $\log(\text{odds})$ на предсказанную вероятность и приравниваем к нулю:

$$\begin{aligned}
 -1 + p + & \\
 \rightarrow -1 + p + & \\
 = 0 & \\
 \uparrow & \\
 -0 + p &
 \end{aligned}$$

Получаем $p = 2/3 = 0.67$, что является вероятностью 'Loves Troll 2 = Yes' ($4/6 = 2/3$), как и ожидалось.

Возвращаемся к $\log(\text{odds})$:

$$\begin{aligned}
 \log(\text{odds}) &= \log\left(\frac{2/3}{1 - 2/3}\right) & F_0(x) &= \log\left(\frac{2}{1}\right) \\
 & & &= 0.69
 \end{aligned}$$

По сути, мы посчитали аналог среднего для классификации.

Перейдем к построению дерева $m = 1$. На шаге подсчёта residuals находим антиградиент лосса в точке $F(x) = F_{m-1}(x) = F_0(x) = \log(2/1)$:

$$\frac{(\text{Observed} - \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}})}{(\text{Observed} - \frac{e^{\log(2/1)}}{1 + e^{\log(2/1)}})} \quad (\text{Observed} - \frac{e^{\log(2/1)}}{1 + e^{\log(2/1)}}) \quad (\text{Observed} - 0.67)$$

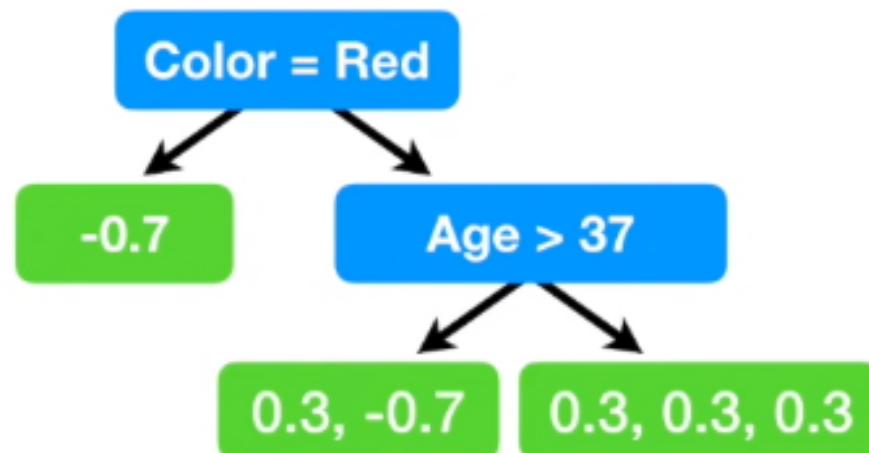
Теперь предсказанное значение - это вероятность принадлежности к классу, а observed - значение 1 или 0. Это значение больше 0.5, так что для начального предсказания имеет смысл

классифицировать всех как 'Yes'.

Округлим 0.67 до 0.7 для удобства и посчитаем residuals для всех объектов:

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3

Аналогично строим дерево регрессии. Договоримся ограничивать 3 листьями:



Теперь нужно посчитать output листов. Для этого мы аналогично находим argmin суммы лоссов. Но тут не всё так просто, как в регрессии.

Нам потребуется аналогично продифференцировать по гамма лосс:

$$L(y_1, F_{m-1}(x_1) + \gamma) = -y_1 \times [F_{m-1}(x_1) + \gamma] + \log(1 + e^{F_{m-1}(x_1) + \gamma})$$

Чтобы сделать это, мы аппроксимируем лосс при помощи полинома Тейлора второго порядка (почему второго?):

$$L(y_1, F_{m-1}(x_1) + \gamma) \approx L(y_1, F_{m-1}(x_1)) + \frac{d}{dF} (y_1, F_{m-1}(x_1))\gamma + \frac{1}{2} \frac{d^2}{dF^2} (y_1, F_{m-1}(x_1))\gamma^2$$

Если продифференцировать это выражение, приравнять к нулю, выполнить несколько преобразований и выразить гамму, то получится следующее:

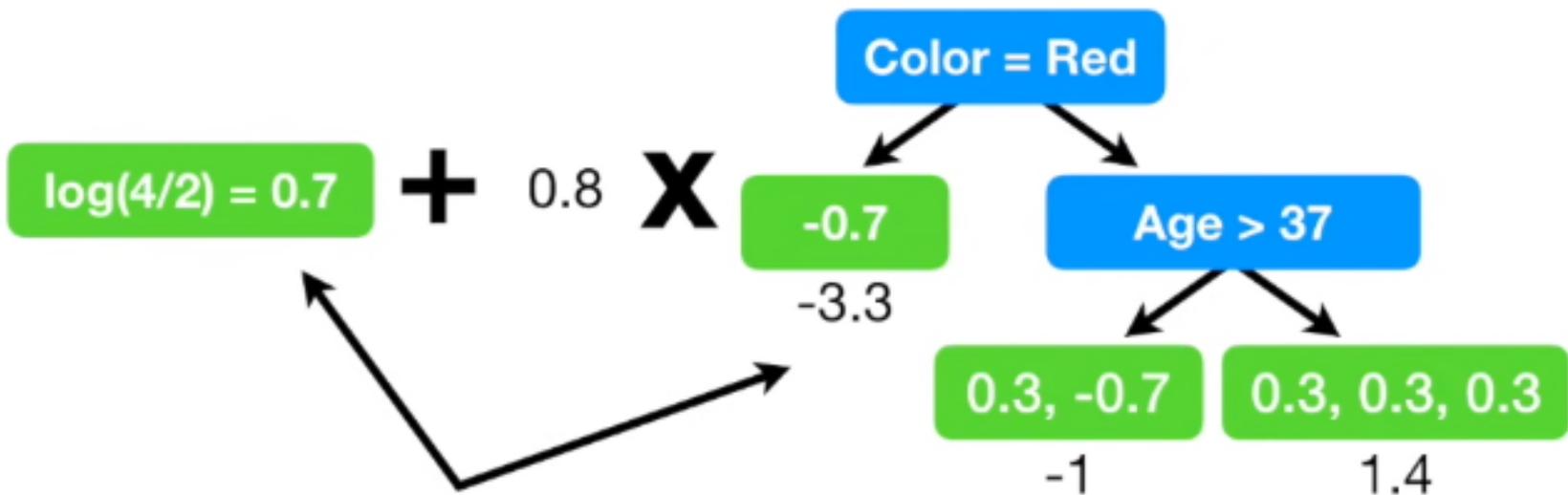
$$\gamma = \frac{\text{Residual}}{p \times (1 - p)}$$

Теперь можно легко считать выходные значения листов. Здесь p , напомню, равен 0.67, но мы округлили его до 0.7.

Если в листе несколько значений - тот же процесс нахождения гаммы (но уже от суммы лоссов) приведет к следующему:

$$\frac{\sum \text{Residual}_i}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)]}$$

Итак, находим выходные значения, выбираем какой-нибудь learning rate, например 0.8, и обновляем предсказания:



Now we are ready to update our **Predictions** by combining the initial leaf with the new tree.

Считаем $\log(\text{odds})$ prediction для первой строки датасета:

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes

$$0.7 + 0.8 * 1.4 = 1.8$$

Конвертируем в вероятность:

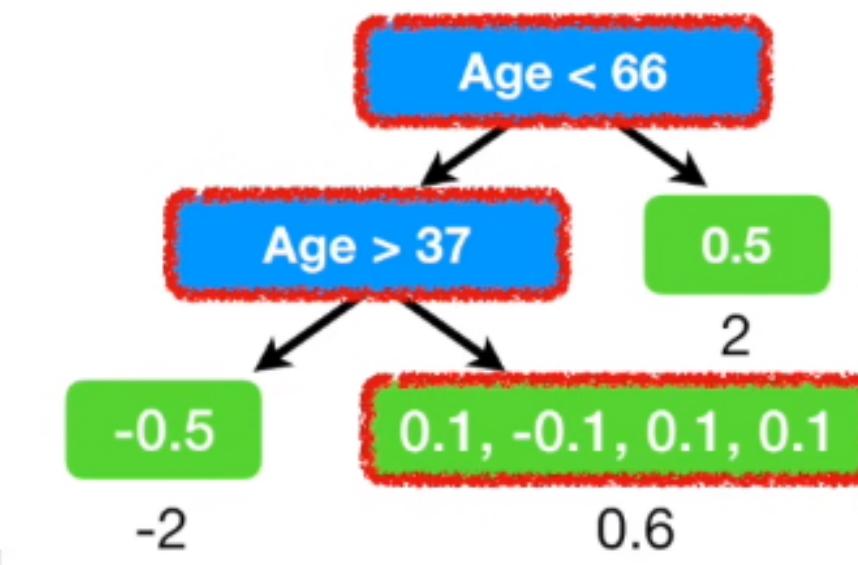
$$\text{Probability} = \frac{e^{1.8}}{1 + e^{1.8}} = 0.9$$

Это больше предыдущей вероятности (0.7) - предсказание улучшилось.

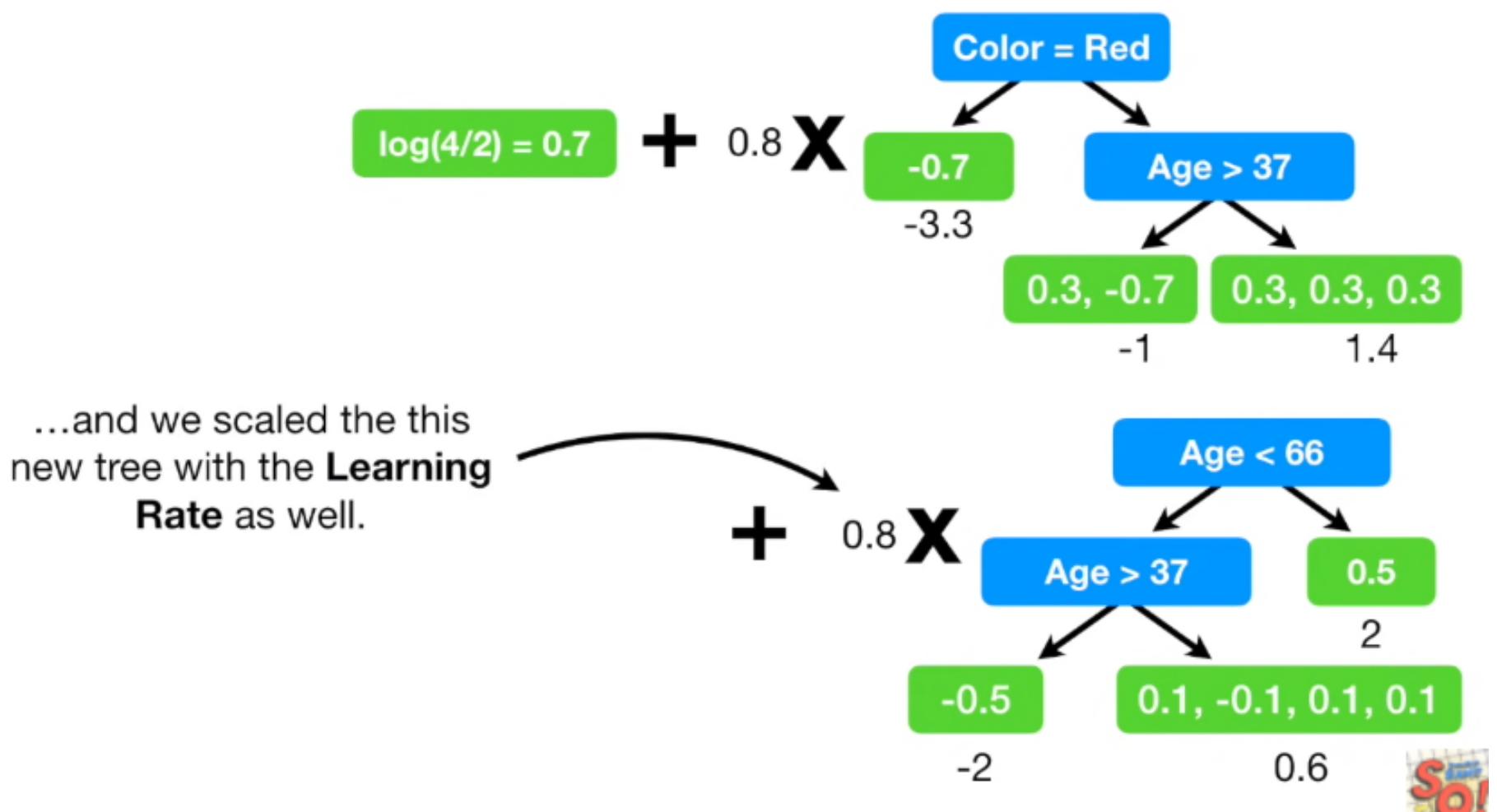
Считаем остальные предсказания и residuals для них:

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	0.5
No	44	Blue	No	0.5	-0.5
Yes	19	Red	No	0.1	-0.1
No	32	Green	Yes	0.9	0.1
No	14	Blue	Yes	0.9	0.1

Строим второе дерево, находим output каждого листа:



Обновляем предсказания:



И так далее, пока residuals не станут маленькими. В итоге сможем предсказывать:

Log(odds) Prediction

that someone **Loves Troll 2**: $= 0.7 + (0.8 \times 1.4) + (0.8 \times 0.6) = 2.3$

$$\text{Probability} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	25	Green	???

Полученная вероятность $- 0.9 > 0.5$ - относим к классу 'Yes'. Порог классификации, конечно, можно и другой задать - смотрите пост про метрики классификации.

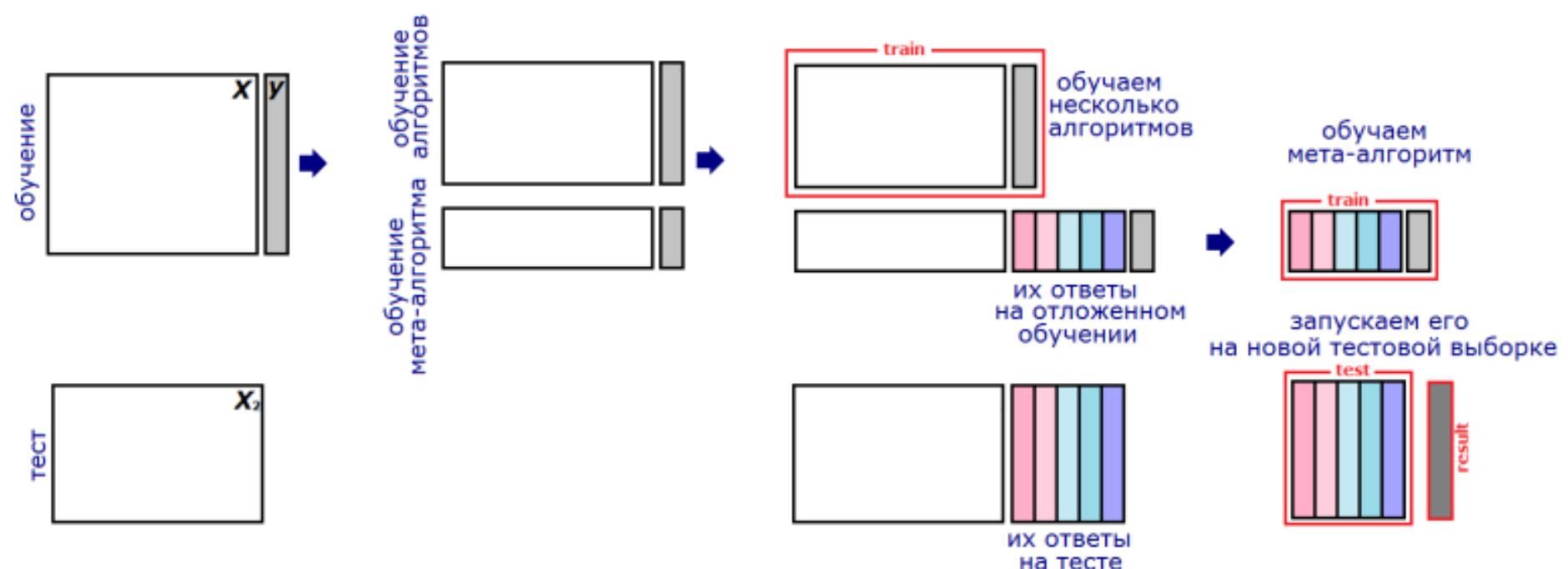
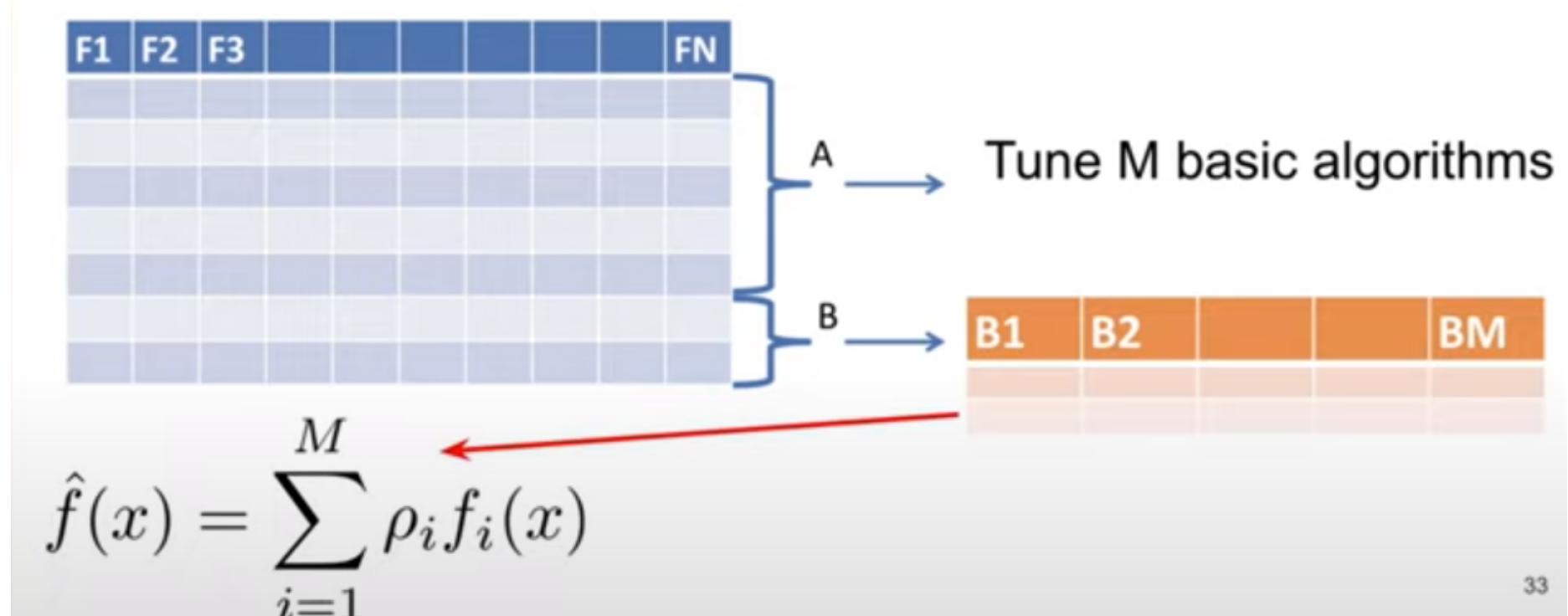
<https://proglib.io/p/reshaem-zadachi-mashinnogo-obucheniya-s-pomoshchyu-algoritma-gradientnogo-bustinga-2021-11-25>

Stacking and blending

Как построить ансамбль, если у нас несколько разных моделей? В бэггинге мы строили ансамбль моделей с одинаковыми весами, а затем усредняли. А что если какие-то модели работают лучше, чем другие? Тогда веса должны быть разными. Давайте для

ансамблирования вместо усреднения весов или голосования моделей использовать другой алгоритм - мета-алгоритм.

По сути, нам нужно построить линейную модель над предсказаниями всех моделей ансамбля. Чтобы обучить её, возьмем часть выборки для оптимизации весов (B), а оставшуюся часть (A) используем для обучения M базовых моделей:

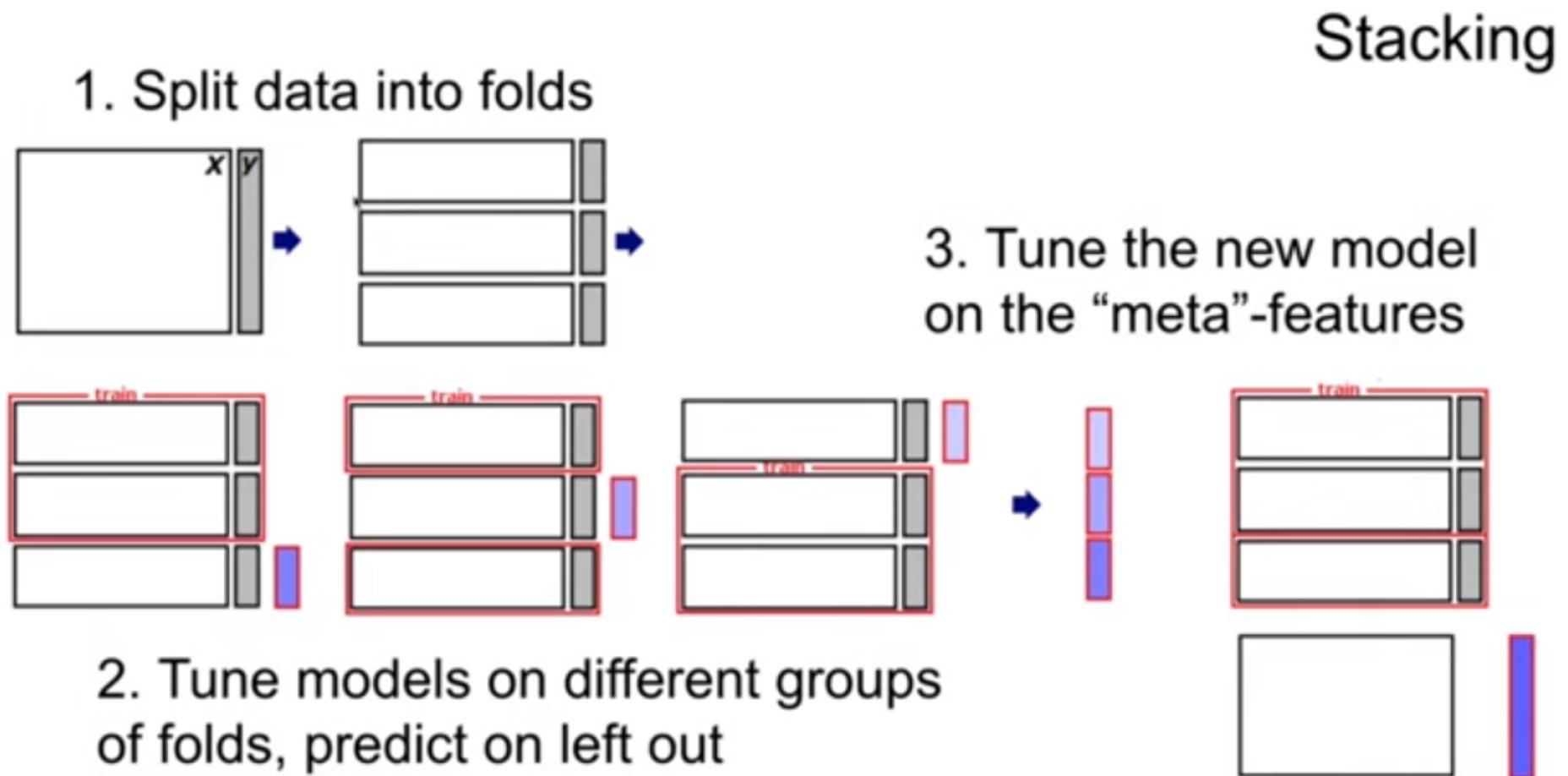


$$\sum_{i=1}^M \rho_i = 1, \quad \rho_i \in [0; 1] \quad \forall i$$

Обучающую выборку делят на две части. На первой обучаю базовые алгоритмы. Затем получают их ответы на второй части и на тестовой выборке. Понятно, что ответ каждого алгоритма можно рассматривать как новый признак (т.н. «метапризнак»). На метапризнаках второй части обучения настраивают метаалгоритм. Затем запускают его на метапризнаках теста и получают ответ.

Этот способ ансамблирования называется **блэндингом**. Чтобы улучшить предсказания, можно сделать разбиение выборки на А и В несколько раз, получить несколько блэндингов, а потом усреднить их предсказания.

Минус тут в том, что приходится отдавать часть данных на подбор весов. Получается, что ни базовые алгоритмы, ни метаалгоритм не используют всего объёма обучения (каждый — только свой кусочек). Это можно исправить с помощью **стекинга**:



Тут мы делим датасет на фолды, обучаем и тестируем базовые алгоритмы как в leave-one-out кросс-валидации, комбинируем предсказания, получая их, тем самым, для всего датасета. Новые предсказания используются в качестве новых признаков. Получилась обучающая выборка того же размера, но с новыми целевыми переменными.

Ясно, что совсем не делить обучение на подвыборки (т.е. обучить базовые алгоритмы на всей обучающей выборке и потом для всей выборки построить метапризнаки) нельзя: будет переобучение, поскольку в каждом метапризнаке будет «зашита» информация о значении целевого вектора (чтобы понять, представьте, что один из базовых алгоритмов — ближайший сосед). Поэтому **выборку разбивают на части (фолды), затем последовательно перебирая фолды обучают базовые алгоритмы на всех фолдах, кроме одного, а на оставшемся получают ответы базовых алгоритмов и трактуют их как значения соответствующих признаков на этом фолде. Для получения метапризнаков объектов тестовой выборки базовые алгоритмы обучают на всей обучающей выборке и берут их ответы на тестовой.**

Здесь тоже желательно реализовывать несколько разных разбиений на фолды и затем усреднить соответствующие метапризнаки (или ответы стекингов!). Но **самый главный недостаток** (классического) стекинга в том, что **метапризнаки на обучении** (пусть и полноценном — не урезанном) **и на teste разные**. Для объяснения возьмём какой-нибудь базовый алгоритм, например, гребневую регрессию. Мета-признак на обучающей выборке — это не ответы какого-то конкретного регрессора, он состоит из кусочков, которые являются ответами разных регрессий (с разными коэффициентами). А метапризнак на контрольной выборке вообще является ответом совсем другой регрессии, настроенной на всём обучении. В классическом стекинге могут возникать весьма забавные ситуации, когда какой-то

метапризнак содержит мало уникальных значений, но множества этих значений на обучении и teste не пересекаются!

Часто с **указанными недостатками борются обычной регуляризацией**. Если в качестве метаалгоритма используется гребневая регрессия, то в ней есть соответствующий параметр. А если что-то более сложное (например бустинг над деревьями), то **к метапризнакам добавляют нормальный шум**. Коэффициент с которым происходит добавка и будет здесь некоторым аналогом коэффициента регуляризации (это очень интересный приём — поиграйтесь на досуге).

- Pros:
 - Powerful ensembling method, if you know how to use it
 - Quite popular in ML-competitions
 - One might perform stacking on the meta-features dataset as well
- Cons:
 - Meta-features on each fold are actually predicted by different models
 - However, regularization usually helps
 - Hard to explain your model behaviour

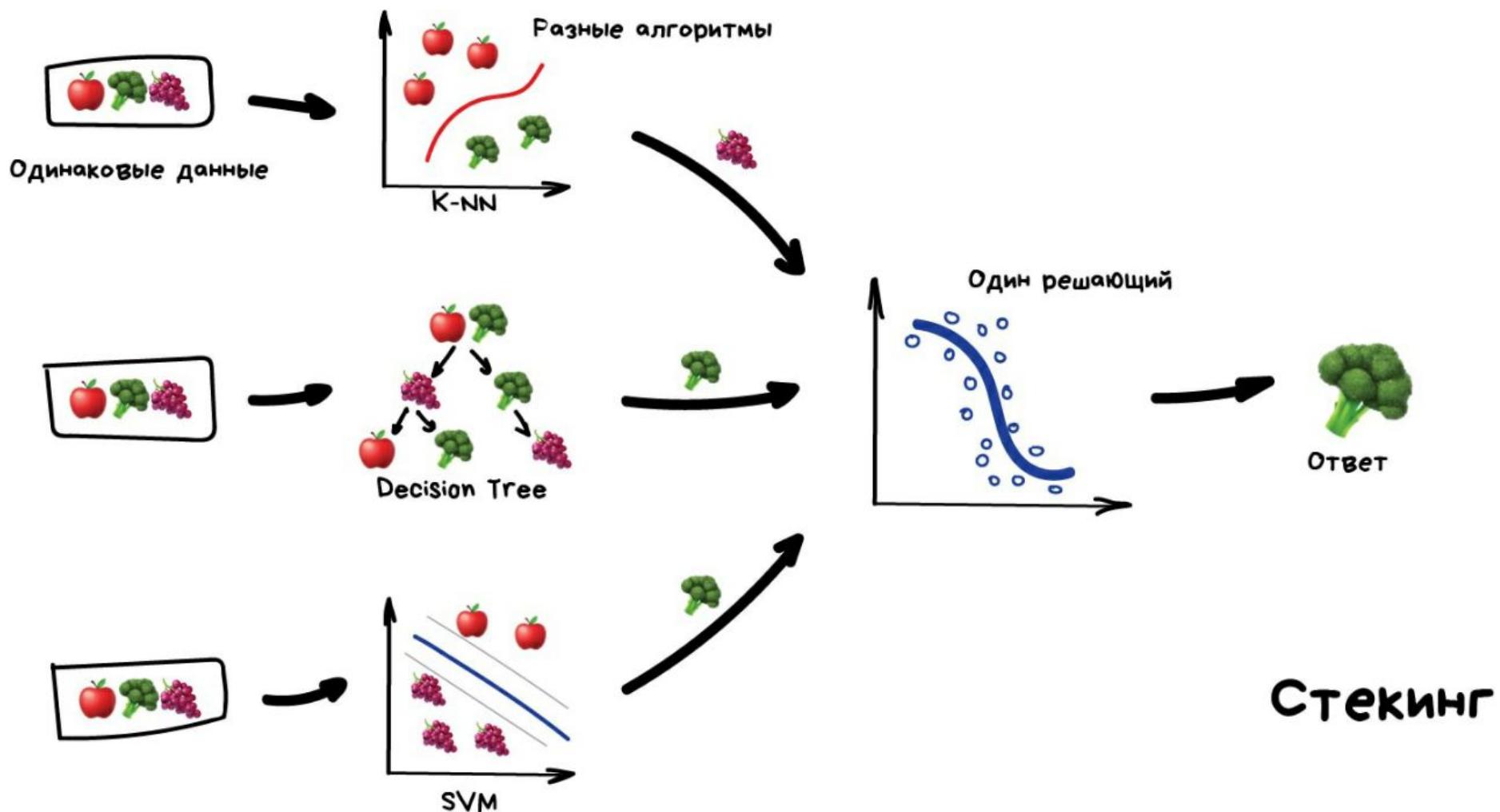
Кстати, **для стекинга нужны достаточно большие выборки**

(скажем, в двумерных модельных задачах он «начинает работать», когда число объектов измеряется десятками тысяч). На малых выборках он тоже может работать, но тут надо аккуратно подбирать базовые алгоритмы и, главное, метаалгоритм.

В отличие от бустинга и традиционного бэгинга при стекинге можно (и нужно!) **использовать алгоритмы разной природы** (например, гребневую регрессию вместе со случайнм лесом). Для формирования мета-признаков используют, как правило, регрессоры.

Но стоит помнить о том, что правильное применение стекинга — это не взять кучу разных алгоритмов и «состекать». Дело в том, что для разных алгоритмов нужны разные признаковые пространства. Скажем, если есть категориальные признаки с малым (3-4) числом категорий, то алгоритму **«случайный лес»** их можно подавать «как есть», а вот для регрессионных (ridge, log_reg) нужно предварительно выполнить **one-hot-кодировку**.

Естественное обобщение стекинга — сделать его многоуровневым, т.е. ввести понятие мета-мета-признака (и мета-мета-алгоритма) и т.д. Опять же, лучше воздержаться от этого при решении реальных бизнес-задач, а в спортивном анализе данных так часто делают.



+1 Ангелина про стеккинг неправильно написано. то, что там написано - это разновидность бэггинга. проверьте плиз

Стекинг выделяется двумя основными чертами: он может объединить в себе алгоритмы разной природы в качестве базовых. Например, взять метод опорных векторов (SVM), k-ближайших соседей (KNN) в качестве базовых и на основе их результатов обучить логистическую регрессию для классификации. Также стоит отметить непредсказуемость работы метамодели. Если в случае бэггинга и бустинга существует достаточно четкий и конкретный ансамблевый алгоритм (увидим далее), то здесь метамодель может с течением времени по-разному обучаться на входных данных.

Библиотеки градиентного бустинга

Помимо классической для машинного обучения [sklearn](#), для алгоритма градиентного бустинга существуют три наиболее используемые библиотеки:

XGBoost

[xgboost_theory_and_implementation.ipynb]

<https://neerc.ifmo.ru/wiki/index.php?title=XGBoost>

<https://www.youtube.com/playlist?list=PLblh5JKOoLULU0irPgs1SnKO6wqVjKUsQ>

- Регрессия с помощью алгоритма XGBoost.

CatBoost

<https://neerc.ifmo.ru/wiki/index.php?title=CatBoost>

Значок в профиль GitHub

LightGBM

Значок в профиль GitHub

Есть линейная регрессия и xgboost Есть матрица объект-признак где есть один супер информативный признак. Мы берём и создаём 10 копий этого признака + случайных шум. Что станет с моделями?

Есть случайный лес и есть бустинг. Оба уже настроены с какими-то параметрами Берём и увеличиваем число деревьев в 10 раз. Что с каждым из алгоритмов станет?