# Методы машинного обучения

*Лекция 9*

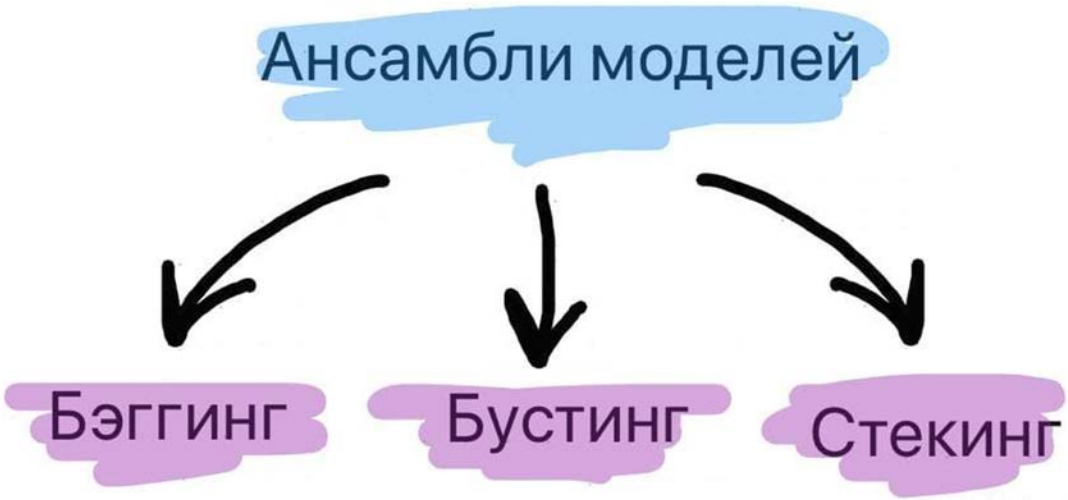
## Ансамбли классификаторов

# Ансамбли (Ensemble)

Ансамбли — это контролируемые алгоритмы обучения, которые комбинируют прогнозы из двух и более алгоритмов машинного обучения для построения более точных результатов. Результаты можно комбинировать с помощью голосования или усреднения. Первое зачастую применяется в классификации, а второе — в регрессии.

Существует 3 основных типа ансамблевых алгоритмов:

1. **Бэггинг.** Алгоритмы обучаются и работают параллельно на разных тренировочных наборах одного размера. Затем все они тестируются на одном наборе данных, а конечный результат определяется с помощью голосования.
2. **Бустинг.** В этом типе алгоритмы обучаются последовательно, а конечный результат отбирается с помощью голосования с весами.
3. **Стекинг (наложение).** Исходя из названия, этот подход состоит из двух уровней, расположенных друг на друге. Базовый представляет собой комбинацию алгоритмов, а верхний — мета-алгоритмы, основанные на базовом уровне.



# Оценка качества работы алгоритма

Предположим, что мы решаем задачу регрессии с квадратичной функцией потерь. При использовании квадратичной функции потерь для оценки качества работы алгоритма $a$ можно использовать следующий функционал:

$$Q(a) = \mathbb{E}_x \mathbb{E}_{X,\epsilon}[y(x,\epsilon) - a(x,X)]^2$$

где

- $X$ – обучающая выборка
- $x$ – точка из тестового множества
- $y(x,\epsilon) = f(x) + \epsilon$ – целевая зависимость, которую мы можем измерить с точностью до случайного шума $\epsilon$
- $a(x,X)$ – значение алгоритма, обученного на выборке $X$, в точке $x$
- $\mathbb{E}_x$– среднее по всем тестовым точкам
- $\mathbb{E}_{X,\epsilon}$ – среднее по всем обучающим выборкам $X$ и случайному шуму $\epsilon$.

Для $Q(a)$ существует разложение на три компонента:

**шум, смещение (отклонение), разброс (дисперсия).**

Это разложение называется **bias-variance decomposition**, и оно является одним из мощных средств для анализа работы ансамблей.

# Компромисс отклонение-дисперсия

**Компромисс отклонение-дисперсия** в статистике и машинном обучении — это свойство набора моделей предсказания, когда модели с меньшим отклонением от имеющихся данных имеют более высокую дисперсию на новых данных (то есть подвержены переобучению), и наоборот. **Компромисс отклонение-дисперсия** — конфликт при попытке одновременно минимизировать эти два источника ошибки, которые мешают алгоритмам обучения с учителем делать обобщение за пределами тренировочного набора.

- ***Смещение*** — это погрешность оценки, возникающая в результате ошибочного предположения в алгоритме обучения на имеющихся (тренировочных) данных. В результате большого смещения алгоритм может пропустить связь между признаками и выводом (недообучение).
- ***Дисперсия*** — это ошибочная чувствительность к малым отклонениям в тренировочном наборе. Высокая дисперсия говорит о том, что алгоритм пытается как-то трактовать случайный шум в тренировочном наборе, а не желаемый результат (переобучение).

**Разложение смещения-дисперсии** — это способ анализа ожидаемой ошибки обобщения алгоритма обучения для частной задачи сведением к сумме трёх членов — смещения, дисперсии и величины, называемой ***неустранимой погрешностью***, которая является результатом шума в самой задаче.

# Bias-variance decomposition (смещение, дисперсия, шум)

Существует представление $Q(a)$ в виде трёх компонент:

$$Q(a) = \mathbb{E}_x bias_X^2 a(x, X) + \mathbb{E}_x \mathbb{V}_X[a(x, X)] + \sigma^2$$

где

- $bias_X a(x, X) = f(x) - \mathbb{E}_X[a(x, X)]$ - **смещение** предсказания алгоритма в точке $x$, усреднённого по всем возможным обучающим выборкам, относительно истинной зависимости $f(x)$;
- $\mathbb{V}_X[a(x, X)] = \mathbb{E}_X[a(x, X) - \mathbb{E}_X[a(x, X)]]^2$ - **дисперсия (разброс)** предсказаний алгоритма в зависимости от обучающей выборки;
- $\sigma^2 = \mathbb{E}_x \mathbb{E}_\epsilon [y(x, \epsilon) - f(x)]^2$ неустранимый **шум** в данных.

Понятно, что с шумом ничего сделать нельзя, а вот смещение и дисперсия нас будут интересовать с точки зрения из минимизации.

# Теорема Кондорсе о присяжных

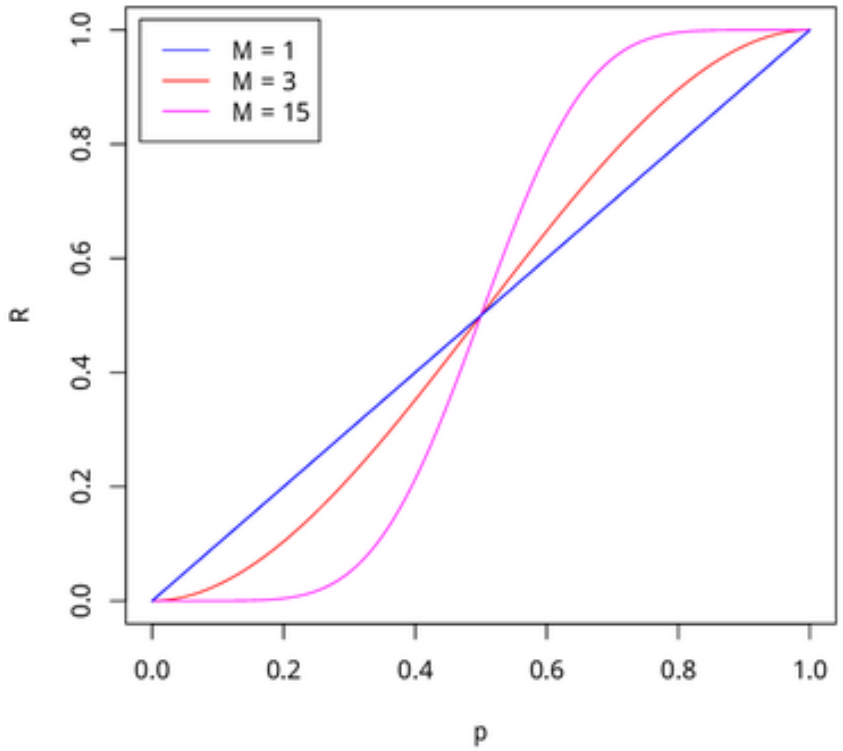
Если каждый член жюри присяжных имеет независимое мнение, и если вероятность правильного решения члена жюри больше 0.5, то тогда вероятность правильного решения присяжных в целом возрастает с увеличением количества членов жюри, и стремится к единице. Если же вероятность быть правым у каждого из членов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

Пусть M – количество присяжных, $p$ – вероятность правильного решения одного эксперта, $R$ – вероятность правильного решения всего жюри, $m$ – минимальное большинство членов жюри:

$$m = \left\lfloor \frac{M}{2} \right\rfloor + 1$$

Тогда

$$R = \sum_{i=m}^{M} C_M^i \, p^i (1-p)^{M-i}$$



# Бэггинг - Идея

Пусть обучающая выборка состоит из $n$ объектов. Выберем из неё $n$ примеров равновероятно, с возвращением. Получим новую выборку $X^1$, в которой некоторых элементов исходной выборки не будет, а какие-то могут войти несколько раз. С помощью некоторого алгоритма $b$ обучим на этой выборке модель $b_1(x) = b(x, X^1)$ . Повторим процедуру: сформируем вторую выборку $X^2$ из $n$ элементов с возвращением и с помощью того же алгоритма обучим на ней модель $b_2(x) = b(x, X^2)$. Повторив процедуру $k$ раз, получим $k$ моделей, обученных на $k$ выборках. Чтобы получить одно предсказание, усредним предсказания всех моделей:

$$a(x) = \frac{1}{k}(b_1(x) + \cdots + b_k(x))$$

Процесс генерации подвыборок с помощью семплирования с возвращением называется **бутстрепом** (**bootstrap**), а модели $b_1(x), \ldots, b_k(x)$ часто называют **базовыми алгоритмами** (моделями). Модель $a(x)$ называется **ансамблем** этих моделей.

# Бэггинг - Смещение

Будем считать, что когда мы берём матожидание по всем обучающим выборкам $X$, то в эти выборки включены также все подвыборки, полученные бутстрепом.

$$bias_X a(x, X) = f(x) - \mathbb{E}_X[a(x, X)] = f(x) - \mathbb{E}_X\left[\frac{1}{k}\sum_{i=1}^{k} b\left(x, X^i\right)\right] =$$

$$= f(x) - \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}_X\left[b\left(x, X^i\right)\right] = f(x) - \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}_X[b(x, X)] =$$

$$= f(x) - \mathbb{E}_X[b(x, X)] = bias_X b(x, X)$$

Получаем, что смещение ансамбля не изменилось по сравнению со средним смещением отдельных моделей.

# Бэггинг – Разброс (Дисперсия)

$$\mathbb{V}_X[a(x,X)] = \mathbb{E}_X[a(x,X) - \mathbb{E}_X[a(x,X)]]^2 =$$

$$= \mathbb{E}_X\left[\frac{1}{k}\sum_{i=1}^{k} b(x,X^i) - \mathbb{E}_X\left[\frac{1}{k}\sum_{i=1}^{k} b(x,X^i)\right]\right]^2 = \frac{1}{k^2}\mathbb{E}_X\left[\sum_{i=1}^{k}\left(b(x,X^i) - \mathbb{E}_X[b(x,X^i)]\right)\right]^2 =$$

$$= \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{E}_X\left(b(x,X^i) - \mathbb{E}_X[b(x,X^i)]\right)^2 +$$

$$+\frac{1}{k^2}\sum_{k_1\neq k_2}\mathbb{E}_X\left[\left(b(x,X^{k_1}) - \mathbb{E}_X[b(x,X^{k_1})]\right)\left(b(x,X^{k_2}) - \mathbb{E}_X[b(x,X^{k_2})]\right)\right] =$$

$$= \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{V}_X b(x,X^i) + \frac{1}{k^2}\sum_{k_1\neq k_2} cov\left(b(x,X^{k_1}), b(x,X^{k_2})\right)$$

Если предположить, что базовые алгоритмы некоррелированы, то:

$$\mathbb{V}_X[a(x,X)] = \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{V}_X b(x,X^i) = \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{V}_X b(x,X) = \frac{1}{k}\mathbb{V}_X b(x,X)$$

**Дисперсия композиции в $k$ раз меньше дисперсии отдельного алгоритма!**

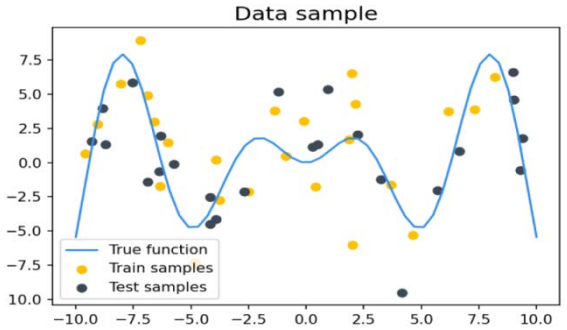
# Пример: бэггинг над решающими деревьями

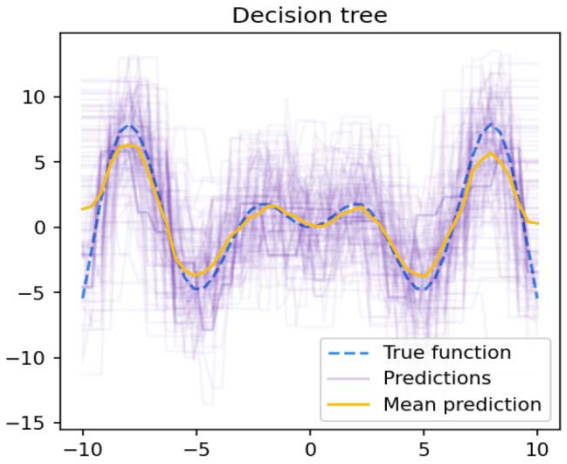
Пусть наша целевая зависимость $f(x)$ задаётся как $f(x) = x \cdot \sin(x)$ и к ней добавляется нормальный шум $\epsilon \sim N(0,9)$.

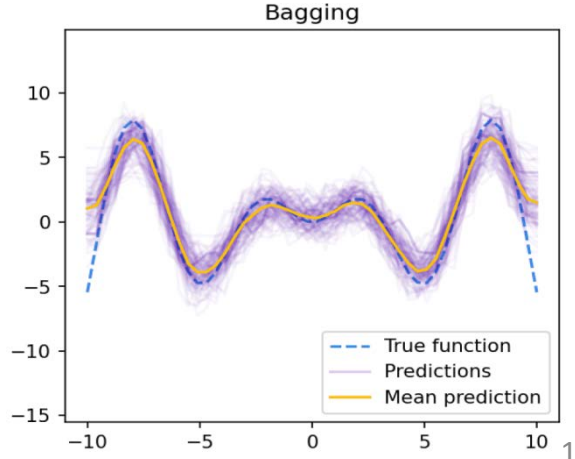
Обучим 100 решающих деревьев глубины 7 на различных случайных выборках размера 20. Возьмём также бэггинг над 10 решающими деревьями глубины 7 в качестве базовых классификаторов и тоже 100 раз обучим его на случайных выборках размера 20.

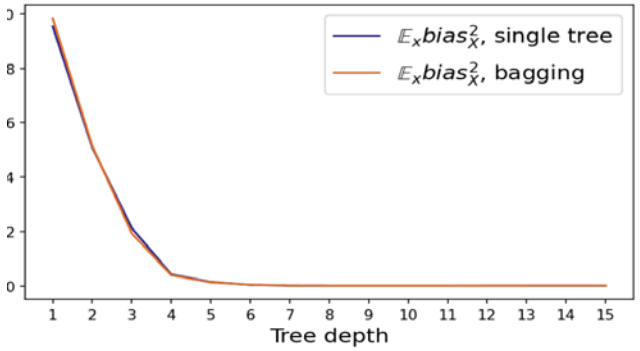
Общая дисперсия предсказаний в зависимости от обучающего множества у бэггинга значительно ниже, чем у отдельных деревьев, а в среднем предсказания деревьев и бэггинга не отличаются.

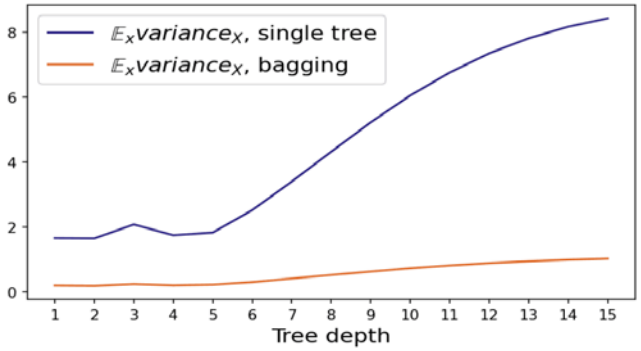
Дисперсия уменьшилась практически в $k$ раз, что равняется числу базовых алгоритмов, которые бэггинг использовал для предсказания.



# Random Forest (Случайный лес)

Мы делали предположение, что базовые алгоритмы некоррелированы. Однако в реальной жизни добиться этого проблематично: ведь базовые алгоритмы учат одну и ту же зависимость на пересекающихся выборках. В реальности достаточно, чтобы алгоритмы были в некоторой степени не похожи друг на друга. Развитие идеи бэггинга для решающих деревьев — **случайный лес**.

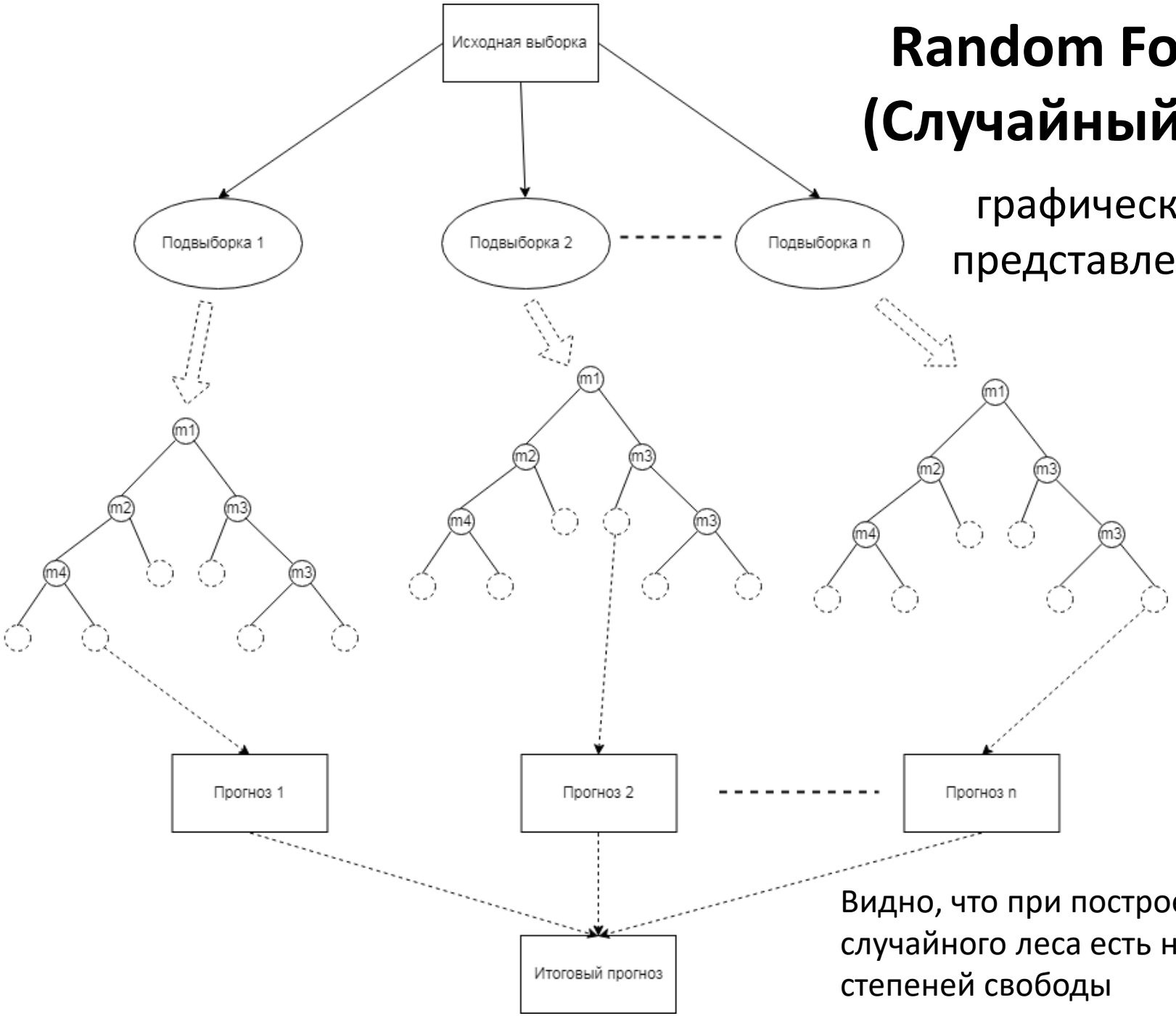
Строим ансамбль алгоритмов по следующей схеме:

1. Для построения *i*-го дерева:
   - Сначала, как в обычном бэггинге, из обучающей выборки $X$ выбирается с возвращением случайная подвыборка $X^i$ того же размера, что и $X$.
   - В процессе обучения каждого дерева **в каждой вершине** случайно выбираются $n < N$ признаков, где $N$ — полное число признаков (метод случайных подпространств), и среди них ищется оптимальное разбиение. Такой приём позволяет управлять степенью скоррелированности базовых алгоритмов.
2. Чтобы получить предсказание ансамбля на тестовом объекте, усредняем отдельные ответы деревьев (для регрессии) или берём самый популярный класс (для классификации).

**Random Forest (случайный лес)** — комбинация бэггинга и метода случайных подпространств над решающими деревьями.

# Random Forest (Случайный лес)

графическое представление




Видно, что при построении случайного леса есть несколько степеней свободы

# Основные параметры при построении случайного леса

- **Глубина деревьев** в случайном лесу

Ошибка модели (на которую мы можем повлиять) состоит из смещения и разброса. Разброс уменьшаем с помощью процедуры бэггинга. У неглубоких деревьев малое число параметров, то есть дерево способно запомнить только верхнеуровневые статистики обучающей подвыборки (низкая дисперсия, высокое смещение). Глубокие деревья запоминают подвыборку подробно (высокая дисперсия, низкое смещение).

**Вывод: используем глубокие деревья.**

- **Число признаков** при разбиении вершины в процессе обучения

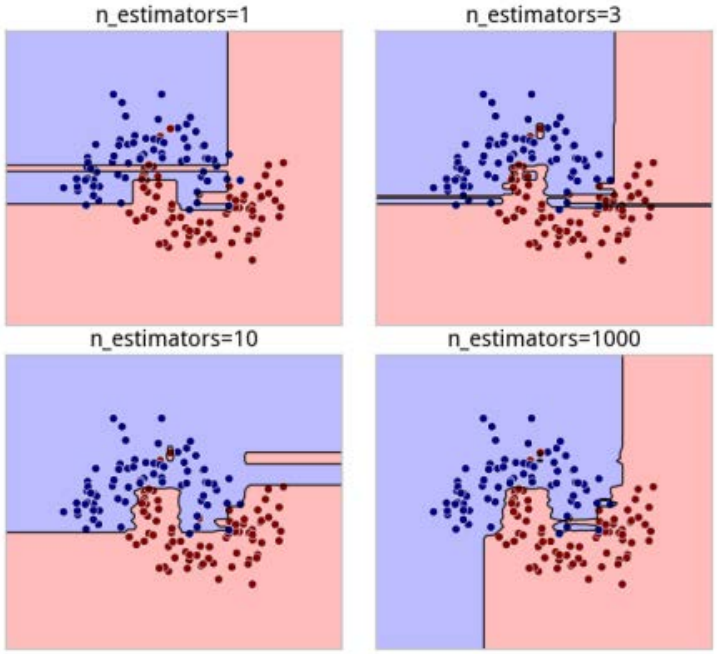
Управляет качеством случайного леса. Чем больше признаков, тем больше корреляция между деревьями и тем меньше эффект от ансамблирования. Чем меньше признаков, тем слабее деревья. Практическая рекомендация — брать корень из числа всех признаков для классификации и треть признаков для регрессии.

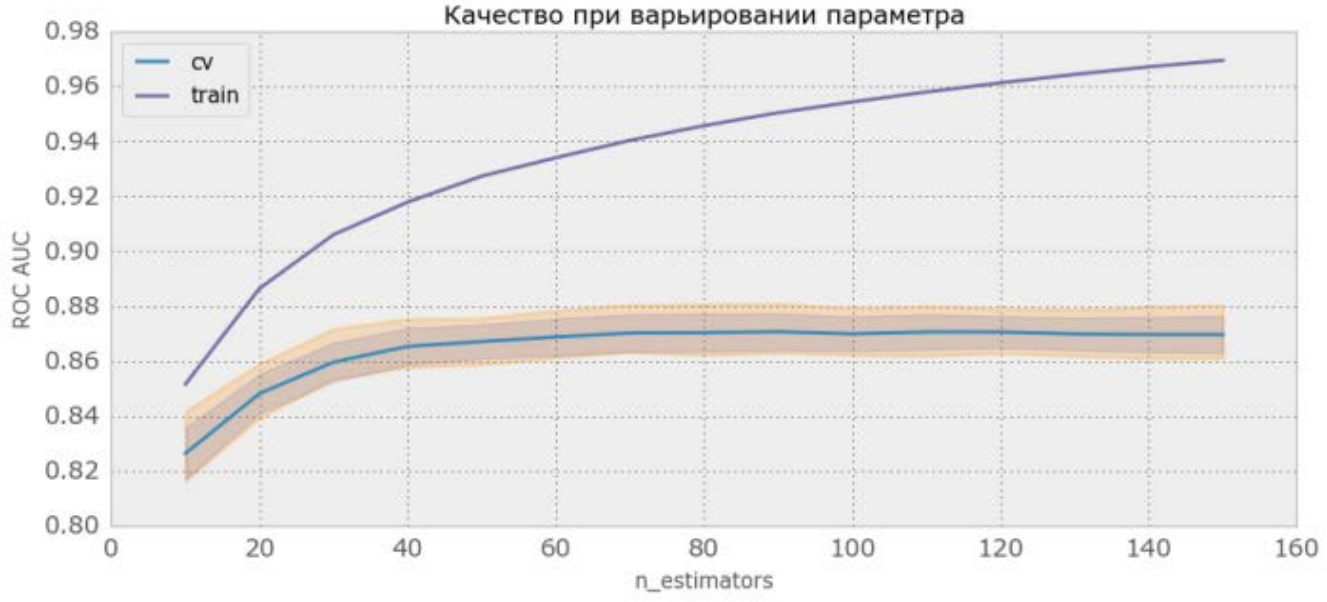
- **Число деревьев** в случайном лесу

Можно построить график ошибки от числа деревьев и ограничить размер леса в тот момент, когда ошибка перестанет значимо уменьшаться.

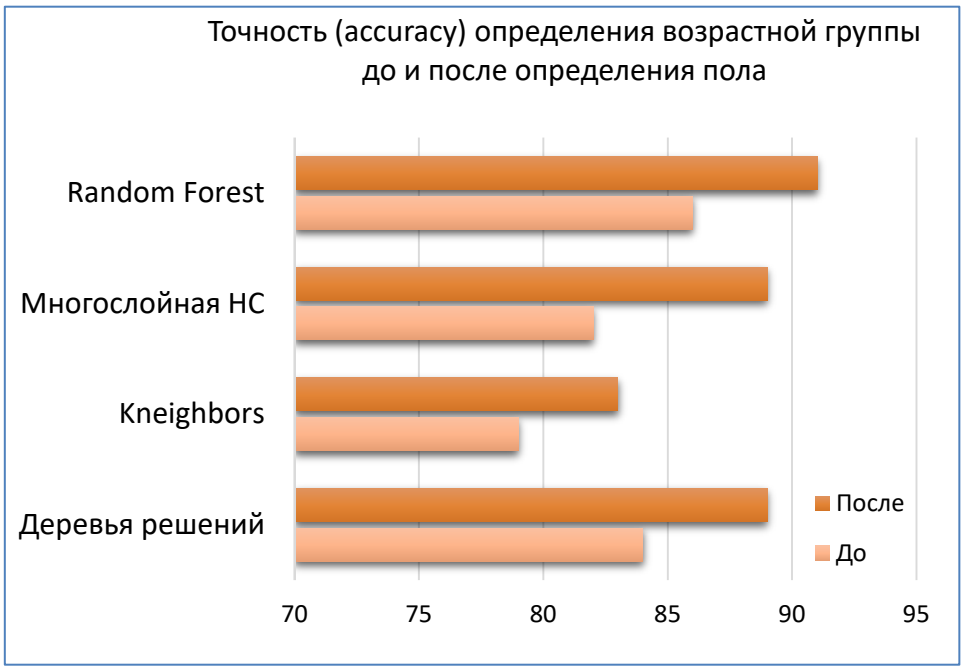
# Число деревьев в случайном лесу

Чем больше деревьев, тем лучше качество, но время настройки и работы также пропорционально увеличиваются. Также можно обратить внимание, что часто при увеличении числа деревьев качество на обучающей выборке повышается (может даже доходить до 100%), а качество на тесте выходит на асимптоту.



# Пример: Оценка биологического возраста на основе комплексного анализа морфометрических данных в задачах судебной медицины

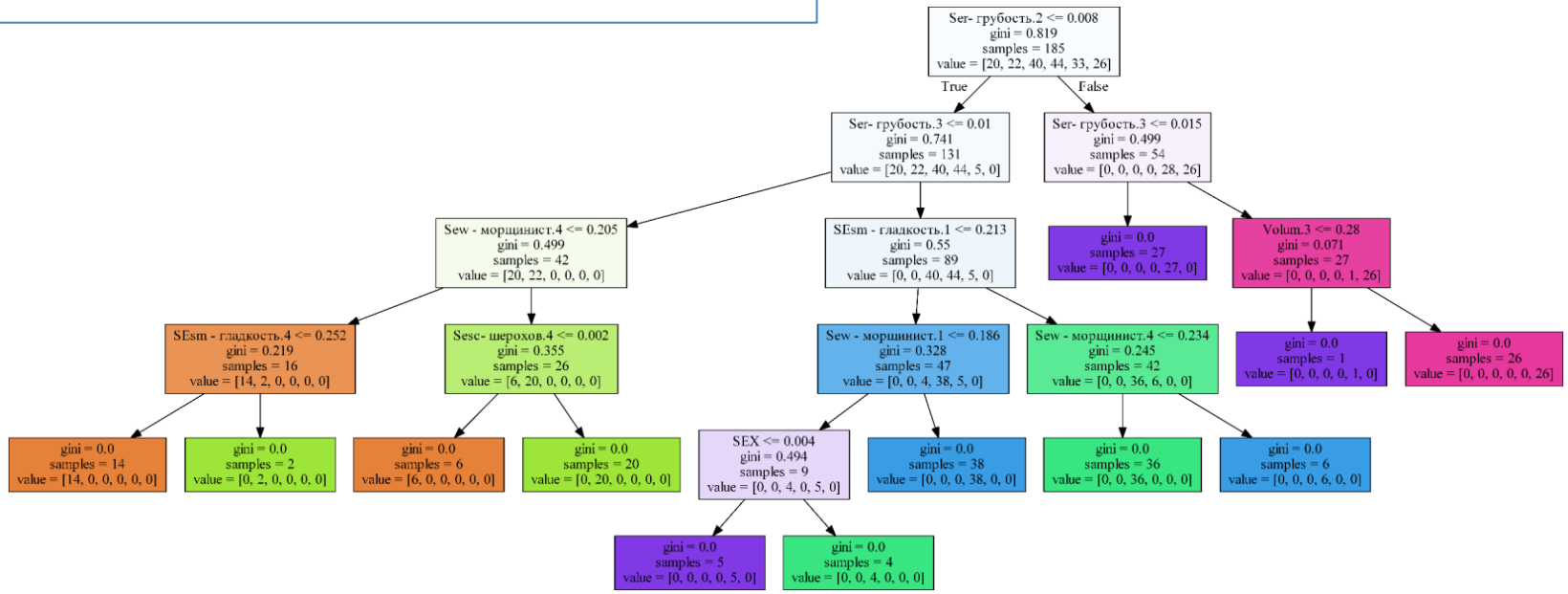



1. **Построена модель для определения возрастной группы, использующая данным по коже**

   Использован алгоритм Random Forest

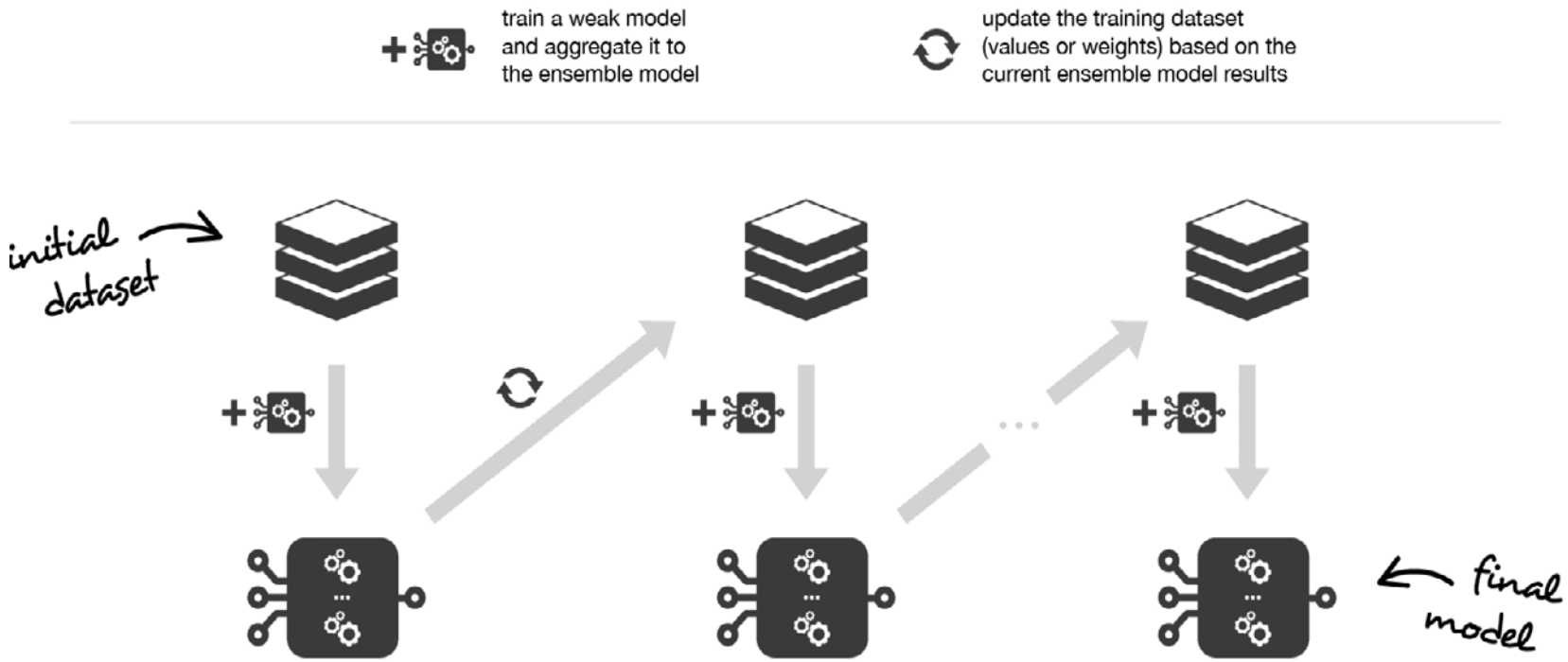
2. **Построена модель для определения пола, использующая данные по минеральной плотности костей**

   Проведен анализ алгоритмов. Использована логистическая регрессия.



# Бустинг (boosting)

**Бустинг (boosting)** — это ансамблевый метод, в котором строится множество базовых алгоритмов из одного семейства слабых предсказывающих моделей, объединяющихся затем в более сильную модель. Отличие состоит в том, что в бэггинге и случайном лесе базовые алгоритмы учатся независимо и параллельно, а в бустинге — последовательно. Каждый следующий базовый алгоритм в бустинге обучается так, чтобы уменьшить общую ошибку всех своих предшественников, т.е. ошибку текущего ансамбля.



# AdaBoost (Adaptive Boosting)

Алгоритм, предложенный Йоавом Фройндом и Робертом Шапире, может использоваться в сочетании с несколькими алгоритмами классификации для улучшения их эффективности за счет объединения их в ансамбль. Является адаптивным в том смысле, что каждый следующий ансамбль классификаторов строится по объектам, неверно классифицированным предыдущими комитетами.

AdaBoost — алгоритм построения «сильного» классификатора

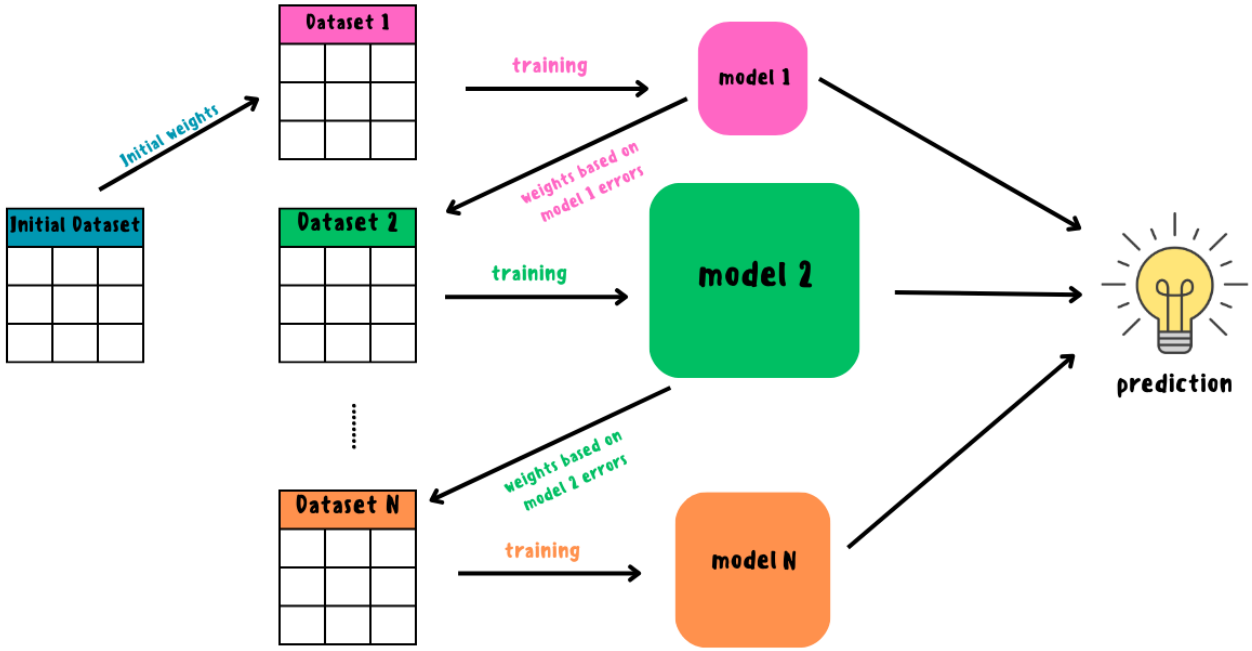
$$H(x) = sign\big(f(x)\big),$$

где

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

является линейной комбинацией «слабых» классификаторов

$$h_t(x): \chi \to \{-1, +1\}.$$




**Основная идея**

AdaBoost вызывает слабые классификаторы в цикле $t = 1, \dots, T$. После каждого вызова обновляется распределение весов $D_t$, которые отвечают за важность объектов обучающего множества для классификации. На каждой итерации веса каждого неверно классифицированного объекта возрастают, таким образом, новый комитет классификаторов «фокусирует своё внимание» на этих объектах.

# Алгоритм AdaBoost

Дано: $(x_1, y_1), \dots, (x_m, y_m)$ где $x_i \in X$, $y_i \in Y = \{-1, +1\}$.

1.Инициализировать веса $D_1(i) = \frac{1}{m}$, $i = 1, \dots, m$.

2.Для каждого $t = 1, \dots, T$:

- Используя выборку и распределение весов обучить слабый классификатор $h_t(x)$: $\chi \to \{-1, +1\}$ который минимизирует взвешенную ошибку классификации:

$$h_t = arg \min_{h_j \in H} \epsilon_j, \text{ где } \epsilon_j = \sum_{i=1}^{m} D_t(i) \left[ y_i \neq h_j(x_i) \right]$$

- Выбрать $\alpha_t \in \mathbb{R}$, $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$, где $\epsilon_t$ взвешенная ошибка классификатора $h_t$.

- Обновить веса примеров: $D_{t+1}(i) = \frac{D_t(i) \cdot exp(-\alpha_t y_i h_t(x_i))}{Z_t}$,

  где $Z_t$ является нормализующим параметром, чтобы $\sum_{i=1}^{m} D_{t+1}(i) = 1$.

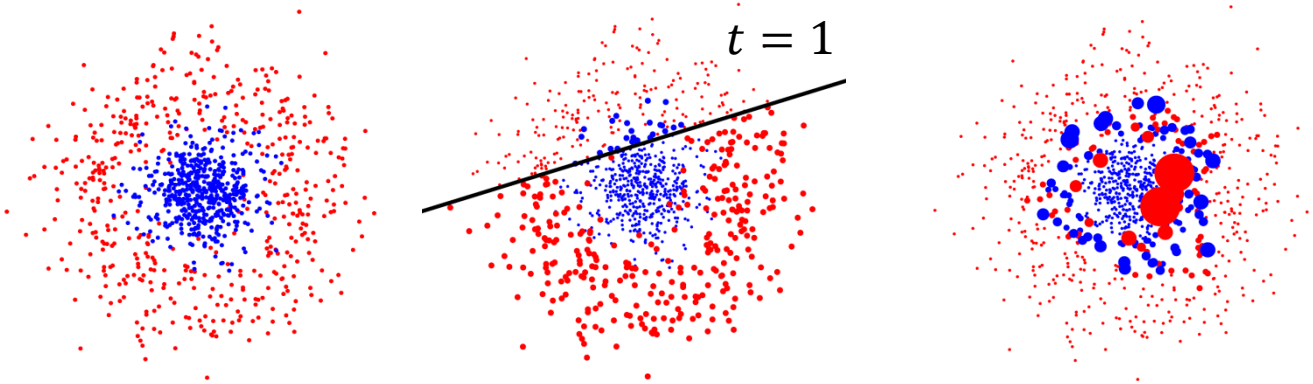
  Выражение для обновления $D_t$ должно быть выбрано таким образом, чтобы выполнялось условие:

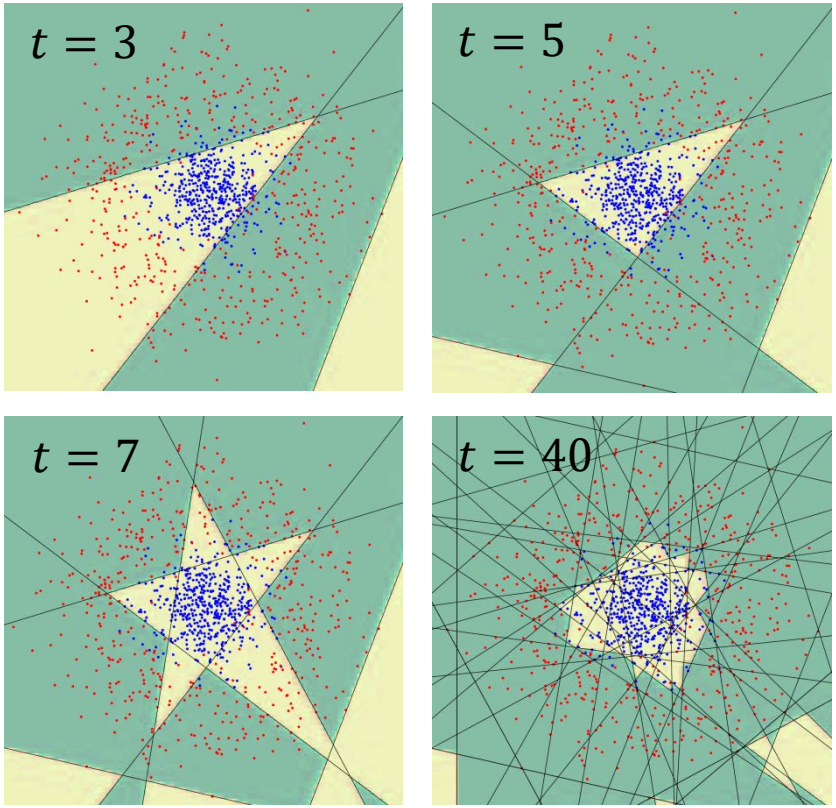
$$exp(-\alpha_t y_i h_t(x_i)) \begin{cases} < 1, y(i) = h_t(x) \\ > 1, y(i) \neq h_t(x) \end{cases}$$

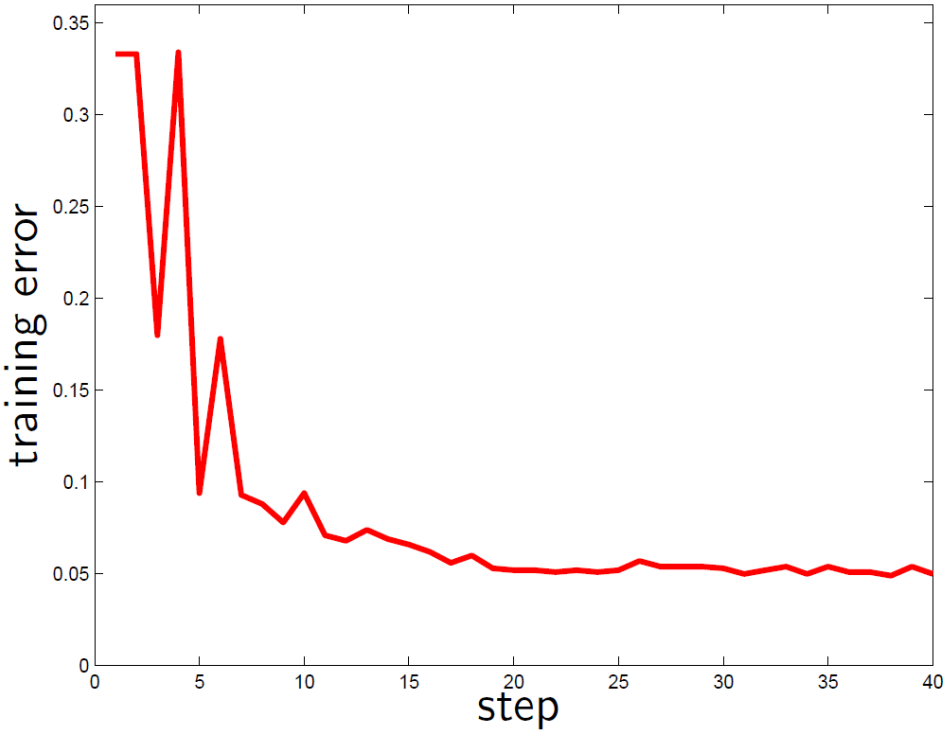
3.Рассчитать значение результирующего классификатора: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$.

Таким образом, после выбора оптимального классификатора $h_t$ для распределения $D_t$, объекты $x_i$, которые классификатор $h_t$ идентифицирует корректно, имеют веса меньшие, чем те, которые идентифицируются некорректно. Следовательно, когда алгоритм тестирует классификаторы на распределении $D_{t+1}$, он будет выбирать классификатор, который лучше идентифицирует объекты неверно распознаваемые предыдущим классификатором.

# Визуализация AdaBoost

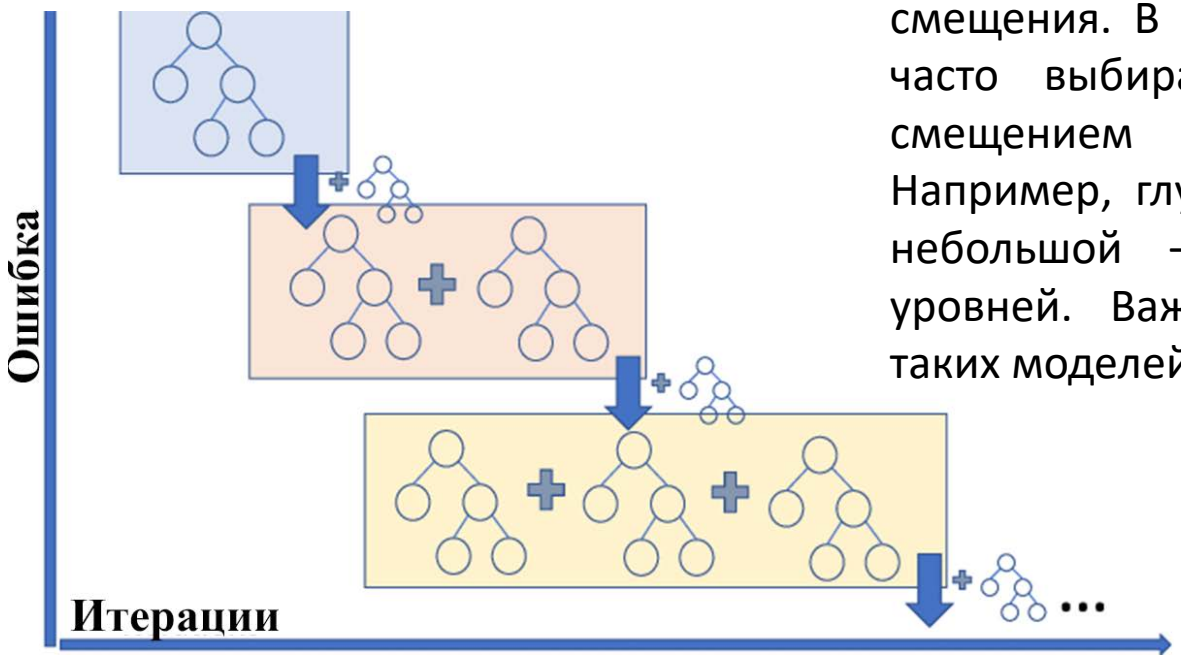


# Градиентный бустинг над решающими деревьями

Бустинг, использующий деревья решений в качестве базовых алгоритмов - **Gradient Boosting on Decision Trees** (**GBDT**). Отлично работает на выборках с «табличными», неоднородными данными. Такой бустинг способен эффективно находить нелинейные зависимости в данных различной природы. Широко применяется во многих конкурсах по машинному обучению и задачах из индустрии (поисковом ранжировании, рекомендательных системах, таргетировании рекламы, предсказании погоды, пункта назначения такси и многих других).




Основная цель бустинга — уменьшение смещения. В качестве базовых алгоритмов часто выбирают алгоритмы с высоким смещением и небольшим разбросом. Например, глубина деревьев должна быть небольшой — обычно не больше 2-3 уровней. Важной причиной для выбора таких моделей – скорость обучения.

# Бустинг - Идея

Рассмотрим задачу регрессии с квадратичной функцией потерь:

$$\mathcal{L}(y, x) = \frac{1}{2}\sum_{i=1}^{N}(y_i - a(x_i))^2 \to min$$

Для решения будем строить композицию из $K$ базовых алгоритмов

$$a(x) = a_K(x) = b_1(x) + \cdots + b_K(x)$$

Если мы обучим единственное решающее дерево, то качество такой модели, скорее всего, будет низким. Однако о построенном дереве мы знаем, на каких объектах оно давало точные предсказания, а на каких ошибалось.

Попробуем использовать эту информацию и обучим еще одну модель. Допустим, что предсказание первой модели на объекте $x_l$ на 10 больше, чем необходимо (т.е. $b_1(x_l) = y_l + 10$). Если бы мы могли обучить новую модель, которая на $x_l$ будет выдавать ответ -10, то сумма ответов этих двух моделей на объекте $x_l$ в точности совпала бы с истинным значением:

$$b_1(x_l) + b_2(x_l) = (y_l + 10) + (-10)$$

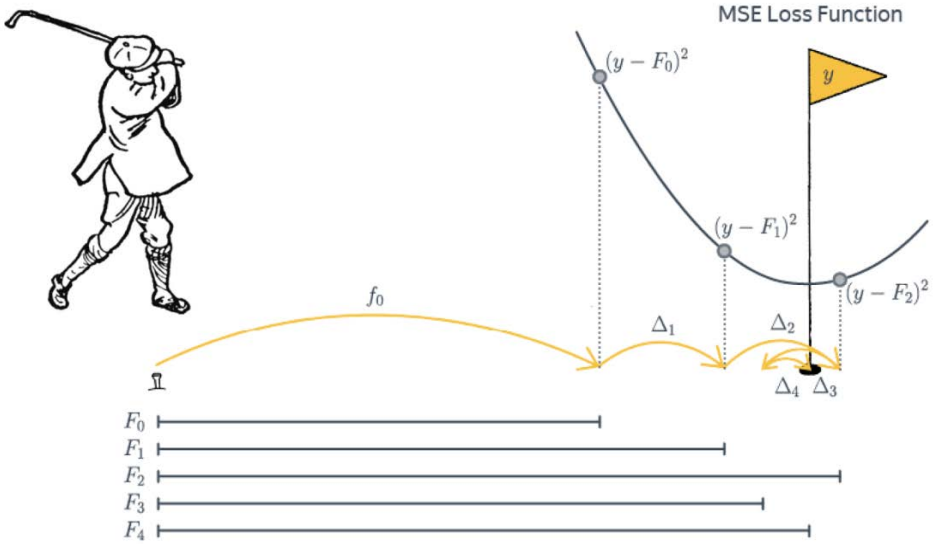
Другими словами, если вторая модель научится предсказывать разницу между реальным значением и ответом первой, то это позволит уменьшить ошибку композиции.

В реальности вторая модель тоже не сможет обучиться идеально, поэтому обучим третью, которая будет «компенсировать» неточности первых двух. Будем продолжать так, пока не построим композицию из $K$ алгоритмов.

# Бустинг – Аналогии

- **Гольфист.**

Цель– загнать мяч в лунку с координатой $y_{ball}$. Положение мяча здесь – ответ композиции $a(x_{ball})$. Каждым ударом гольфиста переводит мяч из текущего положения $a_k(x_{ball})$ в положение $a_{k+1}(x_{ball})$ и это та поправка, которую вносит очередной базовый алгоритм в композицию.




Если гольфист все делает правильно, то функция потерь будет уменьшаться:

$$\mathcal{L}\big(y, a_{k+1}(x)\big) < \mathcal{L}\big(y, a_k(x)\big)$$

то есть мяч постепенно будет приближаться к лунке.

- **Ряд Тейлора**

Бесконечно дифференцируемую функцию $f(x)$ на интервале $x \in (a - R, a + R)$ можно представить в виде бесконечной суммы степенных функций:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(X - A)^n$$

Одна, самая первая степенная функция в разложении, очень грубо приближает $f(x)$. Прибавляя к ней следующую, мы получим более точное приближение. Каждая следующая элементарная функция увеличивает точность приближения, но менее заметна в общей сумме.

# Задача регрессии: Формальное описание – Шаг 1

Рассмотрим тот же пример с задачей регрессии и квадратичной функцией потерь:

$$\mathcal{L}(y, x) = \frac{1}{2}\sum_{i=1}^{N}(y_i - a(x_i))^2 \to min$$

Для решения также будем строить композицию из *K* базовых алгоритмов семейства *B*:

$$a(x) = a_K(x) = b_1(x) + \cdots + b_K(x)$$

В качестве базовых алгоритмов выберем семейство *B* решающих деревьев некоторой фиксированной глубины.

Используя известные нам методы построения решающих деревьев, обучим алгоритм $b_1(x) \in B$ , который наилучшим образом приближает целевую переменную:

$$b_1(x) = \underset{b \in B}{\operatorname{argmin}}\, \mathcal{L}(y, b(x))$$

Построенный алгоритм $b_1(x)$, скорее всего, работает не идеально. Более того, если базовый алгоритм работает слишком хорошо на обучающей выборке, то высока вероятность переобучения (низкий уровень смещения, но высокий уровень разброса). Далее вычислим, насколько сильно отличаются предсказания этого дерева от истинных значений:

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

# Задача регрессии: Формальное описание – Шаг 2

Теперь мы хотим скорректировать $b_1(x)$ с помощью $b_2(x)$; в идеале так, чтобы $b_2(x)$ идеально предсказывал величины $s_i^1$, ведь в этом случае

$$a_2(x_i) = b_1(x_i) + b_2(x_i) = b_1(x_i) + s_i^1 = b_1(x_i) + (y_i - b_1(x_i)) = y_i$$

Найти совершенный алгоритм, скорее всего, не получится, но по крайней мере мы можем выбрать из семейства наилучшего представителя для такой задачи. Итак, второе решающее дерево будет обучаться предсказывать разности $s_i^1$:

$$b_2(x) = \underset{b \in B}{\operatorname{argmin}} \mathcal{L}(s^1, b(x))$$

Ожидается, что композиция из двух таких моделей $a_2(x_i) = b_1(x_i) + b_2(x_i)$ станет более качественно предсказывать целевую переменную *y*.

Далее рассуждения повторяются до построения всей композиции. На *k*-ом шаге вычисляется разность между правильным ответом и текущим предсказанием композиции из *k-1* алгоритмов:

$$s_i^{k-1} = y_i - \sum_{i=1}^{k-1} b_{k-1}(x_i) = y_i - a_{k-1}(x_i)$$

Затем k-ый алгоритм учится предсказывать эту разность:

$$b_k(x) = \underset{b \in B}{\operatorname{argmin}} \mathcal{L}\left(s^{k-1}, b(x)\right)$$

а композиция в целом обновляется по формуле:

$$a_k(x) = a_{k-1}(x) + b_k(x)$$

Обучение *K* базовых алгоритмов завершает построение композиции.

# Обобщение на другие функции потерь

Отметим важное свойство функции потерь в рассмотренном выше примере с регрессией. Для этого посчитаем производную функции потерь по предсказанию $z = a_k(x_i)$ модели для *i*-го объекта:

$$\frac{\partial \mathcal{L}(y_i, z)}{\partial z}\Big|_{z=a_k(x_i)} = \frac{\partial}{\partial z}\frac{1}{2}(y_i - z)^2\Big|_{z=a_k(x_i)} = a_k(x_i) - y_i$$

Видим, что разность, на которую обучается *k*-й алгоритм, выражается через производную:

$$s_i^k = y_i - a_k(x_i) = -\frac{\partial \mathcal{L}(y_i, z)}{\partial z}\Big|_{z=a_k(x_i)}$$

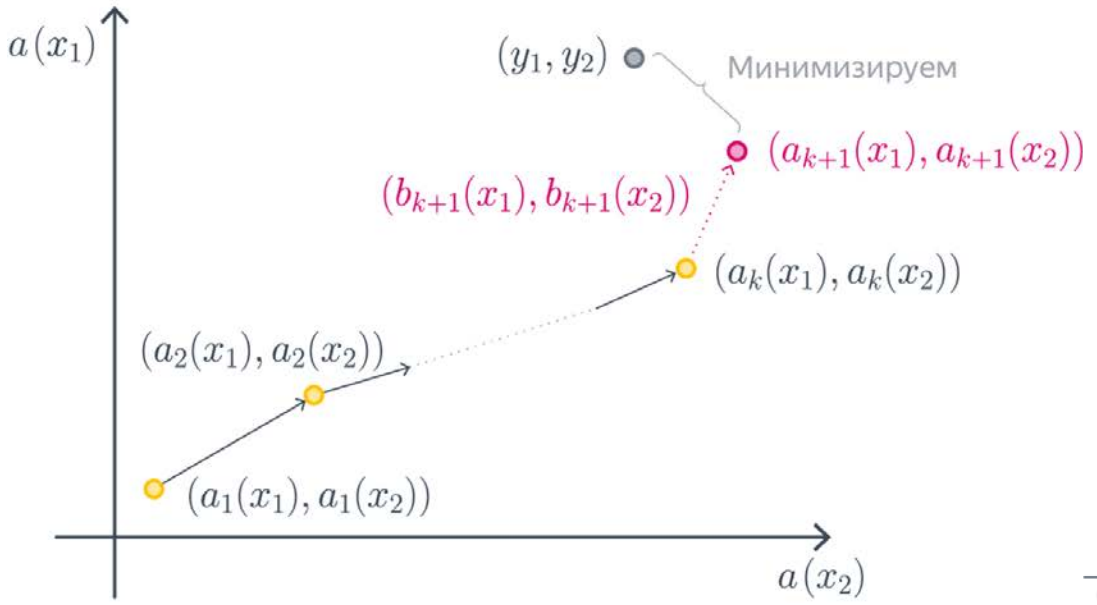
Таким образом, для каждого объекта $x_i$ очередной алгоритм в бустинге обучается предсказывать антиградиент функции потерь по предсказанию модели $-\frac{\partial \mathcal{L}(y_i,z)}{\partial z}$ в точке $a_k(x_i)$ предсказания текущей части композиции на объекте $x_i$.

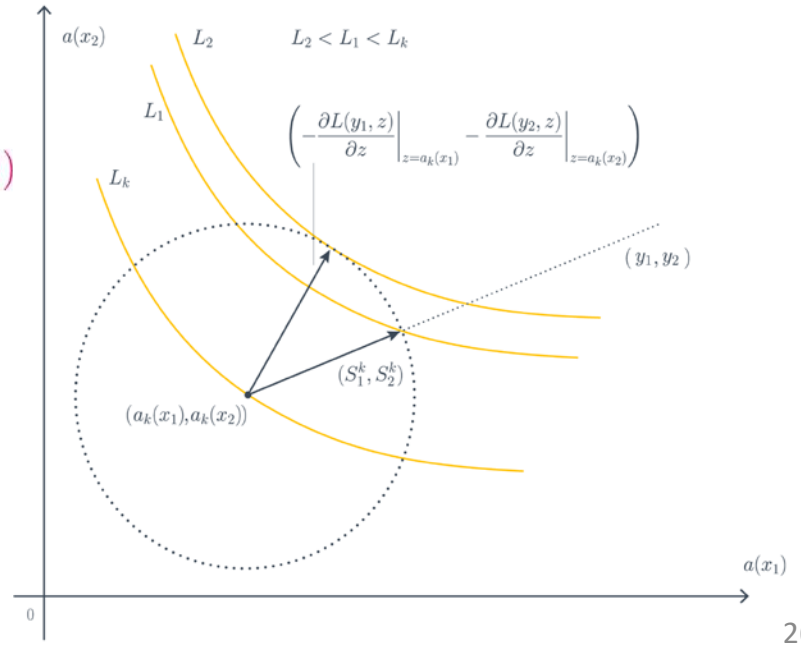
# Почему же это важно?

Это наблюдение позволяет обобщить подход построения бустинга на произвольную дифференцируемую функцию потерь. Для этого мы заменяем обучение на разность $s_i^k$ обучением на антиградиент функции потерь $(-g_i^k)$ , где

$$g_i^k = \frac{\partial \mathcal{L}(y_i, z)}{\partial z}\bigg|_{z=a_k(x_i)}$$

Обучение композиции можно представить (вспомните аналогию с гольфистом) как перемещение предсказания из точки $(a_k(x_1), a_k(x_2), \dots, a_k(x_N))$ в точку $(a_{k+1}(x_1), a_{k+1}(x_2), \dots, a_{k+1}(x_N))$. В конечном итоге мы ожидаем, что точка $(a_K(x_1), a_K(x_2), \dots, a_K(x_N))$ будет располагаться как можно ближе к точке с истинными значениями $(y_1, y_2, \dots, y_N)$.



# Обучение базового алгоритма

При построении очередного базового алгоритма $b_{k+1}$ мы решаем задачу регрессии с таргетом, равным антиградиенту функции потерь исходной задачи на предсказании $a_k = b_1 + \cdots + b_k$.

Теоретически можно воспользоваться любым методом построения регрессионного дерева. Важно выбрать оценочную функцию S, которая будет показывать, насколько текущая структура дерева хорошо приближает антиградиент. Её нужно будет использовать для построения критерия ветвления:

$$|R| \cdot S(R) - \left|R_{right}\right| \cdot S\left(R_{right}\right) - \left|R_{left}\right| \cdot S\left(R_{left}\right) \to max$$

где $S(R)$– значение функции S в вершине R,

$S\left(R_{right}\right), S\left(R_{left}\right)$– значения в левой и правой дочерних вершинах R после добавления предиката,

$|\cdot|$– количество элементов, пришедших в вершину.

В итоге обучение базового алгоритма проходит в два шага:

1. по **функции потерь** вычисляется целевая переменная для обучения следующего базового алгоритма:

$$g_i^k = \left.\frac{\partial \mathcal{L}(y_i, z)}{\partial z}\right|_{z=a_k(x_i)}$$

2. строится регрессионное дерево на обучающей выборке $(x_i, -g_i^k)$, минимизирующее выбранную **оценочную функцию**.

# Темп обучения (learning rate)

Обучение ансамбля с помощью градиентного бустинга может привести к переобучению, если базовые алгоритмы слишком сложные. Например, если сделать решающие деревья слишком глубокими (более 10 уровней), то при обучении бустинга ошибка на обучающей выборке даже при малом числе базовых моделей *K* может приблизиться к нулю, но на тестовой выборке ошибка будет большой.

Существует два решения этой проблемы:

- Во-первых, необходимо упростить базовую модель, уменьшив глубину дерева (либо примерив какие-либо ещё техники регуляризации).
- Во-вторых, мы можем ввести параметр, называемый **темпом обучения** (**learning rate**) $\eta \in (0,1]$:

$$a_{k+1}(x) = a_k(x) + \eta b_{k+1}(x)$$

Присутствие этого параметра означает, что каждый базовый алгоритм вносит относительно небольшой вклад во всю композицию: если расписать сумму целиком, она будет иметь вид

$$a_{k+1}(x) = b_1(x) + \eta b_2(x) + \cdots + \eta b_{k+1}(x)$$

Значение параметра обычно определяется эмпирически по входным данным. Темп обучения связан с количеством итераций градиентного бустинга. Чем меньше learning rate, тем больше итераций потребуется сделать для достижения того же качества на обучающей выборке.

# Значимость признаков (Feature importance)

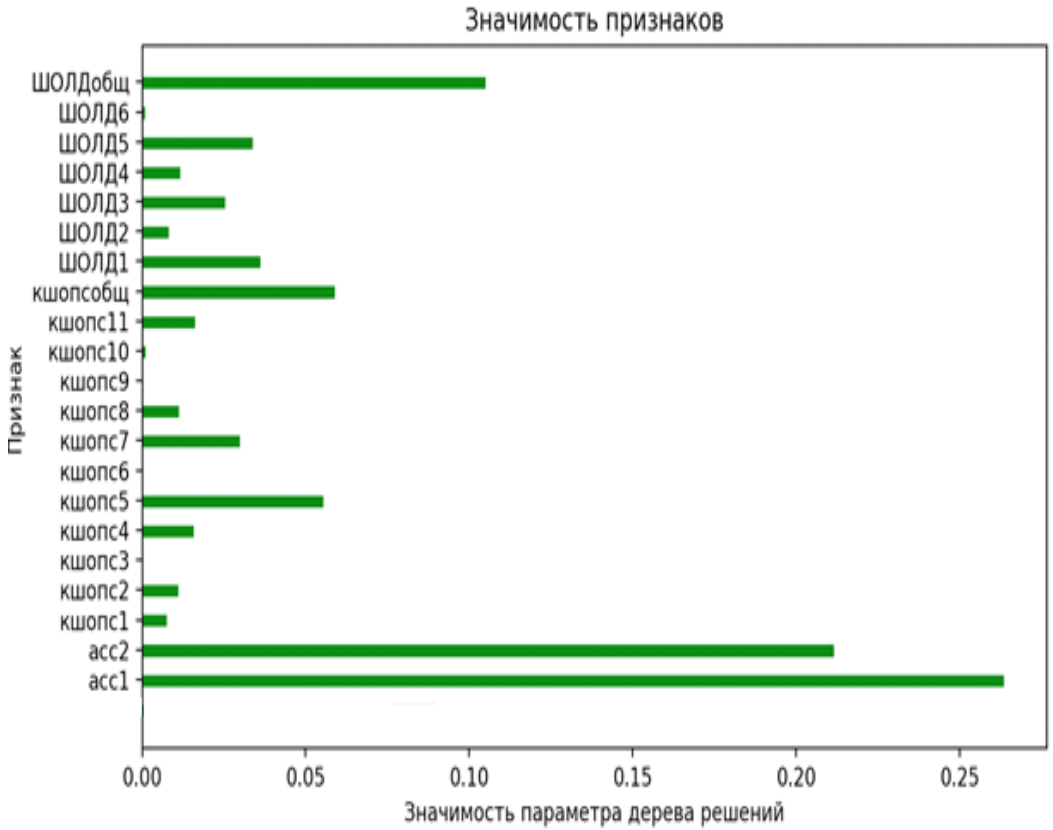
Признаки, используемые в верхней части дерева, влияют на окончательное предсказание для большей доли обучающих объектов, чем признаки, попавшие на более глубокие уровни.

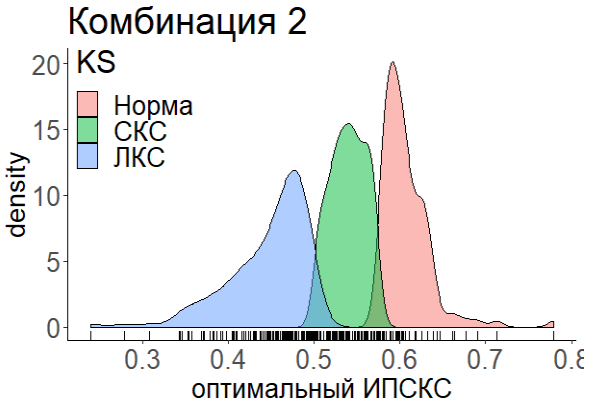
Таким образом, ожидаемая доля обучающих объектов, для которых происходило ветвление по данному признаку, может быть использована в качестве оценки его относительной важности для итогового предсказания. Усредняя полученные оценки важности признаков по всем решающим деревьям из ансамбля, можно уменьшить дисперсию такой оценки и использовать ее для отбора признаков.

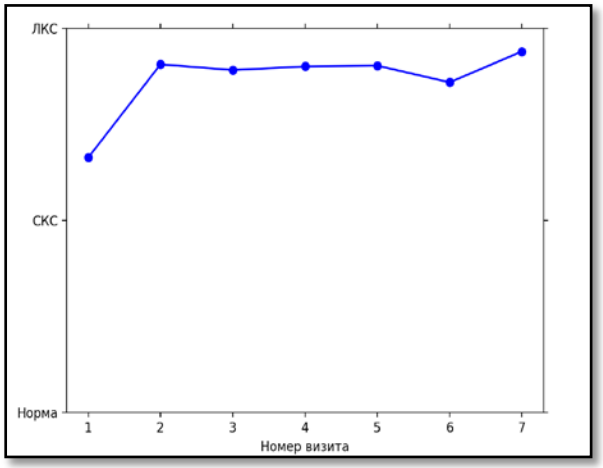
Этот метод известен как **MDI** (**mean decrease in impurity**).

# Исследование роли возрастных, сердечно-сосудистых и нейродегенеративных факторов в развитии субъективного и легкого когнитивного снижения в среднем и пожилом возрасте

Разработаны алгоритмы расчета интегрального показателя степени когнитивного снижения (ИПСКС) и классификаторов на их основе, (оптимизированный статистический классификатор и классификатор на основе методов машинного обучения).



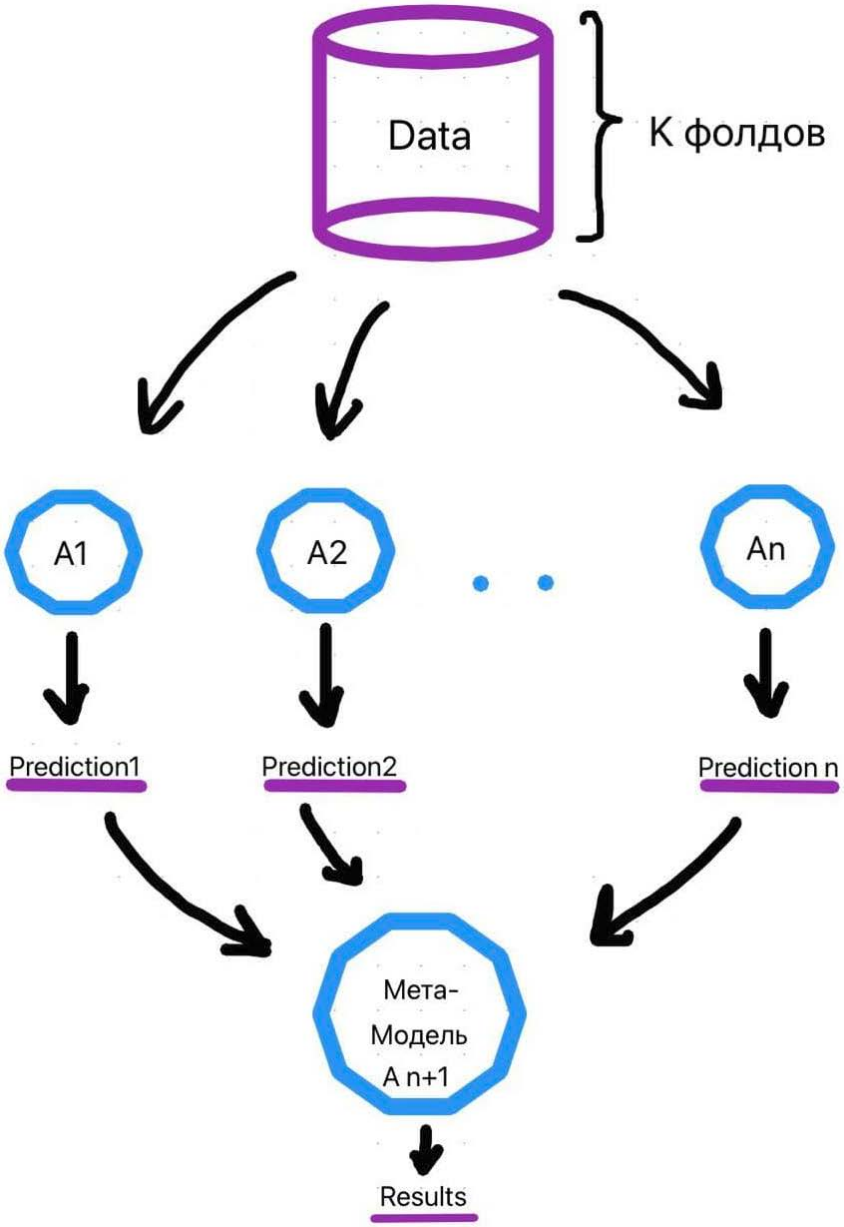



# Стекинг (stacking)

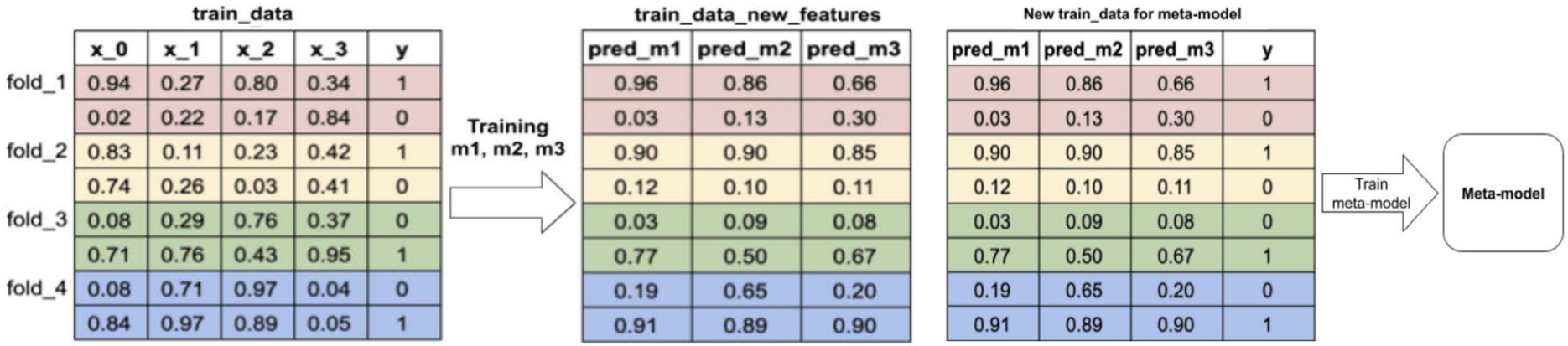
Алгоритм ансамблирования, основные отличия которого от предыдущих состоят в следующем:

- может использовать алгоритмы разного типа, а не только из какого-то фиксированного семейства;
- результаты базовых алгоритмов объединяются в один с помощью обучаемой мета-модели, а не с помощью какого-либо обычного способа агрегации (суммирования или усреднения).



# Стекинг (stacking) - Обучение

- Общая выборка разделяется на тренировочную и тестовую.
- Тренировочная выборка делится на n фолдов. Затем эти фолды перебираются тем же способом, что используется при кросс-валидации: на каждом шаге фиксируются (n-1) фолдов для обучения базовых алгоритмов и один — для их предсказаний (вычисления мета-факторов). Такой подход нужен для того, чтобы можно было использовать всё тренировочное множество, и при этом базовые алгоритмы не переобучались.
- На полученных мета-факторах обучается мета-модель. Кроме мета-факторов, она может принимать на вход и параметры из исходного датасета. Выбор зависит от решаемой задачи.



# Спасибо за внимание