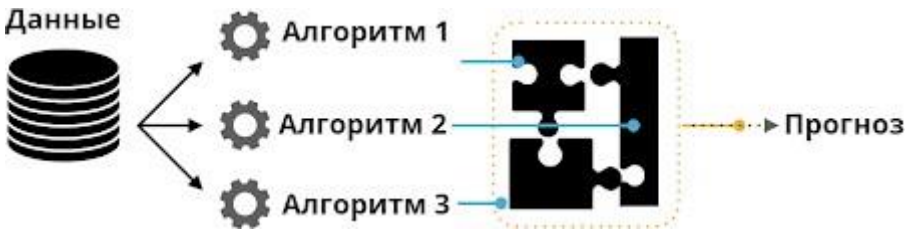
Ансамбль моделей — это метод, в котором несколько алгоритмов (или вариации одного и того же) обучаются на одних данных, а итоговый прогноз строится на основе всех полученных от моделей прогнозов.



Ансамбли моделей. Бутстреппинг. Бэггинг

В основе бэггинга лежит статистический метод, который называется **бутстрепом (bootstrap)**.

Идея бутстрепа заключается в генерации выборок размера n (так называемых бутстреп-выборок) из исходного датасета размера N путём случайного выбора элементов с повторениями в каждом из наблюдений.

Пусть у нас есть выборка из 12 клиентов компании: у каждого из них есть свой ID (от 1 до 12) и какие-то характеристики). Мы можем создавать из данной выборки множество различных новых выборок клиентов с новым количеством человек (в данном случае представлены выборки из пяти человек). При этом информацию про одного и того же клиента можно использовать повторно.

Это намного проще, чем находить новые выборки. По сути, мы собираем данные лишь единожды, а затем на их основе генерируем много выборок для обучения моделей. Это экономит огромные объёмы ресурсов и времени.

Bias и Variance

Bias-variance decomposition, или, как его называют по-русски, «разложение ошибки на смещение и разброс», очень полезно для анализа ансамблей моделей.

Смещение — это разница между математическим ожиданием для прогноза и реальным значением:

$$Bias[\hat{f}(x)] = E[\hat{f}(x)] - y$$

Здесь:

- $E[\widehat{f}(x)]$ — математическое ожидание для прогноза,
- y — реальное значение функции.

Смысл смещения — способность получить лучшую среди всех возможных моделей, то есть максимально точные прогнозы.

Разброс — это величина разницы в результатах обучения модели на разных выборках:

$$\mathrm{Var}[\hat{f}(x)] = \mathrm{E}\left[\left(\mathrm{E}[\hat{f}(x)] - \hat{f}(x)\right)^2\right]$$

Примечание. С математической точки зрения разброс модели определяется как математическое ожидание квадрата разницы ожидаемого прогноза и реализованного прогноза модели.

Разброс характеризует устойчивость модели к изменениям в обучающей выборке:

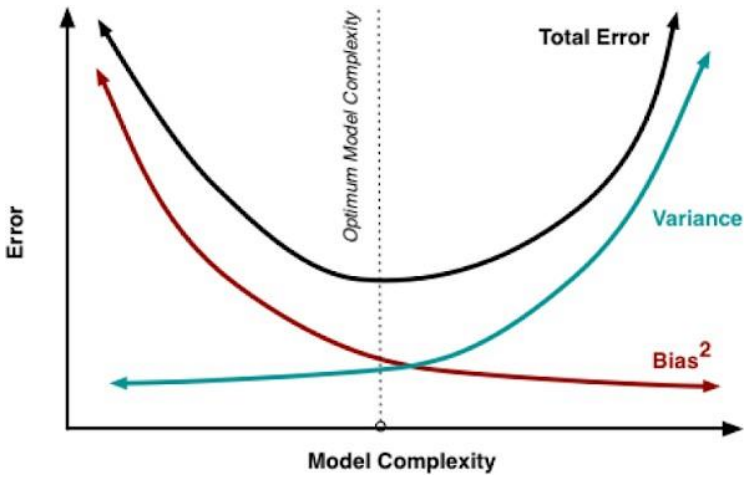
- Если результат сильно зависит от того, какие объекты присутствуют в выборке, разброс будет большим.

- Если алгоритм работает стабильно вне зависимости от особенностей выборки, разброс будет маленьким.

Когда говорят про разложение на bias и variance, то часто упоминают некую точку баланса.

- Если модель очень простая, с маленьким количеством параметров, то, скорее всего, у неё будет очень большое смещение, но маленький разброс.
- Если модель очень сложная, со множеством параметров, у неё будет большой разброс и маленькое смещение.

Схематично эти зависимости можно изобразить следующим образом (это схема не для конкретной модели, а лишь иллюстрация тенденций).



Бэггинг

При построении моделей всегда есть вероятность, что при обучении на других данных получились бы другие результаты. Для того чтобы нивелировать такую вероятность, можно использовать **бэггинг**.

Его идея состоит в том, что мы берём несколько независимых моделей (т. е. моделей, которые обучены на независимых выборках) и усредняем полученные по ним результаты. Таким образом мы получаем модель, имеющую меньший разброс, так как при её построении мы учли несколько моделей.

Важно отметить, что при бэггинге **размер каждой бутстреп-выборки должен совпадать с размером исходной выборки.**

Схематично процесс бэггинга можно представить следующим образом:

Теперь давайте сформулируем и объясним эту идею математически.

Пусть у нас есть некоторая выборка, и мы с помощью бутстрепа генерируем из неё ещё B выборок:

$$X_1, \ldots, X_B$$

После этого мы определяем много базовых алгоритмов (всего B моделей — по числу выборок) и обучаем каждый базовый алгоритм $a_i(x)$ на своей выборке.

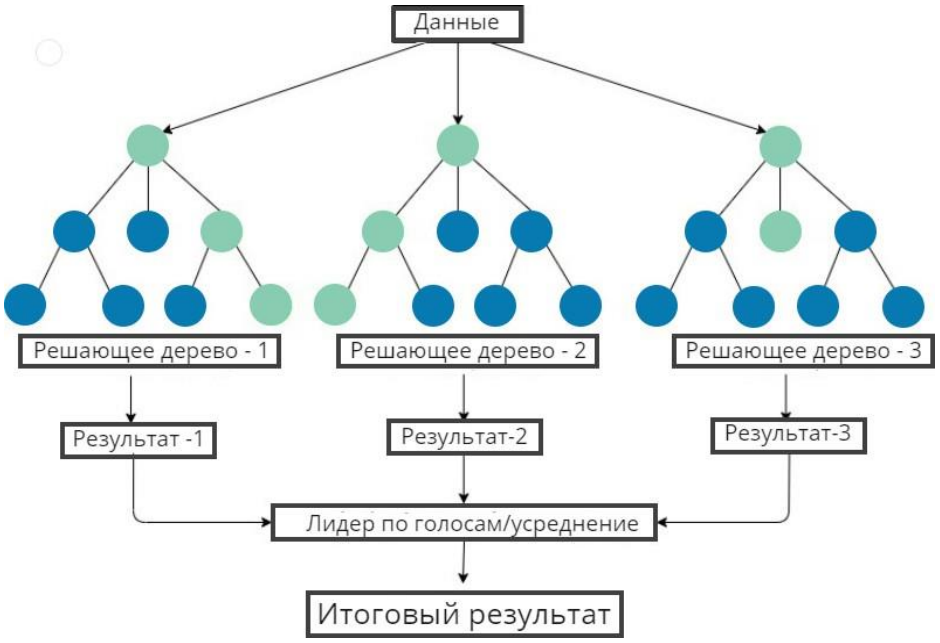
После этого получаем итоговый результат:

$$a(x) = \frac{1}{B} \sum_{i=1}^{B} a_i(x)$$

- ➔ Если мы рассматриваем задачу классификации, то, по сути, модели «голосуют» за свой класс.
- ➔ Если мы рассматриваем задачу регрессии, то результат — просто среднее арифметическое прогнозов по всем моделям.

Случайный лес

Алгоритм обучения случайного леса можно изобразить в виде следующей схемы:



Есть какое-то количество решающих деревьев, каждое из которых мы обучаем на некоторой подвыборке из данных. Получив вердикты от всех моделей, определяем итоговый результат для каждого объекта.

Для **регрессии** правило формирования итогового результата формулируется следующим образом:

$$f(x) = \frac{1}{K}\sum_{i=1}^{K} a_i(x)$$

Здесь:

- K — количество моделей,
- $a_i(x)$ — алгоритм решающего дерева,
- $f(x)$ — значение финального предсказания ансамбля.

Это означает, что мы просто находим среднее арифметическое для всех полученных предсказаний.

Правило формирования итогового результата для **классификации**:

$$f(x) = \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{K} [a_i(x) = y]$$

Здесь деревья просто голосуют за некоторый класс, и объекту присваивается метка класса, за который было отдано наибольшее количество голосов.

Давайте рассмотрим алгоритм реализации случайного леса.

Одно из важных понятий, которое здесь появляется, — это **метод случайных подпространств**, который используется для построения ансамблей моделей.

Кратко опишем его принцип:

1. Отбираем обучающую выборку.
2. Определяем число моделей, которые войдут в ансамбль.
3. Для каждой модели берём не все признаки, а только часть из них и формируем выборку с использованием случайно выбранного набора признаков.
4. Объединяем все результаты и определяем итоговое решение по объектам.

Обратите внимание на важную особенность: здесь выбирается не только обучающая выборка, но ещё и случайная выборка из признаков.

Алгоритм случайного леса в таком контексте реализуется следующим образом:

1. Для того чтобы построить i-е дерево леса, из обучающей выборки X берём случайную подвыборку X_i того же размера, что и вся обучающая выборка.
2. После этого в каждой вершине каждого дерева из M возможных признаков выбираем случайную группу признаков объёма L. Для выбранных признаков ищем оптимальное разбиение. Рекомендуется использовать $L = \sqrt{M}$ в задачах классификации и L = M/3 — в задачах регрессии.
3. Для получения предсказания необходимо воспользоваться обычным принципом бэггинга: взять усреднённый ответ в случае регрессии или самый популярный класс — для классификации.

Out-of-bag error

Ошибка out-of-bag — это способ оценить качество случайного леса.

Для того чтобы найти out-of-Bag-оценку:

1. Для каждого объекта x_i получаем предсказания всех деревьев a_b, обучавшихся на бутстреп-выборках X_b не содержащих x_i
2. Усредняем эти предсказания.
3. Находим значение ошибки для усреднённого предсказания.
4. Усредняем значение функционала ошибки для всех объектов выборки.

Строго математически это можно записать следующим образом:

$$\text{OOB} = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, \frac{1}{\sum_{b=1}^{B}[x_i \notin X_b]}\sum_{b=1}^{B}[x_i \notin X_b]\, a_b(x_i)\right)$$

Бустинг

В целом, идеи бустинга и бэггинга очень похожи: в обоих случаях мы берём слабые модели и объединяем их для получения более качественного прогноза. Однако есть одно ключевое различие:

- В **бэггинге** все модели обучаются одновременно, независимо и параллельно. В качестве итогового предсказания берётся усреднённый ответ (в задаче регрессии) или делается прогноз по большинству голосов (в задаче классификации).
- В **бустинге** все модели обучаются поочерёдно, причём каждая последующая старается исправить ошибки, совершённые предыдущими.

AdaBoost

1. Инициализируем веса объектов:

$$w_j = \frac{1}{N}, j = 1, 2, \ldots, N$$

2. Для всех i от 1 до K (если у нас K базовых моделей):

2.1. Строим классификатор $a_i(x)$, используя веса w_j.

2.2. Вычисляем ошибку:

$$\text{err}_i = \sum_{j=1}^{N} w_j \left[y_j \neq a_i\left(x_j\right)\right]$$

2.3. Вычисляем вес нового алгоритма:

$$c_i = \frac{1}{2}\ln\frac{1 - err_i}{err_i}$$

2.4. Получаем новые веса объектов:

$$w_j <- w_j \cdot \exp(c_i \left[y_j \neq a_i\left(x_j\right)\right]), j = 1, \ldots, N$$

2.5. Нормируем веса объектов:

$$w_j \longleftarrow \frac{w_j}{\sum_{j=1}^{N} w_j}$$

3. Группируем полученные модели:

$$f_K(x) = \text{sign}\left[\sum_{i=1}^{K} c_i a_i(x)\right]$$

Градиентный бустинг

Принцип его работы аналогичен AdaBoost: следующие модели улучшают композицию построенных ранее.

Пусть у нас есть некоторая функция потерь $L(y, \hat{y})$. Она зависит от двух аргументов: y — истинный ответ, $\hat{y} = a(x)$ — прогноз модели.
Для задачи регрессии это может быть, например, квадратичная ошибка:

$$L(y, \hat{y}) = (y - \hat{y})^2,$$

а для задачи классификации — логистическая функция потерь:

$$L(y, \hat{y}) = \log(1 + \exp(-y\hat{y}))$$

Пусть к некоторому моменту обучены K-1 алгоритмов $a_1(x), \ldots, a_{K-1}(x)$, то есть композиция имеет вид:

$$f_{K-1}(x) = \sum_{i=1}^{K-1} a_i(x)$$

Теперь добавляем в композицию ещё один алгоритм — $a_K(x)$. Этот алгоритм обучается так, чтобы как можно сильнее уменьшить ошибку композиции на обучающей выборке:

$$\sum_{j=1}^{N} L\left(y_j, f_{K-1}\left(x_j\right) + a_k\left(x_j\right)\right) \to \min_{a_k}$$

Для того чтобы найти a_K, минимизирующее функционал, задачу разбивают на две подзадачи:

1. На первом этапе определяем, какие значения $s_1, \ldots, s_N$ должен принимать алгоритм $a_K(x_j) = s_j$, чтобы на объектах обучающей выборки ошибка была минимальной. Формально это можно представить так:

$$F(s) = \sum_{j=1}^{N} L\left(y_j, a_{K-1}\left(x_j\right) + s_j\right) \rightarrow \min_s,$$

где $s = (s_1, \ldots, s_N)$.

Иными словами, необходимо найти такой вектор сдвигов s, который будет минимизировать функцию F(s).

Вы, вероятно, помните, что направление наискорейшего убывания функции — это **антиградиент**, так что берём его в качестве вектора s:

$$s = -\nabla F = \begin{bmatrix} -L'_{\hat{y}}\left(y_1, a_{K-1}\left(x_1\right)\right) \\ -L'_{\hat{y}}\left(y_2, a_{K-1}\left(x_2\right)\right) \\ \vdots \\ -L'_{\hat{y}}\left(y_N, a_{K-1}\left(x_N\right)\right) \end{bmatrix}$$

Компоненты данного вектора — это те значения, которые должен принимать алгоритм $a_K(x)$ на объектах обучающей выборки, чтобы итоговая ошибка композиции была как можно меньше.

2.

Второй этап — построение такого алгоритма $a_K(x)$. По сути, задача построения алгоритма $a_K(x)$ — это обычная задача регрессии на размеченных данных. В данном случае у нас есть обучающая выборка $(x_j, s_j)_{j=1}^N$, нам просто нужно предсказать значения вектора s.

Например, используем квадратичную функцию потерь:

$$a_K(x) = \arg\min_a \frac{1}{N} \sum_{j=1}^{N} \left(a\left(x_j\right) - s_j\right)^2$$

Последовательность шагов реализации алгоритма, которую можно запрограммировать:

1. Инициализируем композицию **GBM (Gradient Boosting Machine)** — $f(x) = a_0(x)$, то есть добавляем первый базовый алгоритм. Например, можно использовать:
   - алгоритм $a_0(x) = 0$, который всегда возвращает 0 (в задаче регрессии);
   - более сложный алгоритм $a_0(x) = \frac{1}{N}\sum_{j=1}^{N} y_i$, который возвращает средний истинный ответ по всем элементам обучающей выборки (в задаче регрессии);
   - алгоритм $a_0(x) = arg\ max_{y \in Y} \sum_{j=1}^{N} [y_i = y]$, который всегда возвращает метку самого распространённого класса в обучающей выборке (в задаче классификации).
2. Итеративно повторяем следующие три шага:

   2.1. Вычисляем вектор сдвига:

$$s = -\nabla F = \begin{bmatrix} -L'_{\hat{y}}\left(y_1, a_{K-1}\left(x_1\right)\right) \\ -L'_{\hat{y}}\left(y_2, a_{K-1}\left(x_2\right)\right) \\ \vdots \\ -L'_{\hat{y}}\left(y_N, a_{K-1}\left(x_N\right)\right) \end{bmatrix}$$

   2.2. Строим очередной базовый алгоритм $a_K(x)$, который предсказывает вектор-сдвиг:

$$a_K(x) = \arg\min_a \frac{1}{N}\sum_{j=1}^{N}\left(a\left(x_j\right) - s_j\right)^2$$

   2.3. Добавляем $a_K(x)$ в композицию:

$$a_K(x) = \sum_{i=1}^{K} a_i(x)$$

3. Если выполнен критерий остановки, останавливаем итеративный процесс.

XGBoost и CatBoost

XGBoost — одна из самых эффективных реализаций алгоритма Gradient Boosted Trees. Название XGBoost расшифровывается как eXtreme Gradient Boosting. XGBoost — это улучшение GBM через системную оптимизацию и усовершенствование алгоритма.

У XGBoost есть **ряд существенных улучшений** по сравнению с классической реализацией градиентного бустинга:

1. **Регуляризация**

   В алгоритме XGBoost есть встроенные L1- и L2-регуляризации, которые предотвращают переобучение. Из-за этого XGBoost ещё иногда называют регуляризованным GBM.

2. **Параллелизация вычислений**

   XGBoost использует параллельные вычисления, поэтому он работает быстрее, чем классический GBM.

3. **Работа с разрежёнными данными**

   В алгоритме XGBoost есть встроенная возможность для обработки пропусков: при встрече с пропущенным значением алгоритм пробует различные варианты разделения и ищет оптимальное.

4. **Кросс-валидация**

   XGBoost позволяет реализовывать кросс-валидацию на каждой итерации алгоритма.

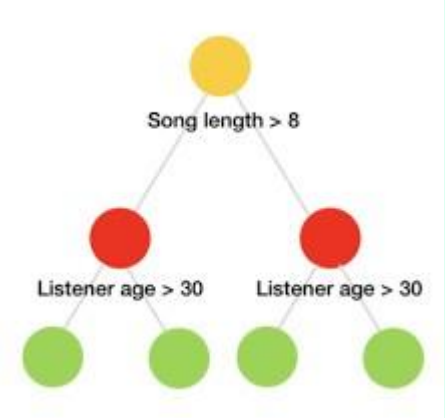
5. **Эффективная обрезка деревьев**

   XGBoost использует для этого более совершенный с точки зрения вычислительной производительности алгоритм.

6. **Аппаратная оптимизация**

> CatBoost — это библиотека градиентного бустинга, созданная Яндексом. Её особенность заключается в том, что в ней используются так называемые небрежные (oblivious) деревья решений, чтобы «вырастить» сбалансированное дерево.

Небрежные деревья — это такие деревья, в которых одни и те же функции используются для левого и правого разбиения на одном уровне дерева:



Одно из главных преимуществ CatBoost заключается в том, что его можно использовать для данных, где категориальные признаки заранее не были преобразованы.

Стекинг

Стекинг — третий вид ансамблирования моделей. Он является абсолютной эвристикой, под которой нет никакого теоретического фундамента — его эффективность можно наблюдать только на практике.

Стекинг — это агрегация ответов моделей машинного обучения. Подход использует понятие **базовых моделей**, каждая из которых обучается независимо от остальных, и **метамодели**, которая использует предсказания базовых моделей как признаки.

Блендинг

Блендинг является простейшей реализацией стекинга.

Формализовать этот **алгоритм** можно следующим образом:

- Пусть у нас есть обучающая выборка (X,y) и тестовая выборка $(X_{\text{test}}, y_{\text{test}})$.
- Мы хотим использовать K базовых моделей: $a_1(x), a_2(x), \ldots, a_K(x)$.
- Делим обучающую выборку на две части: $(X_{\text{train}}, y_{\text{train}})$ и $(X_{\text{meta}}, y_{\text{meta}})$.
- Введём для удобства обозначения $(X_{train}, y_{train}) = A$, $(X_{meta}, y_{meta}) = B$, $(X_{test}, y_{test}) = C$
- Для каждой модели $a_i(x)$:
  - Обучаем модель $a_i(x)$ на подвыборке A.
  - Для каждого объекта из подвыборки B делаем предсказание с помощью $a_i(x)$ — получаем столбец новых признаков для метамодели.
  - Для каждого объекта из подвыборки C делаем предсказание с помощью $a_i(x)$ — получаем ещё один столбец признаков для метамодели.
- Итак, получили матрицу метапризнаков B_{meta} из предсказаний $a_i(x)$ для подвыборки B и матрицу метапризнаков C_{meta} из предсказаний $a_i(x)$ для подвыборки C.
- Обучаем метамодель на подвыборке B_{meta}.
- С помощью обученной метамодели делаем предсказания для всех объектов из C_{meta} — это и будут наши ответы.

Стекинг

К сожалению, у блендинга есть проблема: ни базовые модели, ни метамодель не обучаются на полных данных.

Эту проблему решает **стекинг**.

Алгоритм стекинга следующий:

- Пусть у нас есть обучающая выборка (X,y) и тестовая выборка $(X_{\text{test}}, y_{\text{test}})$.
- Мы хотим использовать K базовых моделей: $a_1(x), a_2(x), \ldots, a_K(x)$.
- Делим обучающую выборку на L частей : A_1, A_2, ..., A_L.
- Для каждой части A_i из L частей обучающей выборки:
  - Обучаем все K базовых моделей на всех частях выборки, кроме A_i.
  - Делаем предсказание для каждого объекта из подвыборки A_i
  - В итоге получаем новые признаки для метамодели.
- Обучаем базовые модели на всей обучающей выборке (X,y), делаем предсказание на тестовой выборке $(X_{\text{test}}, y_{\text{test}})$ и получаем метапризнаки для тестовой выборки — матрицу C_{meta}.
- Обучаем метаалгоритм на новых признаках метамодели из обучающей выборки.
- Делаем предсказание с помощью метаалгоритма для C_{meta} — это и будут наши ответы.