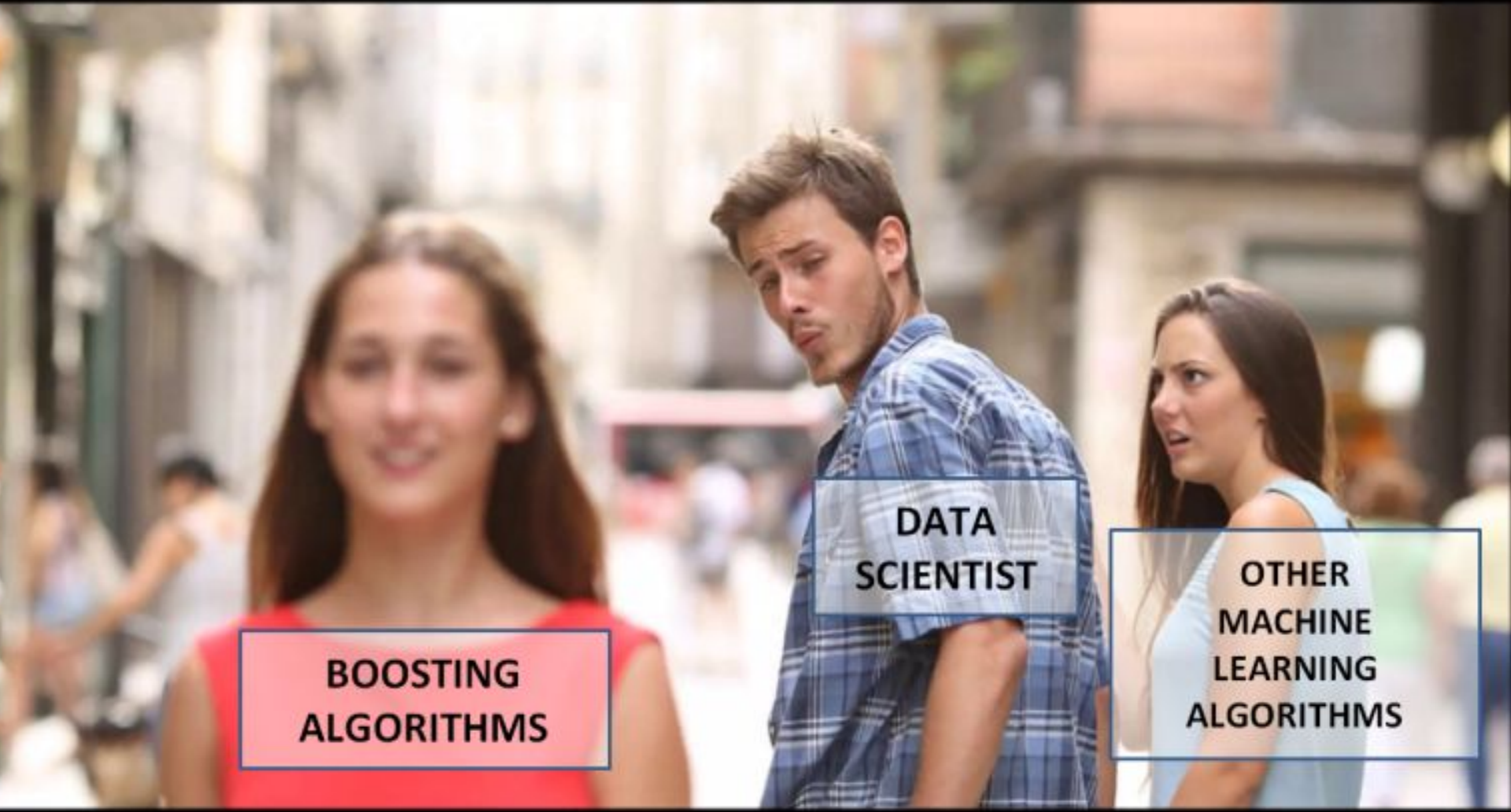

BOOSTING ALGORITHMS
DATA SCIENTIST
OTHER MACHINE LEARNING ALGORITHMS

План уроку

+ Adaboost
+ Градієнтний бустинг
  + Xgboost
  + LightGBM
+ Порівняння AdaBoost, Gradient Boosting (scikit-learn), XGBoost, LightGBM та що в якому випадку використовувати

Алгоритми бустингу

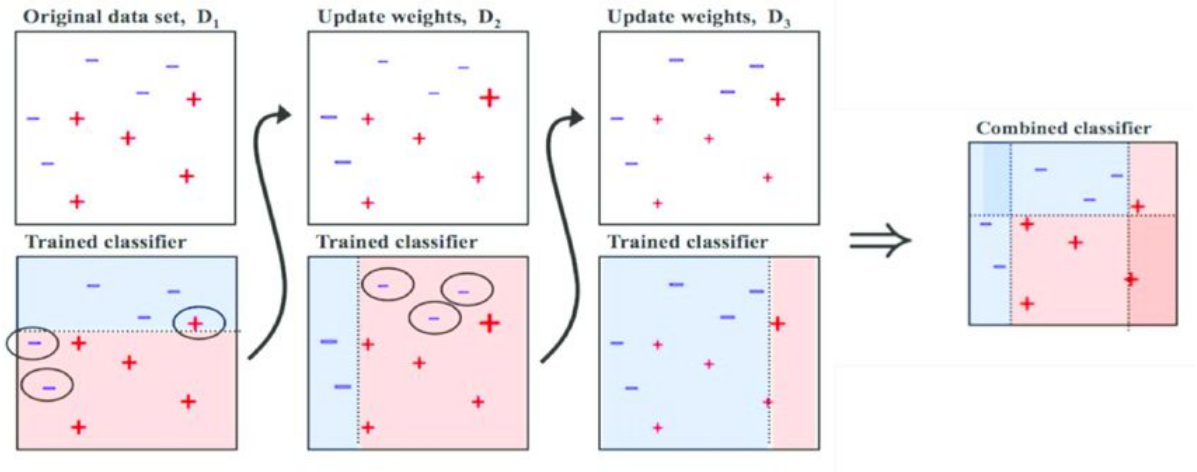
Два основних типи алгоритмів Бустінгу — це Adaptive Boosting та Gradient Boosting.

XGBoost, LightGBM та CatBoost (відомий алгоритм, але розробка яндексу, тому не розглядаємо в курсі) — це різні реалізації Gradient Boosting.

Адаптивний бустинг / «Adaboost»

AdaBoost, скорочення від Adaptive Boosting, є одним із найпопулярніших Boosting алгоритмів.

Він працює шляхом навчання послідовності слабких учнів (часто - пнів - дерев з одним вузлом), де кожен учень зосереджується на виправленні помилок, зроблених його попередником. Остаточний прогноз виходить шляхом поєднання зважених прогнозів кожного слабкого учня.



Adaboost: Принцип роботи

1. **Ініціалізація ваг**:
   - Всі зразки спочатку мають однакові ваги:

$$w_i^{(1)} = \frac{1}{N}, \quad i = 1, 2, \ldots, N$$

   де N — кількість зразків.

2. **Навчання слабких моделей**:
   - Для кожного раунду t (від 1 до T):
     1. Навчити слабкий класифікатор $h_t(x)$ з вагами $w_i^{(t)}$.
     2. Обчислити похибку класифікатора:

$$\epsilon_t = \sum_{i=1}^{N} w_i^{(t)} I(y_i \neq h_t(x_i))$$

   де $I(\cdot)$ — індикаторна функція, яка дорівнює 1, якщо умова виконується, і 0 — якщо ні.

   3. Обчислити вагу класифікатора:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Adaboost: Принцип роботи

1. **Ініціалізація ваг**:
   - Всі зразки спочатку мають однакові ваги:

$$w_i^{(1)} = \frac{1}{N}, \quad i = 1, 2, \ldots, N$$

де N — кількість зразків.

2. **Навчання слабких моделей**:
   - Для кожного раунду t (від 1 до T):
     1. Навчити слабкий класифікатор $h_t(x)$ з вагами $w_i^{(t)}$.
     2. Обчислити похибку класифікатора:

$$\epsilon_t = \sum_{i=1}^{N} w_i^{(t)} I(y_i \neq h_t(x_i))$$

де $I(\cdot)$ — індикаторна функція, яка дорівнює 1, якщо умова виконується, і 0 — якщо ні.

3. Обчислити вагу класифікатора:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

4. Оновити ваги зразків:

$$w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(x_i))$$

де y_i — правильна мітка для зразка x_i.

5. Нормалізувати ваги:

$$w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_{j=1}^{N} w_j^{(t+1)}}$$

3. **Фінальна модель**:
   - Фінальний прогноз:

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Формула фінального прогнозу

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Що означає sign:

- $\text{sign}(z)$ — це сигнум-функція, яка повертає:
  - $+1$, якщо $z > 0$
  - 0, якщо $z = 0$
  - -1, якщо $z < 0$

Як це працює у контексті AdaBoost:

- α_t — вага слабкого класифікатора $h_t(x)$, яка відображає його надійність (чим менша похибка класифікатора, тим більша його вага).
- $h_t(x)$ — прогноз t-го слабкого класифікатора, який зазвичай повертає $+1$ або -1 для класифікації.

Функція sign застосовується до суми зважених прогнозів всіх слабких класифікаторів. Вона визначає, чи є загальний зважений сумарний прогноз позитивним або негативним:

- Якщо сума зважених прогнозів більше нуля ($\sum_{t=1}^{T} \alpha_t h_t(x) > 0$), то sign повертає $+1$, що означає, що остаточний прогноз класифікатора $H(x)$ є позитивним класом.
- Якщо сума менше нуля ($\sum_{t=1}^{T} \alpha_t h_t(x) < 0$), то sign повертає -1, що означає, що остаточний прогноз є негативним класом.

Таким чином, функція sign перетворює суму зважених прогнозів слабких класифікаторів в остаточне рішення, яке належить або до позитивного, або до негативного класу.

Оригінальний пейпер

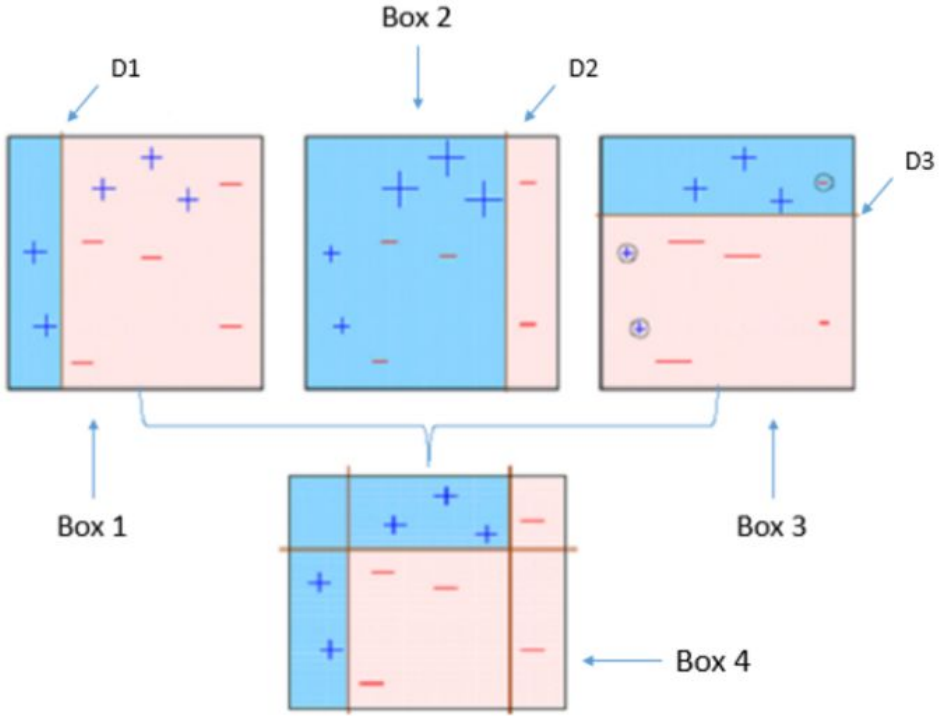
Приклад: бінарна класифікація

1. Алгоритм оновлює ваги об'єктів у наборі даних і навчає нового слабкого учня, звертаючи особливу увагу на спостереження, які були неправильно класифіковані поточною ансамблевою моделлю.

2. Алгоритм додає слабкого учня до зваженої суми відповідно до коефіцієнта оновлення, який виражає ефективність цієї слабкої моделі: чим краще слабкий учень виконав свою роботу, тим більше він буде врахований в сильному учні.

Візуалізація роботи

Adaboost оновлює ваги об'єктів на кожній ітерації. Ваги добре класифікованих об'єктів зменшуються відносно ваг неправильно класифікованих об'єктів. Моделі, які працюють краще, мають більшу вагу в кінцевій моделі ансамблю.



Плюси та мінуси Adaboost

Плюси:

- У AdaBoost є **відносно мало гіперпараметрів**, які потрібно налаштувати для підвищення продуктивності моделі.
- **Покращення точності класифікації:** Алгоритм адаптивно коригує ваги, зосереджуючись на важких для класифікації зразках.
- **Гнучкість у використанні різних слабких моделей:** Можна використовувати будь-які базові класифікатори.

Плюси та мінуси Adaboost

Плюси:

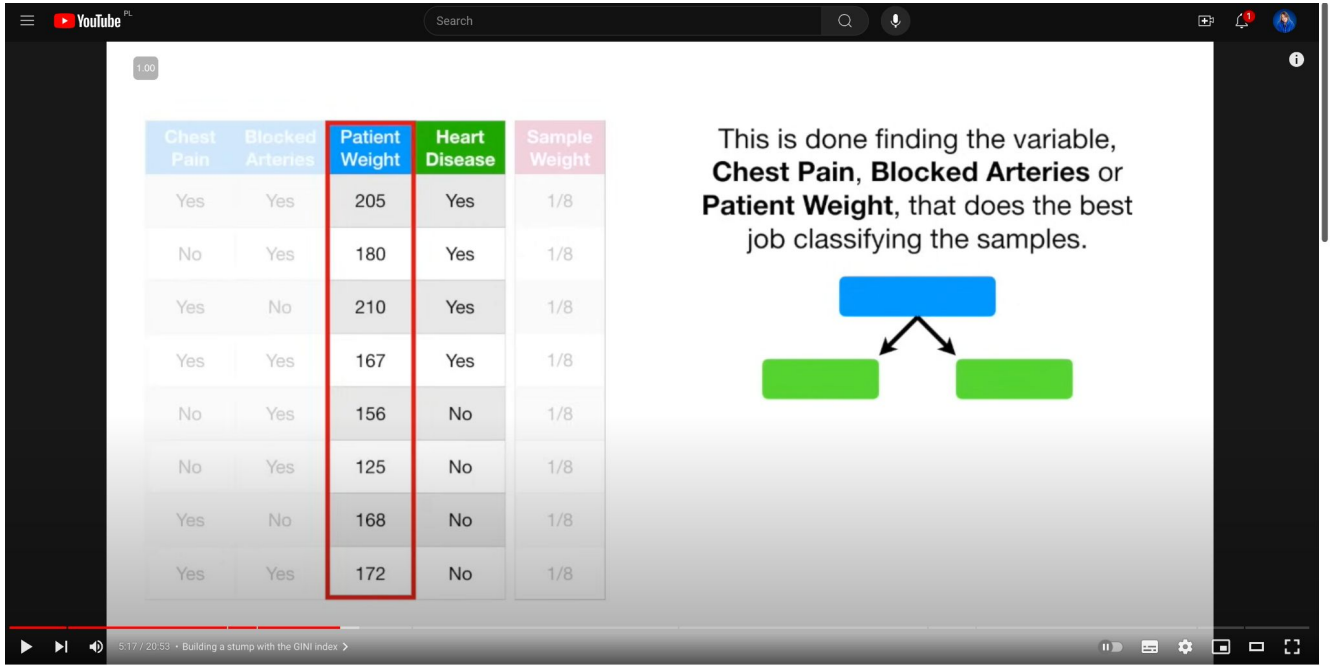
- У AdaBoost є **відносно мало гіперпараметрів**, які потрібно налаштувати для підвищення продуктивності моделі.
- **Покращення точності класифікації:** Алгоритм адаптивно коригує ваги, зосереджуючись на важких для класифікації зразках.
- **Гнучкість у використанні різних слабких моделей:** Можна використовувати будь-які базові класифікатори.

Мінуси AdaBoost:

- **Нестійкий до шуму:** ефективність алгоритму сильно залежить від викидів, оскільки алгоритм намагається ідеально відповідати кожній точці.
- У порівнянні з випадковими лісами та XGBoost, AdaBoost працює гірше, коли включені нерелевантні ознаки.
- AdaBoost не оптимізований за швидкістю.

Де додатково дізнатись про Adaboost

https://youtu.be/LsK-xG1cLYA?si=JZHlUeRKQkLL0ZDF



Градієнтний бустинг

Градієнтний бустинг

При градієнтному бустингу модель ансамблю, яку ми намагаємося побудувати, також представляє собою зважену суму слабких учасників.

Основна відмінність від адаптивного бустингу полягає **в визначенні процесу послідовної оптимізації.**

Теоретичний процес градієнтного спуску за ансамблевою моделлю може бути записаний як

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

де E (.) - помилка підгонки даної моделі, c_l - коефіцієнт, що відповідає розміру кроку, та

$$-\nabla_{s_{l-1}} E(s_{l-1})(.)$$

є антиградієнтом помилки підгонки відносно моделі ансамблю на кроці l_1.

Цей (досить абстрактний) антиградієнт є функцією, яка на практиці може оцінюватися лише для об'єктів у навчальній вибірці (для якої ми знаємо вхідні та вихідні дані). Ці оцінки називаються псевдо-залишками, прикріпленими до кожного об'єкта. Більше того, навіть якщо нам відомі значення цих псевдо-залишків для спостережень, ми не хочемо додавати до нашої моделі ансамблю жодну функцію: ми хочемо додати лише новий екземпляр слабкої моделі.

Отже, природна річ, яку потрібно зробити: слабкий учитель псевдо-залишкам для кожного спостереження. Коефіцієнт c_l обчислюється згідно з одновимірним процесом оптимізації (лінійний пошук для отримання найкращого розміру кроку c_l).

Кроки алгоритму

Припустимо, ми хочемо використати градієнтний бустінг з родиною слабких моделей. На початку алгоритму (перша модель послідовності) псевдо-залишки встановлюються рівними значенням об'єктів. Потім ми повторюємо L разів (для L моделей послідовності) наступні кроки:

Навчити кращого можливого слабкого учня псевдо-залишкам (наблизити антиградієнт відносно поточного сильного учня).
Визначити значення оптимального розміру кроку, який визначає, наскільки ми оновлюємо модель ансамблю в напрямку нового слабкого учня.
Оновити модель ансамблю, додавши нового слабкого учня, помноженого на розмір кроку (зробити крок градієнтного спуску).
Визначити нові псевдо-залишки, які показують для кожного спостереження, в якому напрямку ми б хотіли оновити наступні прогнози моделі ансамблю.

Повторюючи ці кроки, ми послідовно будуємо наші L моделі та агрегуємо їх відповідно до підходу градієнтного спуску.

Принцип роботи

На самому початку алгоритму градієнтного бустингу, коли ще немає жодної моделі, робиться початковий прогноз. Цей прогноз визначається як значення c, яке мінімізує функцію втрат $L(y_i, c)$ для всіх зразків у наборі даних. Тут c є константою, яка представляє найкраще фіксоване передбачення для цільової змінної на початковому етапі.

Приклад функцій втрат і значення c:

1. **Регресія з квадратичною функцією втрат**:
   - Функція втрат:
   $$L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$
   - Початковий прогноз c буде середнім значенням цільової змінної:
   $$c = \frac{1}{N} \sum_{i=1}^{N} y_i$$
2. **Класифікація з логістичною функцією втрат**:
   - Функція втрат:
   $$L(y_i, \hat{y}_i) = -\left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right)$$
   - Початковий прогноз c буде логарифмом odds:
   $$c = \log\left(\frac{\text{mean}(y)}{1 - \text{mean}(y)}\right)$$

Принцип роботи

1. **Ініціалізація**:

   - Початковий прогноз $\hat{y}_i^{(0)}$ для всіх i:

$$\hat{y}_i^{(0)} = c = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

   де $L(y_i, c)$ — функція втрат, а c — константа, яка мінімізує втрати.

2. **Обчислення залишків**:

   - Для кожного зразка i в раунді t, залишок $r_i^{(t)}$ обчислюється як негативний градієнт функції втрат:

$$r_i^{(t)} = -\left[\frac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}\right]$$

   Наприклад, для задачі регресії з квадратичною функцією втрат $L(y, \hat{y}) = (y - \hat{y})^2$, залишок буде:

$$r_i^{(t)} = y_i - \hat{y}_i^{(t-1)}$$

Принцип роботи

3. **Навчання моделі на залишках**:

   - Нова слабка модель $h_t(x)$ навчається на даних, де цільовими значеннями є залишки $r_i^{(t)}$. Це означає, що модель $h_t(x)$ намагається передбачити залишки, тобто вона навчається на помилках попередньої моделі:

$$h_t(x) = \arg\min_h \sum_{i=1}^{N} \left(r_i^{(t)} - h(x_i) \right)^2$$

     Вона намагається знайти функцію $h_t(x)$, яка максимально зменшує залишки.

4. **Оновлення прогнозів**:

   - Після навчання нової моделі $h_t(x)$, її прогнози додаються до поточних прогнозів з урахуванням швидкості навчання:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \nu h_t(x_i)$$

     Це оновлення коригує поточні прогнози на основі передбачених залишків, поступово зменшуючи загальну похибку.

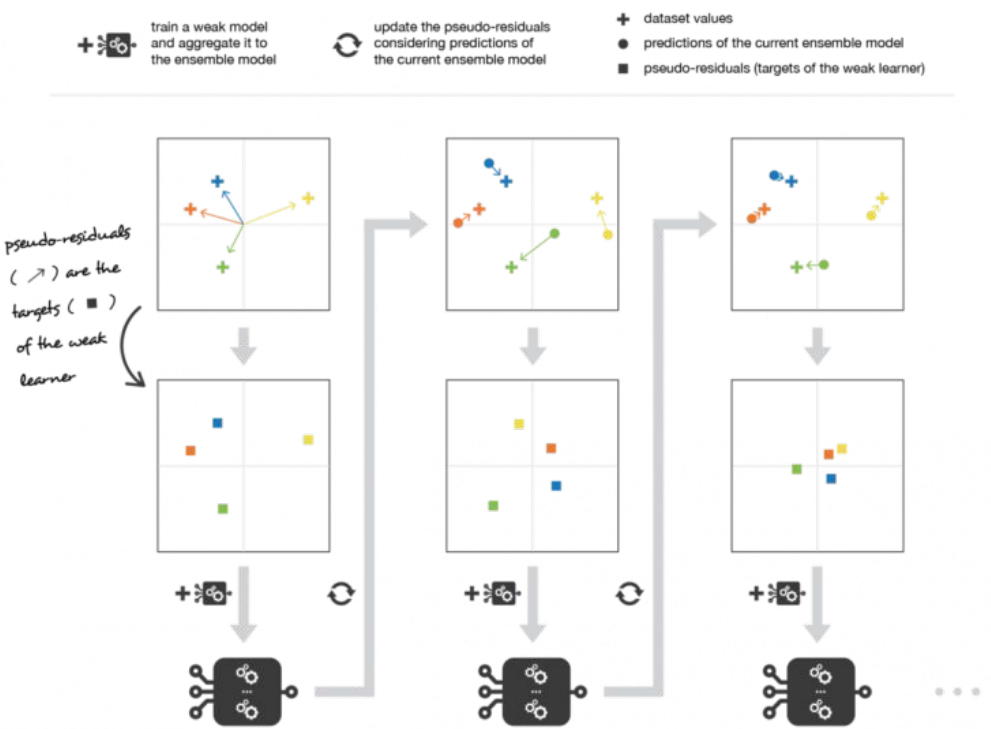
Adaboost та Градієнтний бустинг

Адаптивний бустінг намагається вирішити на кожній ітерації саме «локальну» задачу оптимізації (знайти найкращого слабкого учителя і його коефіцієнт, який потрібно додати до сильної моделі), тоді як градієнтний бустінг використовує замість цього підхід з градієнтним спуском і його легше адаптувати до великої кількості функцій втрат.

Отже, градієнтний бустінг можна розглядати як узагальнення adaboost для довільних диференційовних функцій втрат.

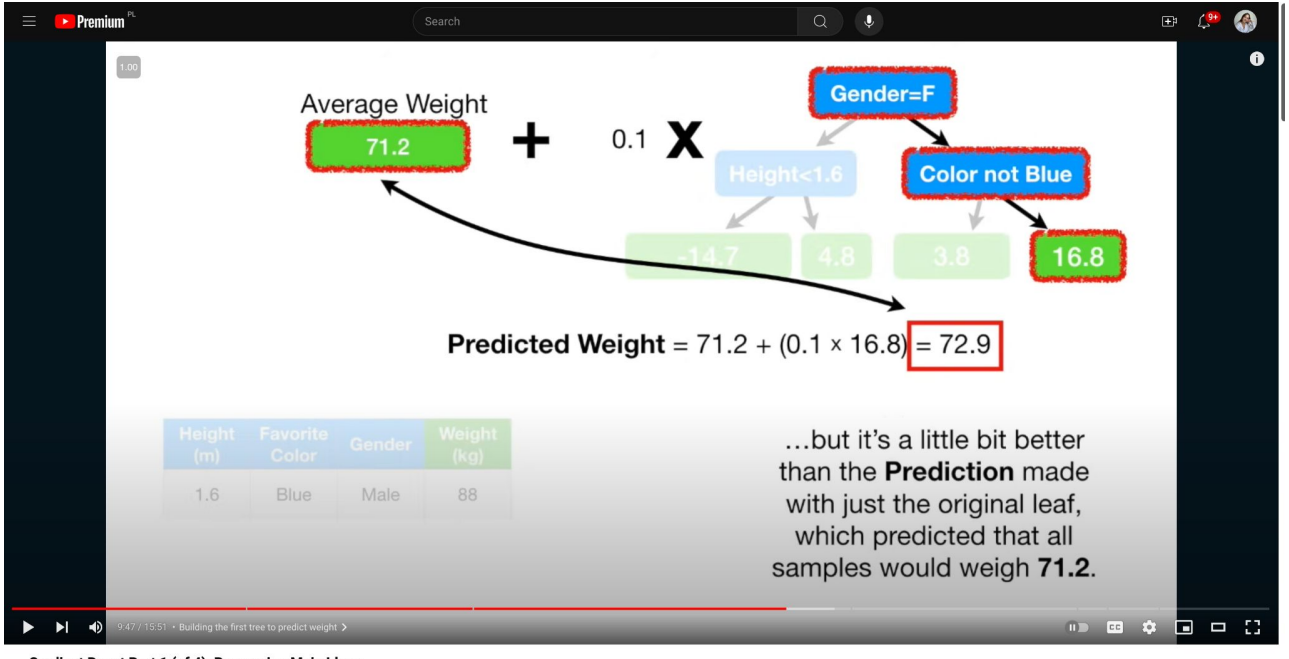
Візуалізація процесу

Градієнтний бустінг оновлює значення спостережень на кожній ітерації. Слабкі учні навчаються підганятися під залишки, які показують, в якому напрямку коригувати прогнози поточної моделі ансамблю, щоб знизити помилку.



Де додатково дізнатись про градієнтний бустинг

https://youtu.be/3CC4N4z3GJc?si=x-pp_ZK6XOJGt5aK



Переваги градієнтного бустингу

1. **Гнучкість у виборі функції втрат:** Градієнтний бустинг дозволяє використовувати різні функції втрат, що робить його універсальним для різних типів задач (регресія, класифікація, ранжування).

2. **Висока точність:** За рахунок послідовного навчання моделей на залишках попередніх моделей, градієнтний бустинг може досягати високої точності.

3. **Регулювання внеску кожної моделі:** Використання параметру швидкості навчання (learning rate) дозволяє краще контролювати внесок кожної нової моделі, що допомагає уникати переобучення.

Adaboost vs Gradient Boosting

Порівняльна Таблиця

Параметр	AdaBoost	Градієнтний Бустинг
Простота реалізації	Легка	Складніша
Точність	Висока	Висока
Чутливість до шуму	Висока	Висока
Обчислювальні витрати	Відносно невеликі	Високі
Гнучкість у виборі моделі	Підтримує різні слабкі моделі	Підтримує різні функції втрат і моделі
Налаштування гіперпараметрів	Менш чутливий до гіперпараметрів	Сильно залежить від гіперпараметрів

Найбільш популярні реалізації град. бустингу - це XGBoost та LightGBM

XGBoost

XGBoost (eXtreme Gradient Boosting) - це оптимізована розподілена бібліотека градієнтного бустингу, розроблена для забезпечення високої ефективності та гнучкості обчислень.

Загальна документація: https://xgboost.readthedocs.io/en/latest/

Для Python: https://xgboost.readthedocs.io/en/latest/python/index.html

Пошук оптимальних гіперпараметрів:
https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Основні математичні особливості XGBoost

1. **Цільова Функція (Objective Function)**:
   XGBoost мінімізує цільову функцію, яка включає функцію втрат і регуляризацію:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t)$$

   де $L(y_i, \hat{y}_i)$ — функція втрат (наприклад, квадратична помилка для регресії), а $\Omega(f_t)$ — регуляризація.

2. **Регуляризація**:
   Регуляризація зменшує складність моделі для запобігання переобученню:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

   де γ — параметр для кількості листів у дереві, λ — параметр регуляризації для ваг.

Основні математичні особливості XGBoost

3. **Формулювання Буста**:

   XGBoost використовує другий порядок похідних для оптимізації цільової функції:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{N} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

   де $g_i = \frac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ і $h_i = \frac{\partial^2 L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)^2}}$ — перший і другий порядки похідних функції втрат.

4. **Оновлення Ваг**:

   Ваги оновлюються з урахуванням регуляризації та другого порядку похідних:

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

   де I_j — набір індексів зразків у j-му листі.

5. **Розбиття Вузлів (Split Finding)**:

   XGBoost використовує жадібне алгоритмічне розбиття для вибору найкращого розбиття вузлів на основі приросту інформації:

$$\text{Gain} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

   де I_L і I_R — лівий і правий вузли після розбиття.

Оригінальний пейпер

Переваги XGBoost

1. **Ефективність та Швидкість**:
    - Паралельне обчислення: XGBoost використовує паралельні обчислення для побудови дерев.
    - Sparsity-aware: Обробка розріджених даних ефективно використовуючи спеціальні техніки.
2. **Висока Точність**:
    - Використання другого порядку похідних покращує точність оптимізації.
    - Вбудована регуляризація зменшує переобучення.
3. **Гнучкість**:
    - Підтримка різних функцій втрат (регресія, класифікація, ранжування).
    - Можливість налаштування багатьох гіперпараметрів.
4. **Масштабованість**:
    - Можливість обробки великих наборів даних завдяки ефективному використанню пам'яті.

Недоліки XGBoost

1. **Складність Налаштування**:
    - Багато гіперпараметрів, які потрібно налаштовувати для досягнення оптимальної продуктивності.
2. **Вимоги до Обчислювальних Ресурсів**:
    - Хоча XGBoost є ефективним, він все ще може вимагати значних обчислювальних ресурсів, особливо для великих наборів даних.

LightGBM

LightGBM (Light Gradient Boosting Machine) — це бібліотека градієнтного бустингу, яка оптимізована для швидкого та ефективного навчання на великих наборах даних.

LightGBM

1. **Побудова дерев за принципом leaf-wise (Leaf-wise Tree Growth)**

   - **Метод**: LightGBM будує дерева за принципом leaf-wise. Це означає, що на кожній ітерації алгоритм додає вузол до листа, який має найбільшу помилку.
   - **Перевага**: Такий підхід дозволяє моделі бути більш точною, оскільки кожне розбиття максимально зменшує залишкову помилку.

2. **Ексклюзивне згортання ознак (Exclusive Feature Bundling, EFB)**

   - **Метод**: Ознаки, які рідко мають ненульові значення одночасно, групуються разом. Це зменшує кількість ознак і, відповідно, кількість розбиттів, які потрібно перевірити.
   - **Перевага**: Така техніка зменшує обчислювальні витрати і обсяг пам'яті, необхідної для навчання.

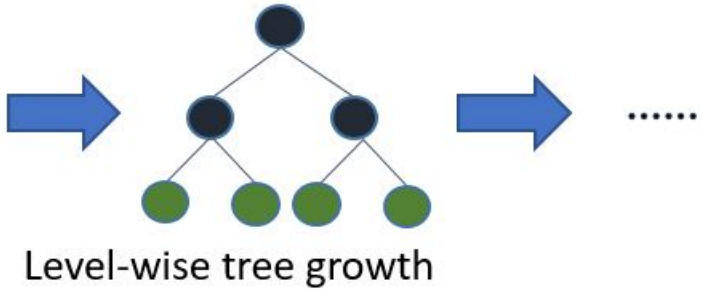
LightGBM

3. **Градієнтний вибір на одній стороні (Gradient-based One-Side Sampling, GOSS)**

    - **Метод**: На кожній ітерації алгоритму зразки з найбільшими градієнтами мають більший вплив на навчання, тому вони обираються з більшою ймовірністю. Зразки з меншими градієнтами вибираються випадковим чином для зменшення загального об'єму даних.
    - **Перевага**: Це дозволяє зменшити об'єм даних для кожної ітерації, прискорюючи процес навчання без значної втрати точності.

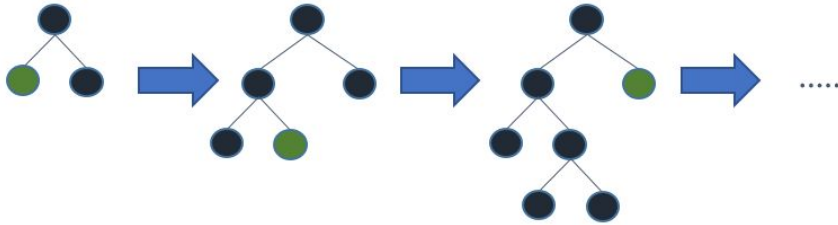
Level-wise Tree Growth (XGBoost)

1. **Метод**:
   - На кожній ітерації додаються нові вузли до всіх листів поточного рівня дерева одночасно. Це означає, що дерево росте рівномірно на всіх гілках.
2. **Перевага**:
   - Такий підхід дозволяє створювати більш збалансовані дерева, що може бути корисним для зменшення переобучення і забезпечення стабільності моделі.
3. **Недолік**:
   - Додавання вузлів до всіх листів одночасно може бути менш ефективним з точки зору зменшення залишкової помилки, оскільки він не зосереджується на найбільших помилках так, як leaf-wise підхід.



Leaf-wise (Best-first) Tree Growth (LightGBM)

1. **Метод**:
    - На кожній ітерації додається новий вузол до листа, який має найбільшу помилку (найбільший залишок). Це означає, що на кожній ітерації дерево може ставати глибшим тільки в одному місці.
2. **Перевага**:
    - Такий підхід дозволяє більш точно адаптуватися до складних структур даних, оскільки він зосереджується на найбільших помилках. Це може призвести до більш точних моделей.
3. **Недолік**:
    - Дерева, побудовані за таким принципом, можуть бути дуже незбалансованими і мати велику глибину в окремих місцях, що може збільшити ризик переобучення.



Leaf-wise tree growth

Окей, то що коли використовуємо?

Порівняння бустинг моделей за особливостями

Параметр	AdaBoost	Gradient Boosting (scikit-learn)	XGBoost	LightGBM
Основна концепція	Комбінує слабкі моделі, підвищуючи вагу помилок	Комбінує слабкі моделі, мінімізуючи функцію втрат	Оптимізований градієнтний бустинг з регуляризацією	Оптимізований градієнтний бустинг для великих даних
Метод побудови дерев	Залежить від базової моделі	Level-wise	Level-wise	Leaf-wise
Швидкість навчання	Помірна	Помірна	Висока	Дуже висока
Використання пам'яті	Низьке	Помірне	Помірне	Низьке
Точність	Висока на даних без шуму	Висока	Дуже висока	Дуже висока
Стійкість до шуму	Низька	Помірна	Помірна	Помірна
Чутливість до гіперпараметрів	Низька	Висока	Висока	Висока
Масштабованість	Погана для великих наборів даних	Помірна	Висока	Дуже висока

Рекомендації з використання

Прикладний випадок	AdaBoost	Gradient Boosting (scikit-learn)	XGBoost	LightGBM
Малі та середні набори даних	Підходить, якщо дані без шуму	Підходить, особливо якщо потрібно налаштовувати функцію втрат	Підходить, висока точність	Може бути зайвим, якщо дані невеликі
Великі набори даних	Не рекомендується	Може бути повільним	Підходить, але потребує налаштування	Дуже підходить, висока швидкість
Дані з шумом	Не рекомендується, переобучення	Помірна стійкість	Помірна стійкість	Помірна стійкість
Час критичний (потрібна швидкість)	Не підходить	Може бути повільним	Підходить, швидке навчання	Дуже підходить, оптимізоване навчання
Модель з високою точністю	Підходить, якщо дані чисті	Підходить, висока точність	Дуже підходить, висока точність	Дуже підходить, висока точність
Простота налаштування	Підходить, мало гіперпараметрів	Потребує налаштування багатьох параметрів	Потребує налаштування багатьох параметрів	Потребує налаштування багатьох параметрів
Розподілені обчислення	Не підходить	Може бути складним	Підходить, підтримує розподілені обчислення	Дуже підходить, підтримує розподілені обчислення

Висновок

1. **AdaBoost**: Найкраще підходить для малих (< 10 000 зразків) і середніх наборів (< 100 000 зразків) даних без шуму. Він простий у налаштуванні і дає високу точність на чистих даних.
2. **Gradient Boosting (scikit-learn)**: Хороший вибір для малих і середніх наборів даних, коли потрібна гнучкість у виборі функції втрат. Вимагає налаштування гіперпараметрів.
3. **XGBoost**: Висока продуктивність і точність, підходить для великих наборів даних. Потребує ретельного налаштування гіперпараметрів, але забезпечує швидке навчання і підтримує розподілені обчислення.
4. **LightGBM**: Ідеальний для дуже великих (> 1 000 000 зразків) наборів даних, де важлива швидкість навчання і ефективність використання пам'яті. Дуже висока точність, але потребує ретельного налаштування гіперпараметрів.

Вибір алгоритму бустингу з точки зору задачі

Параметр/Задача	AdaBoost	Gradient Boosting (scikit-learn)	XGBoost	LightGBM
Регресія	Підходить для простих задач	Добре підходить	Дуже добре підходить, висока точність	Дуже добре підходить, висока точність
Класифікація	Висока точність, чутливий до шуму	Висока точність	Дуже висока точність, стійкість до шуму	Дуже висока точність, стійкість до шуму
Ранжування	Не підтримується	Обмежена підтримка	Підтримується, дуже добре підходить	Підтримується, дуже добре підходить
Робота з категоріальними даними	Необхідне попереднє кодування	Необхідне попереднє кодування	Вбудована підтримка	Вбудована підтримка
Заповнення пропущених значень	Не підтримується	Не підтримується	Вбудована підтримка	Вбудована підтримка