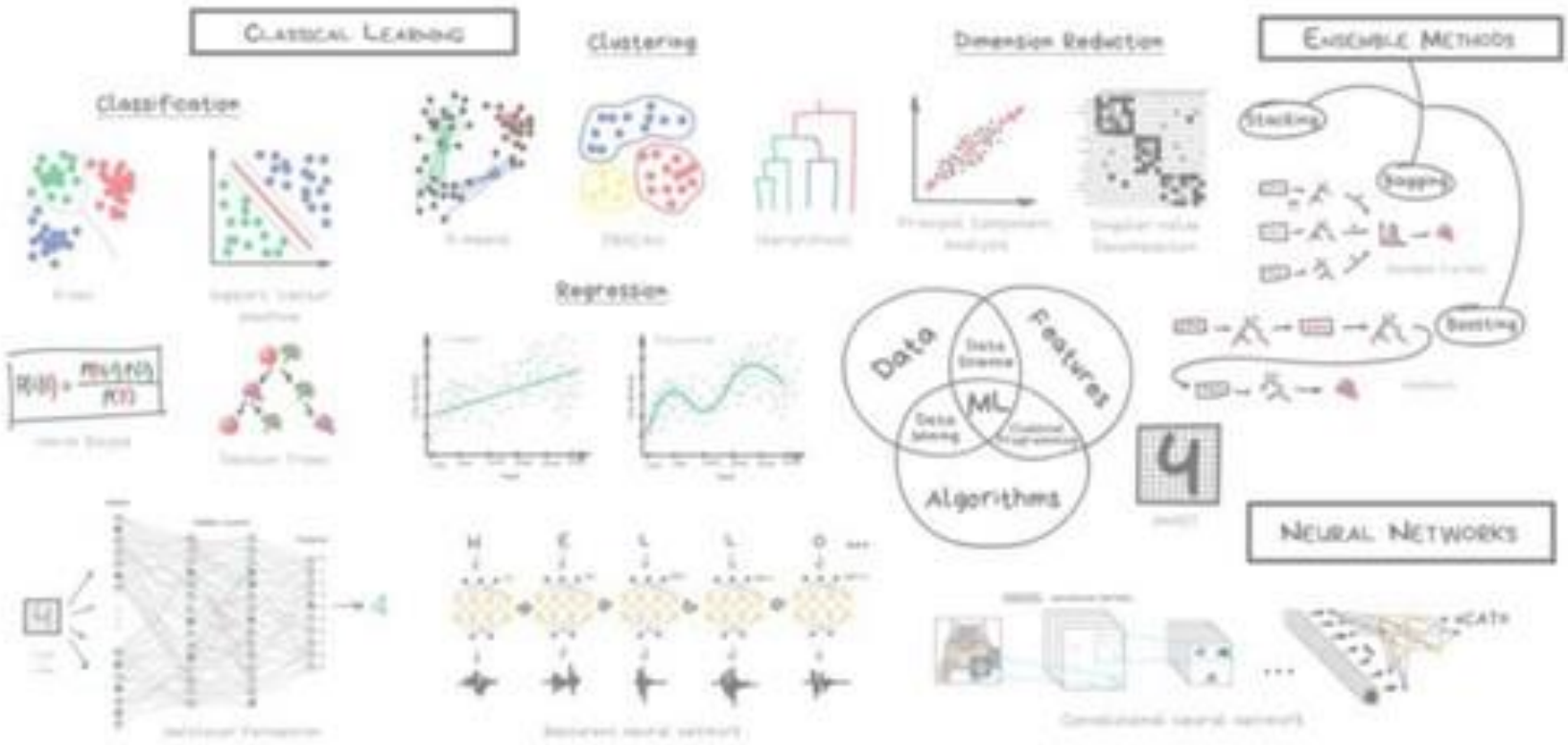
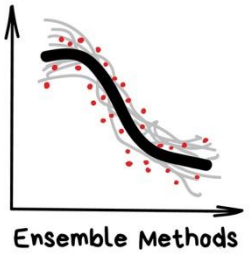
МАШИННЕ НАВЧАННЯ



Ансамблеві методи

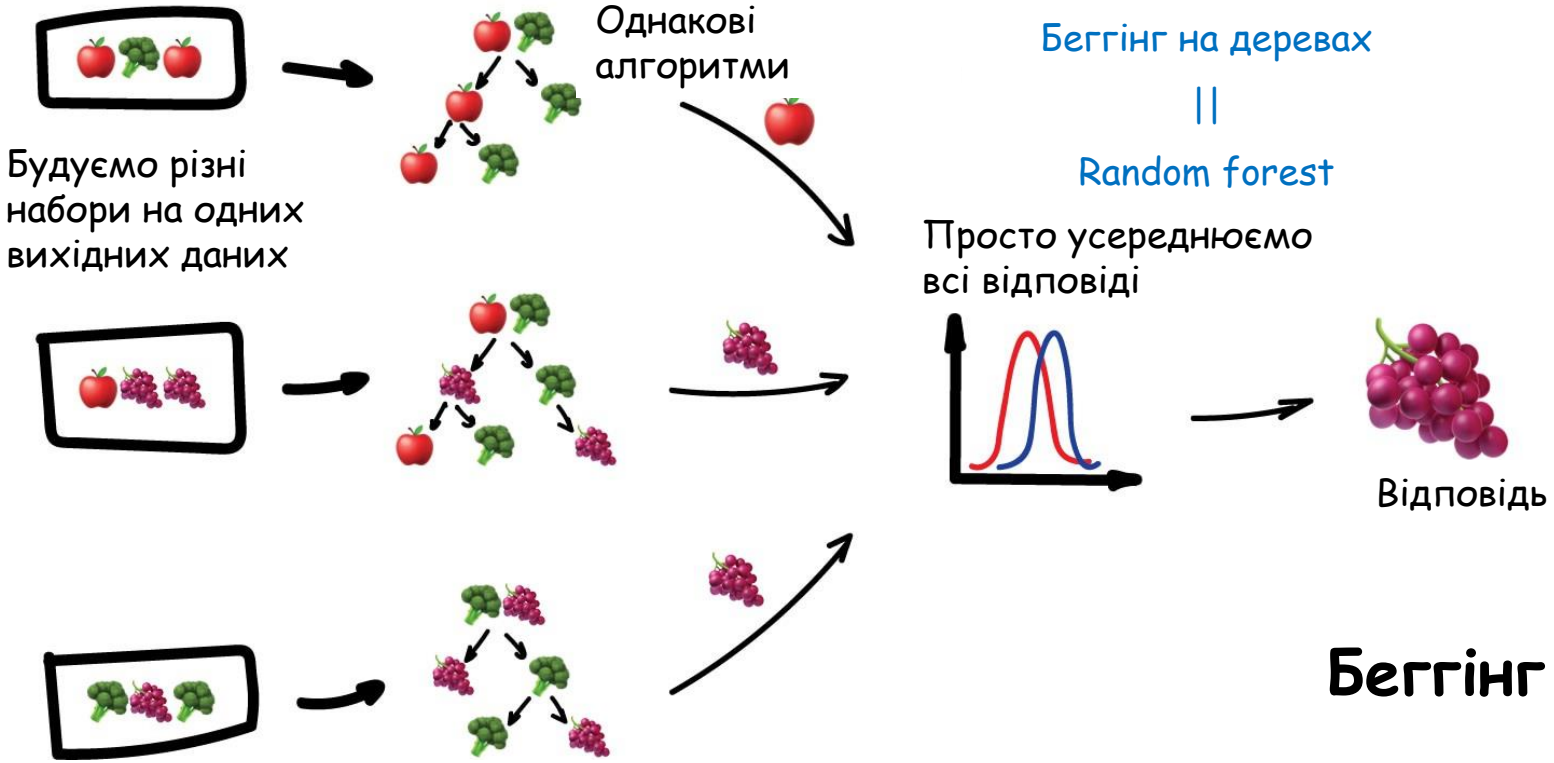
Лекція № 14

Випадковий ліс
Random forest

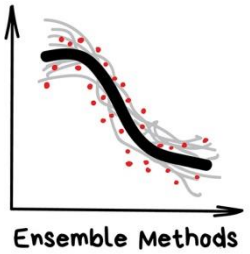


Окремий випадок беггінгу, який виявився досить продуктивним та успішним – випадковий ліс. Один з універсальних методів машинного навчання.

«Випадковий» тому що він заснований на деревах що приймають рішення, які в дусі беггінгу навчаються на випадкових підвибірках.



Випадковий ліс
Random forest



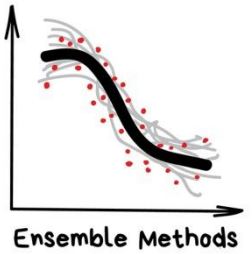
Навчання випадкового лісу:
Беггінг над деревами рішень
Ознаку в кожній вершині дерева обирають з випадкової підмножини k з n ознак. За замовчуванням розділяють:

- $k = \lfloor n/3 \rfloor$ – для регресії
- $k = \lfloor \sqrt{n} \rfloor$ – для класифікації

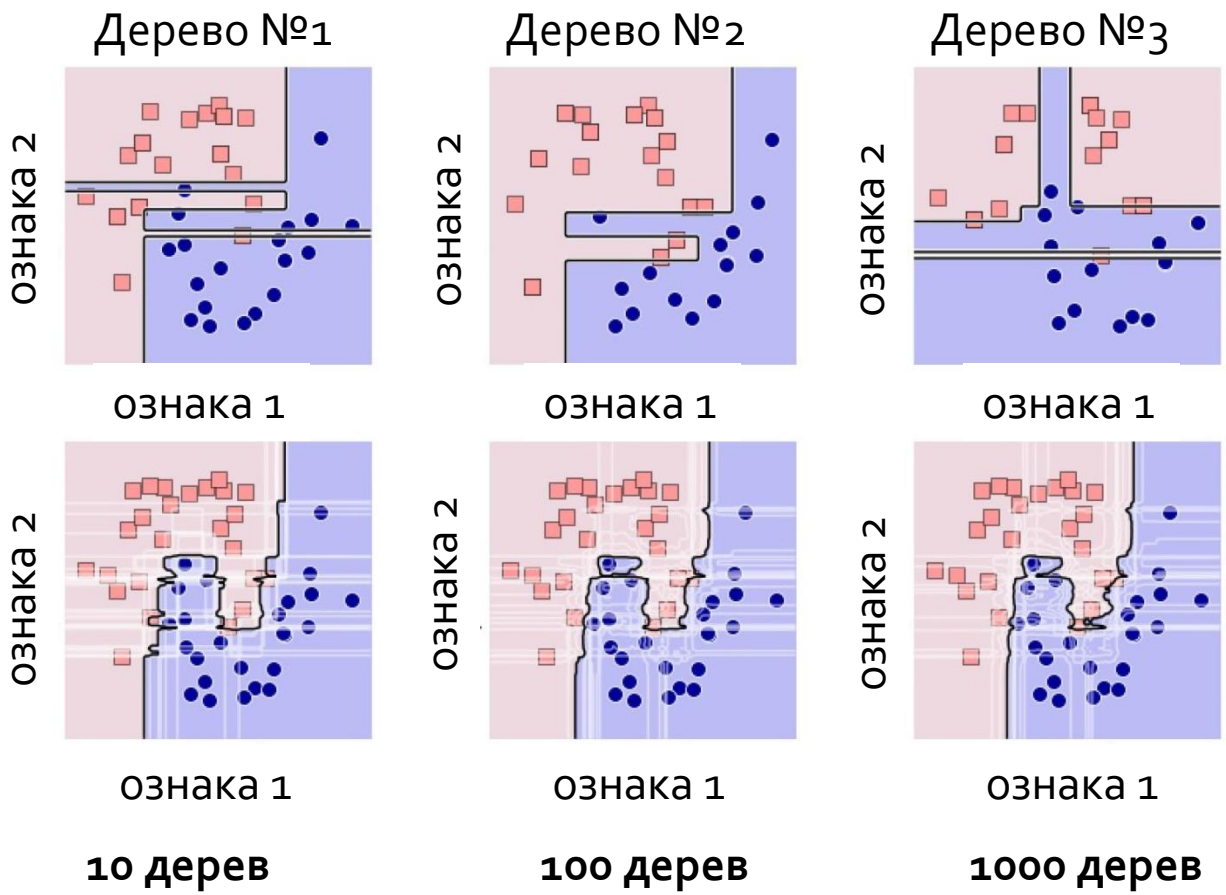
Параметри, які можна настроювати, зокрема по **OOB**

- Кількість **T** дерев
- Кількість **k** випадкових ознак
- Максимальна глибина дерев
- Мінімальна кількість об'єктів у підвибірці що ділиться
- Мінімальна кількість об'єктів у листях
- Критерій ділення: **MSE** для регресії; ентропія або критерій Джині для класифікації

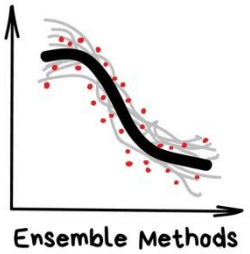
Поступове зглажування разділяючої поверхні



Приклад розділення вибірки за допомогою лісу дерев



Приклад



Припустимо, у нас є дані по мільйону музичних кліпів на Ютубі. По кожному є 100 критеріїв, наприклад:

Чи триває кліп довше, ніж три хвилини.

Чи є там пряма діжка.

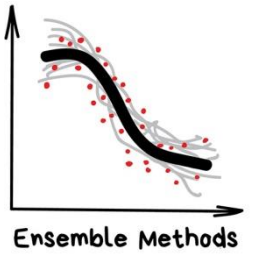
Цей трек у жанрі «хіп-хоп» чи ні.

Чи випустив кліп популярний лейбл.

Чи записано кліп у дворі на мобільний телефон.

Також у нас є дані про те, чи набрали кліп більше мільйона переглядів. Ми хочемо навчитися передбачати цей критерій — назвемо його популярністю. Тобто ми хочемо отримати якийсь алгоритм, якому на вхід подаєш 100 критеріїв кліпу у форматі так/ні, а на виході він тобі каже: «Цьому кліпу судилося стати популярним».

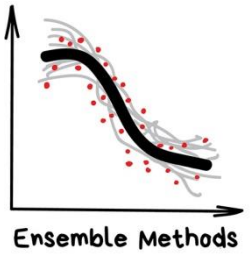
Приклад



Клип	Критерий 1	Критерий 2	Критерий 3	Критерий 4	Критерий 5	...	Критерий 100	Есть 1 000 000?
1	Да	Да	Нет	Да	Да	...	Нет	Нет
2	Нет	Да	Нет	Нет	Да	...	Да	Нет
3	Да	Да	Да	Нет	Да	...	Нет	Нет
4	Нет	Нет	Да	Да	Нет	...	Да	Да
5	Да	Нет	Нет	Да	Да	...	Нет	Нет
6	Нет	Да	Да	Нет	Нет	...	Нет	Нет
7	Нет	Да	Нет	Нет	Да	...	Нет	Да
8	Нет	Нет	Нет	Нет	Нет	...	Да	Нет
9	Да	Да	Нет	Нет	Нет	...	Нет	Нет
10	Нет	Да	Да	Да	Да	...	Нет	Нет
11	нет	Нет	Нет	Нет	Да	...	Нет	Нет
12	Да	Нет	Да	Нет	Нет	...	Да	Да
...	...	...	...	...	...	...	...	...
1 000 000	Нет	Да	Да	Нет	Да	...	Да	Нет

Візьмемо випадкову вибірку з наших вихідних даних. Не мільйон кліпів, а 10 000. До них — випадковий набір критеріїв, не всі 100, а 5:

Клип	Критерий 1	Критерий 2	Критерий 3	Критерий 4	Критерий 5	...	Критерий 100	Есть 1 000 000?
1	Да	Да	Нет	Да	Да	...	Нет	Нет
2	Нет	Да	Нет	Нет	Да	...	Да	Нет
3	Да	Да	Да	Нет	Да	...	Нет	Нет
4	Нет	Нет	Да	Да	Нет	...	Да	Да
5	Да	Нет	Нет	Да	Да	...	Нет	Нет
6	Нет	Да	Да	Нет	Нет	...	Нет	Нет
7	Нет	Да	Нет	Нет	Да	...	Нет	Да
8	Нет	Нет	Нет	Нет	Нет	...	Да	Нет
9	Да	Да	Нет	Нет	Нет	...	Нет	Нет
10	Нет	Да	Да	Да	Да	...	Нет	Нет
11	нет	Нет	Нет	Нет	Да	...	Нет	Нет
12	Да	Нет	Да	Нет	Нет	...	Да	Да
...	...	...	...	...	...	...	...	...
1 000 000	Нет	Да	Да	Нет	Да	...	Да	Нет

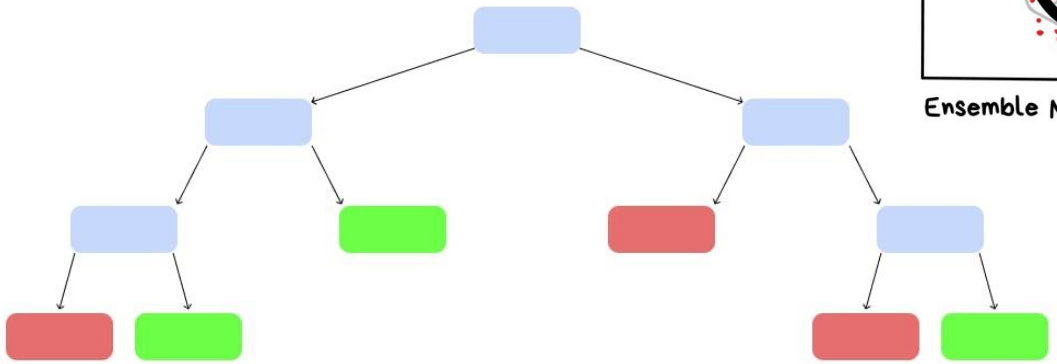


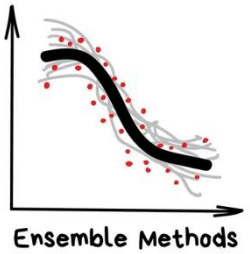
Беремо ці кліпи

Та ці критерії

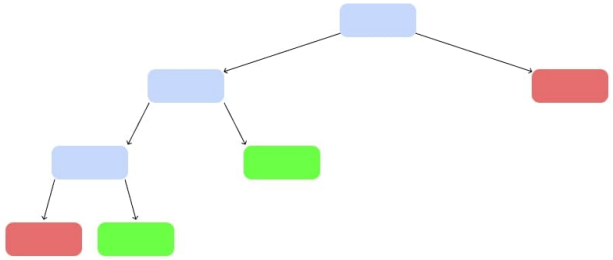
Клип	Критерий 1	Критерий 2	Критерий 3	Критерий 4	Критерий 5	...	Критерий 100	Есть 1 000 000?
1	Да	Да	Нет	Да	Да	...	Нет	Нет
2	Нет	Да	Нет	Нет	Да	...	Да	Нет
3	Да	Да	Да	Нет	Да	...	Нет	Нет
4	Нет	Нет	Да	Да	Нет	...	Да	Да
5	Да	Нет	Нет	Да	Да	...	Нет	Нет
6	Нет	Да	Да	Нет	Нет	...	Нет	Нет
7	Нет	Да	Нет	Нет	Да	...	Нет	Да
8	Нет	Нет	Нет	Нет	Нет	...	Да	Нет
9	Да	Да	Нет	Нет	Нет	...	Нет	Нет
10	Нет	Да	Да	Да	Да	...	Нет	Нет
11	нет	Нет	Нет	Нет	Да	...	Нет	Нет
12	Да	Нет	Да	Нет	Нет	...	Да	Да
...	...	...	...	...	...	...	...	...
1 000 000	Нет	Да	Да	Нет	Да	...	Да	Нет

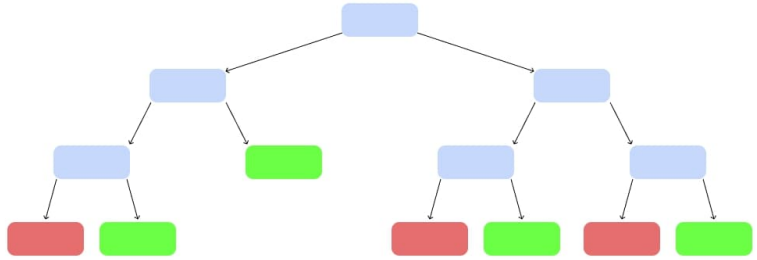
Будуємо дерево

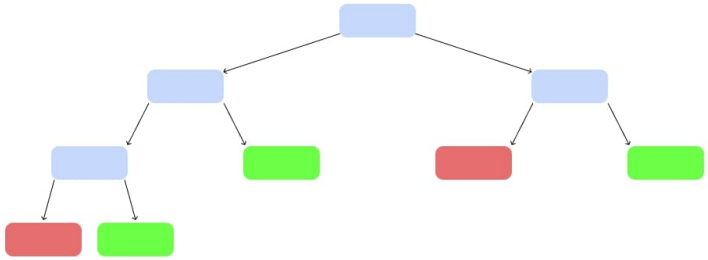


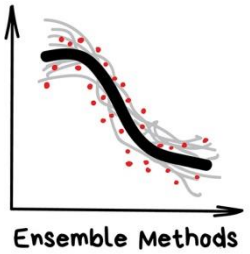


Так будуємо ще де-кілька дерев, кожне — на своєму наборі даних та своєму наборі критеріїв

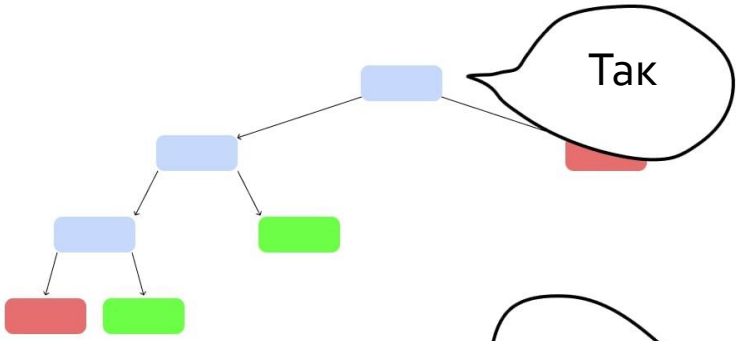


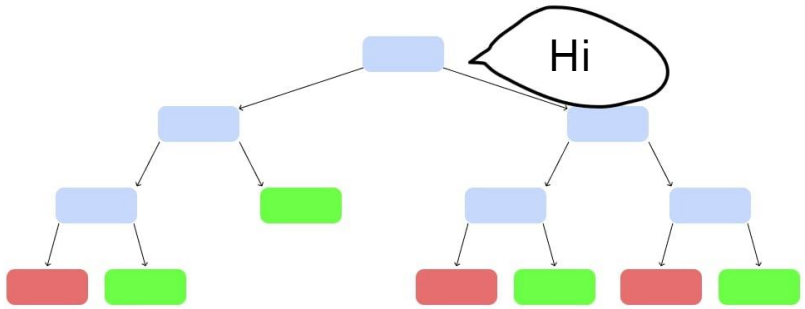


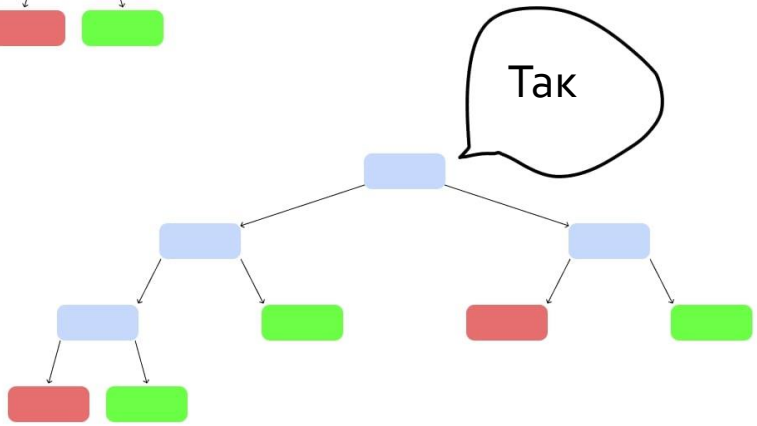


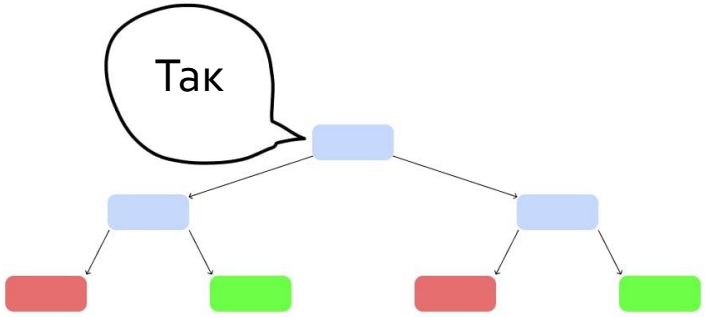


У нас з'явився випадковий ліс.
Випадковий — бо ми щоразу брали рандомний набір даних та критеріїв.
Ліс — бо багато дерев.
Тепер запустимо кліп, якого не було в навчальній вибірці.Кожне дерево видасть свій вердикт, чи стане він популярним — так чи ні. Як голосування на виборах. Вибираємо варіант, який отримає найбільше голосів.



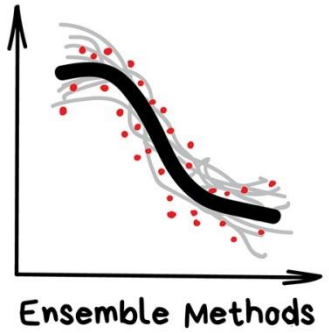




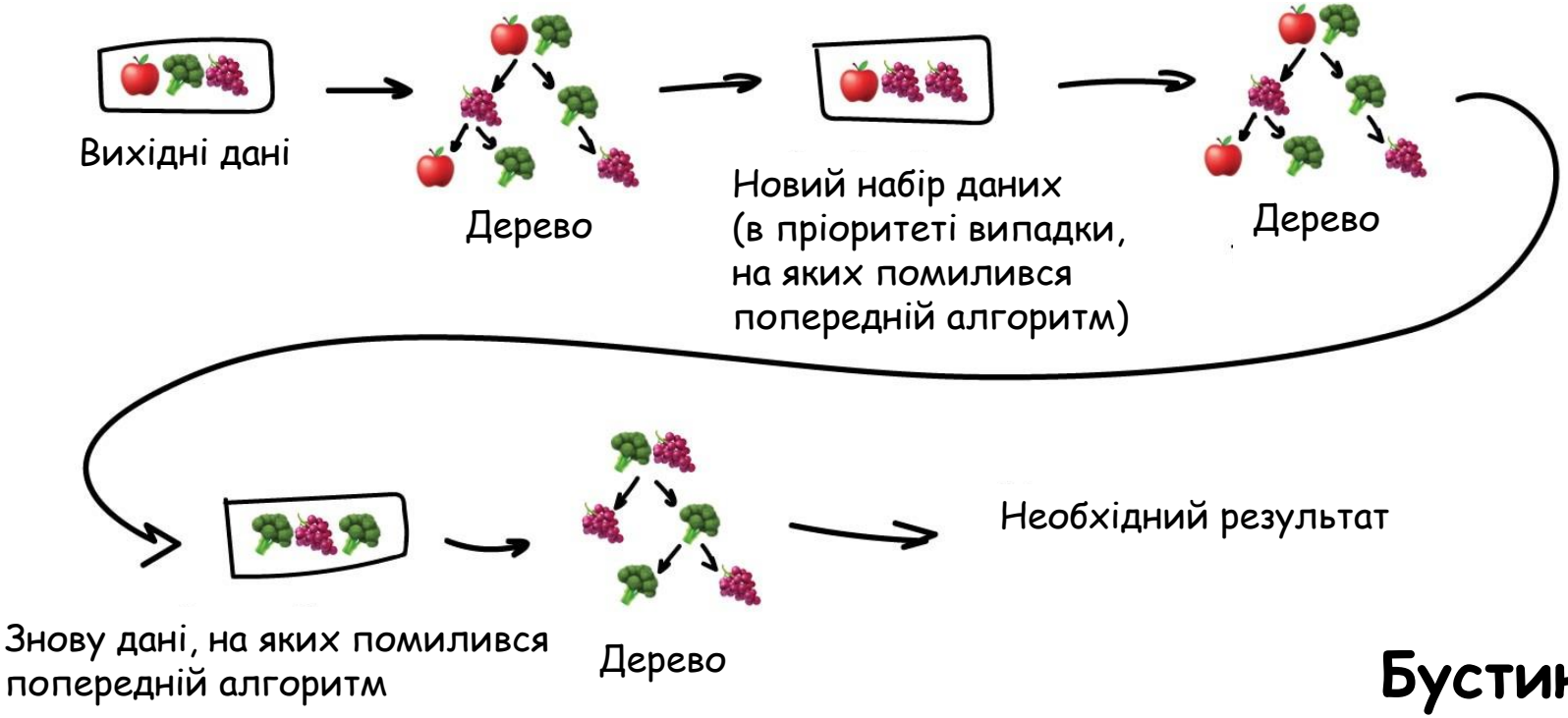


Ансамблеві методи

Бустинг = **Boosting**



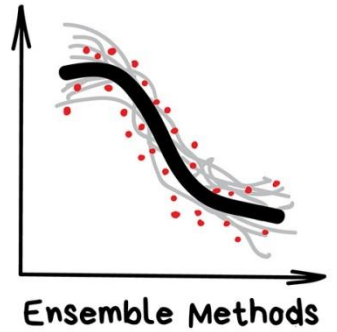
Цей спосіб включає послідовне навчання алгоритмів. Тобто спочатку навчаємо перший і відзначаємо місця, де він помилився. Потім навчаємо другий, особливу увагу приділяючи місцям, де помилявся перший. І так далі. До потрібного результату.



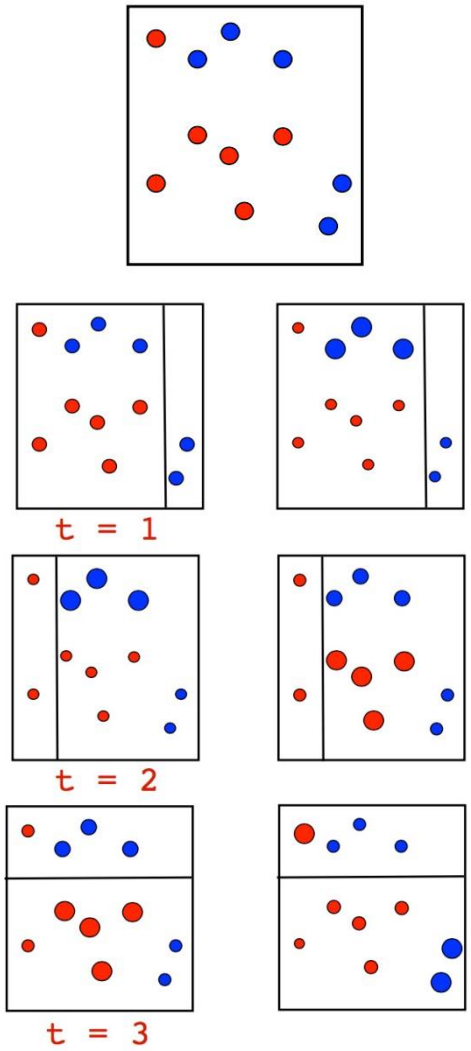
Бустинг

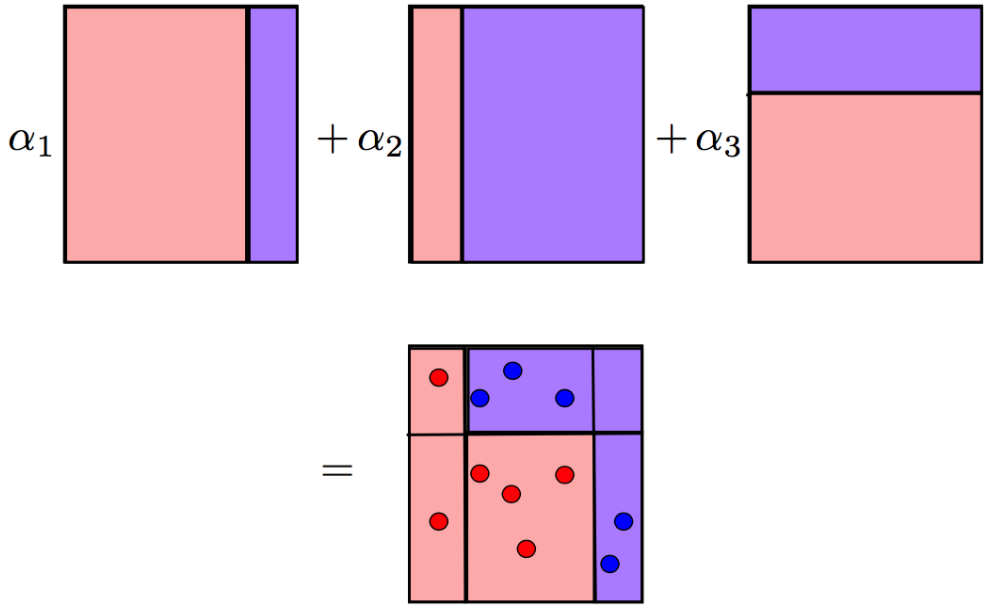
Ансамблеві методи

Бустинг = **Boosting**

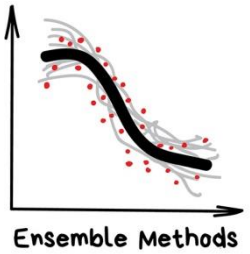


Ансамбль з трьох простих дерев





Бустинг для задачі класифікації с 2-ма класами



Покладемо $Y = \{\pm 1\},\ b_t : X \to \{-1, 0, 1\},\ C(b) = sign(b)$.
$b_t(x) = 0$ – відмова (краще промовчати ніж збрехати).
Зважене голосування:

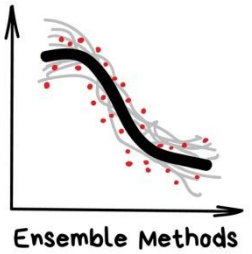
$$a(x) = sign\left(\sum_{t=1}^{T} \alpha_t b_t(x)\right), \qquad x \in X.$$

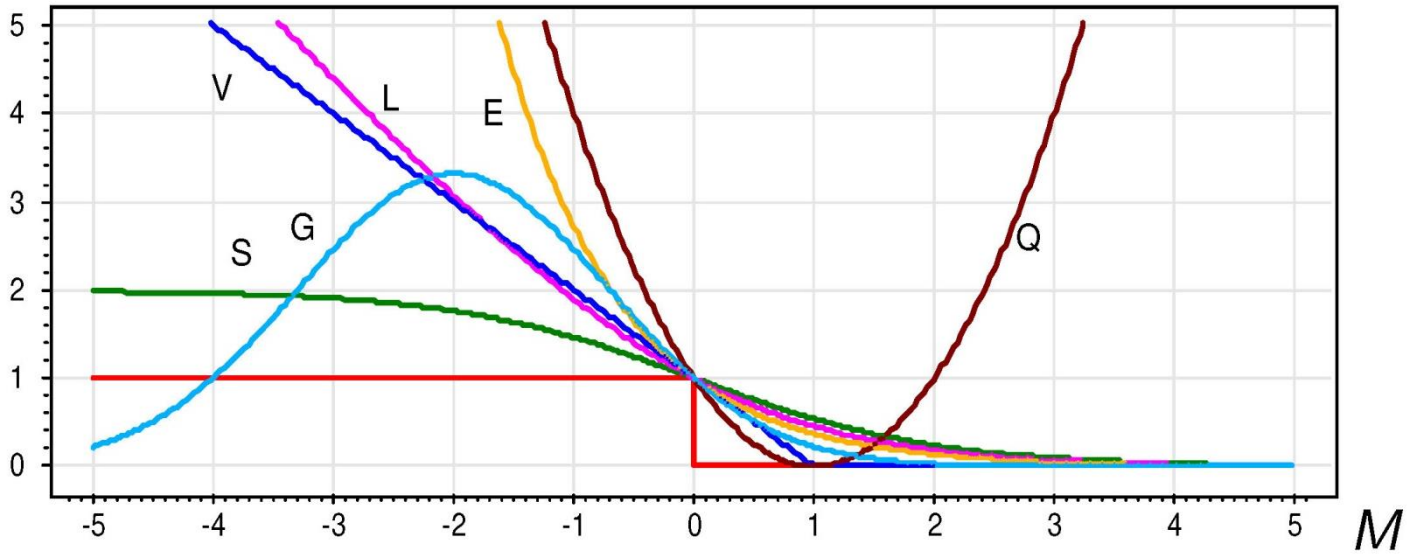
Функціонал якості композиції – кількість помилок на X^ℓ

$$Q_T = \sum_{i=1}^{\ell} \left[y_i \sum_{t=1}^{T} \alpha_t b_t(x_i) < 0 \right].$$

Два основних моменти бустинга:
Фіксація $\alpha_t b_t(x), \ldots, \alpha_{t-1} b_{t-1}(x)$ при додаванні $\alpha_t b_t(x)$
Гладка апроксимація порогової функції втрат $[M \leq 0]$.

Гладкі апроскимації порогові функції втрат





$E(M) = e^{-M}$ — экспоненциальная (AdaBoost);

$L(M) = \log_2(1 + e^{-M})$ — логарифмическая (LogitBoost);

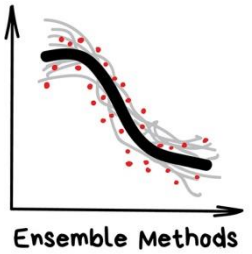
$Q(M) = (1 - M)^2$ — квадратичная (GentleBoost);

$G(M) = \exp\big(-cM(M + s)\big)$ — гауссовская (BrownBoost);

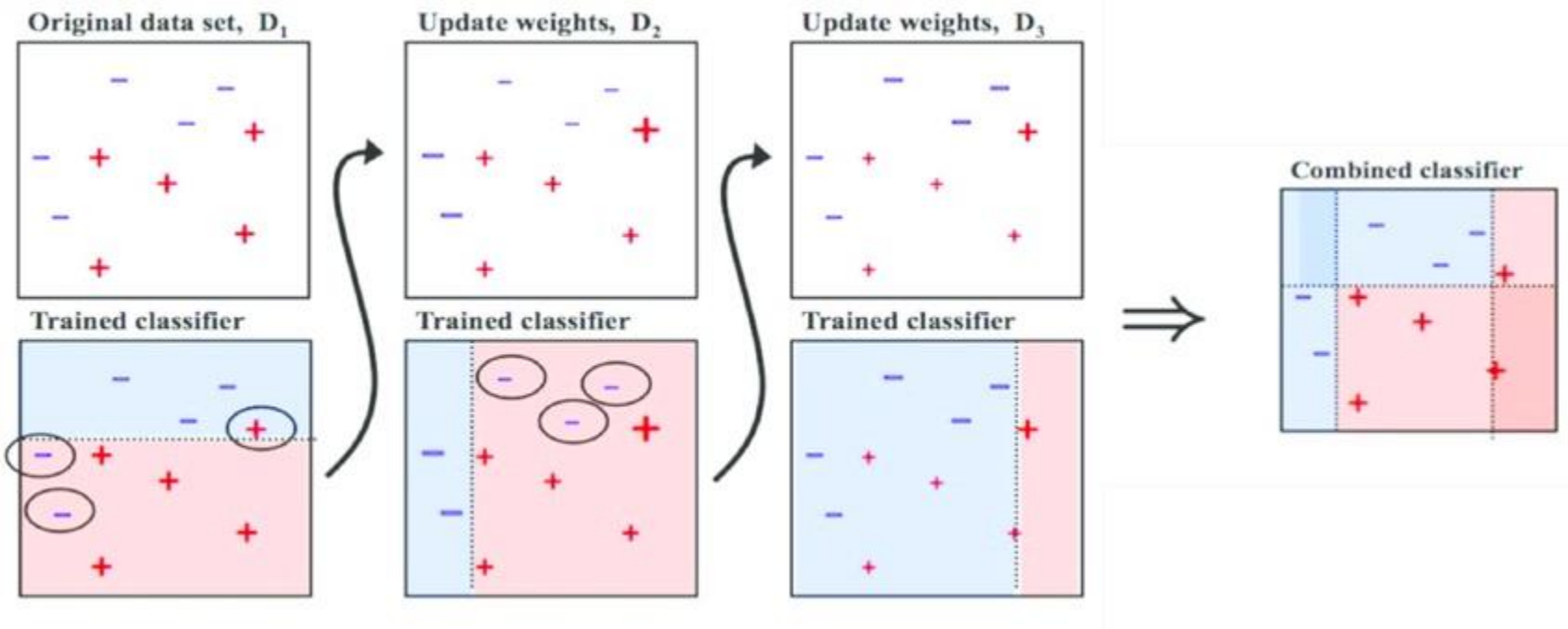
$S(M) = 2(1 + e^{M})^{-1}$ — сигмоидная;

$V(M) = (1 - M)_+$ — кусочно-линейная (из SVM);

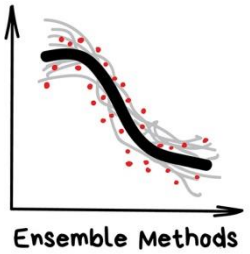
Адаптивний бустинг (AdaBoost)

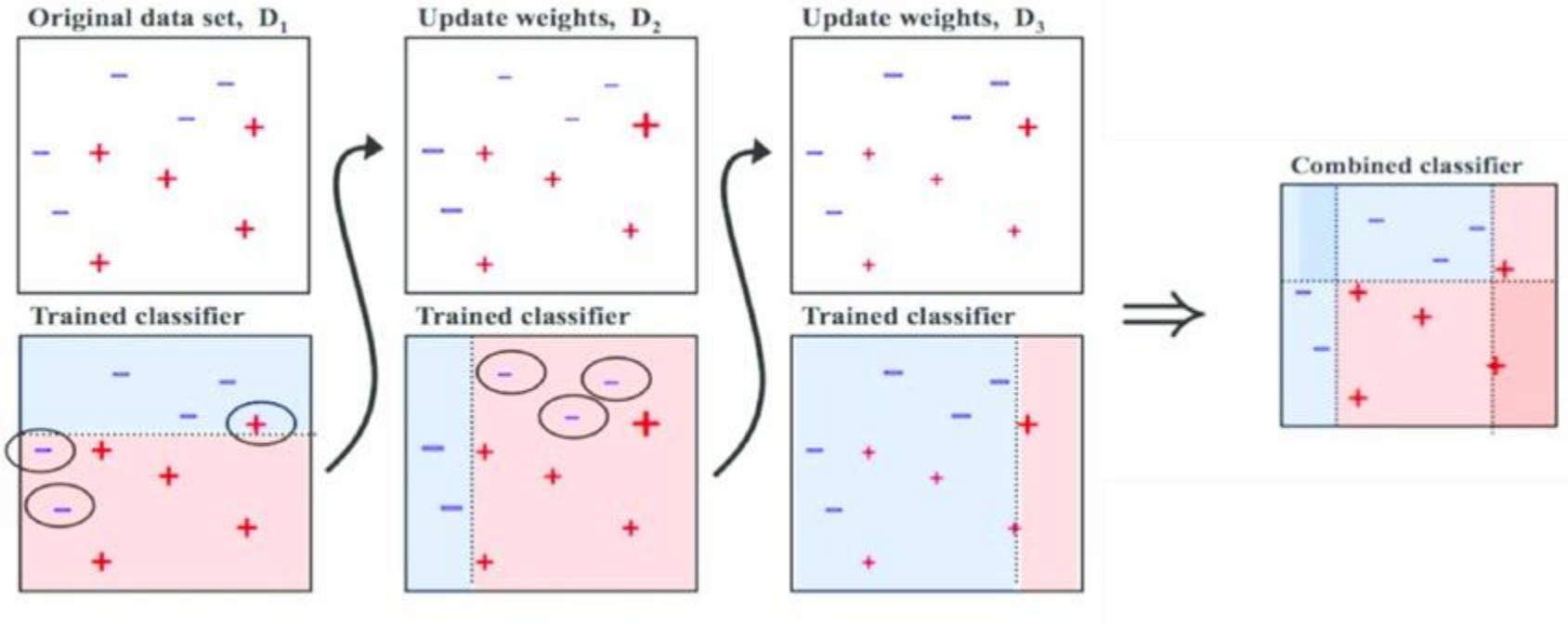


Адаптивний бустинг, відомий як AdaBoost, - це алгоритм Бустингу. Метод, який використовує цей алгоритм для мінімізації помилки попередніх моделей, полягає у концентрації уваги на недонавченості алгоритму. Що означає, при кожному новому пророкуванні алгоритм працюватиме над складними, неочевидними для передбачення підвиборками.



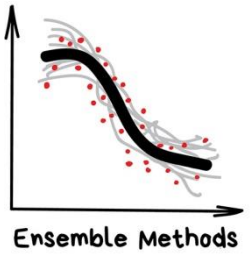
Адаптивний бустинг (AdaBoost)

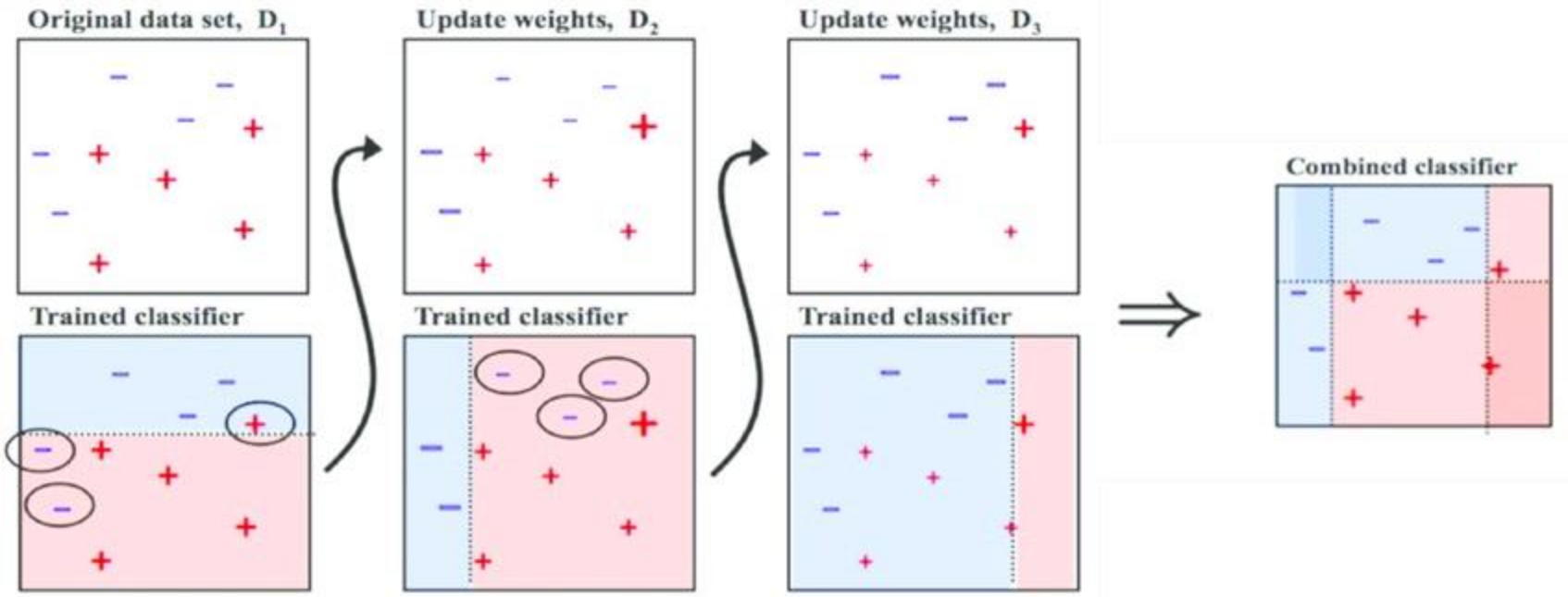




Щоб побудувати класифікатор AdaBoost, як перший базовий алгоритм навчаємо вирішальне дерево, щоб робити передбачення на навчальній вибірці. Тепер, методом адаптивного бустингу, ваги помилково класифікованих елементів зросли. Другий алгоритм навчається вже з урахуванням оновлених коефіцієнтів, і так ця процедура повторюється щоразу.

Адаптивний бустинг (AdaBoost)

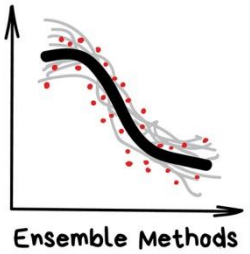




Після передбачень кожної моделі збільшуємо ваги помилково класифікованих об'єктів, щоб наступна модель краще спрацювала на них.

Така техніка послідовного навчання нагадує градієнтний спуск тільки замість зміни параметрів одного предиктора для мінімізації функції втрат AdaBoost додає моделі в ансамбль, поступово покращуючи його. Великим недоліком цього алгоритму можна вважати те, що його не можна розпаралелити, оскільки кожен із предикторов може бути навчений лише після закінчення навчання попереднього.

Адабуст (AdaBoost)
експоненційна функція втрат



Оцінка функціоналу якості Q_T зверху:

$$Q_T \le \widetilde{Q_T} = \sum_{i=1}^{\ell} \underbrace{\exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t b_t(x_i)\right)}_{w_i} \exp(-y_i \alpha_T b_T(x_i))$$

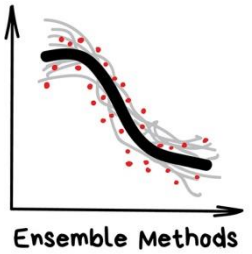
Нормовані ваги $\widetilde{W}^\ell = (\widetilde{w_1}, \dots, \widetilde{w_\ell})$, $\widetilde{w_i} = w_i / \sum_{j=1}^{\ell} w_j$.

Зважена кількість помилок (негативні) та правильних класифікацій (позитивні) при векторі ваг $U^\ell = (u_1, \dots, u_\ell)$:

$$N(b, U^\ell) = \sum_{i=1}^{\ell} u_i [b(x_i) = -y_i]; \quad P(b, U^\ell) = \sum_{i=1}^{\ell} u_i [b(x_i) = y_i]$$

$(1 - P - N)$ – зважена кількість відмов від класифікації

Теорема AdaBoost



Нехай $\mathcal{B}$ – доволі велике сімейство базових алгоритмів

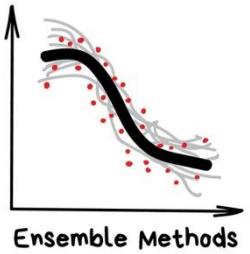
Теорема

Нехай для будь-якого нормованого вектору ваг $U^\ell = (u_1, \dots, u_\ell)$ існує алгоритм $b \in \mathcal{B}$, який класифікує вибірку хоча б трішки краще, ніж навмання: $P(b, U^\ell) > N(b, U^\ell)$. Тоді мінімум функціоналу $\widetilde{Q_T}$ досягається за умови

$$b_T = \arg\max_{b \in \mathcal{B}} \sqrt{P(b, \widetilde{W^\ell})} - \sqrt{N(b, \widetilde{W^\ell})}$$

$$\alpha_T = \frac{1}{2} \ln \frac{P(b_T, ; \widetilde{W^\ell})}{N(b_T; \widetilde{W^\ell})}$$

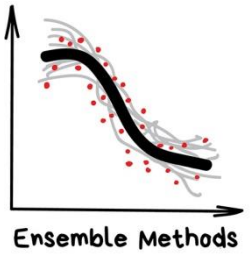
Алгоритм AdaBoost



Кроки реалізації алгоритму AdaBoost:

1. Спочатку всім точкам надано рівні ваги
2. Модель будується на підвиборці даних.
3. За цією моделлю виходять передбачення для всіх даних.
4. За прогнозами та справжніми значеннями обчислюються помилки.
5. У побудові наступної моделі найбільші ваги присвоюються точкам даних, передбачення яких алгоритм помилився.
6. Ваги можуть бути визначені за величиною помилки. А саме, чим більша помилка, тим більша вага.
7. Цей процес повторюється, поки функція помилки не перестане змінюватися або поки не буде досягнуто максимальної кількості предикторів.

Алгоритм AdaBoost



Вхід: навчальна вибірка X^ℓ, параметри базових T алгоритмів
Вихід: базові алгоритми b_t, та їх ваги α_T, $t = 1, \dots, T$

1: ініціювати ваги для всіх об'єктів:

$$w_i = 1/\ell\,,\ \ i = 1, \dots, \ell;$$

2: **для всіх** $t = 1, \dots, T$

3: навчити базовий алгоритм

$$b_T = \arg\min_{\text{b}} N\left(b, \widetilde{W^\ell}\right)$$

4: порахувати ваги алгоритмів

$$\alpha_T = \frac{1}{2}\ln\frac{1-N\left(b_T,;\widetilde{W^\ell}\right)}{N\left(b_T;\widetilde{W^\ell}\right)}$$

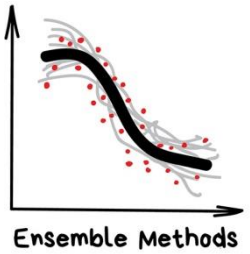
5: оновити ваги об'єктів

$$w_i = w_i\exp(-y_i\alpha_t b_t(x_i)),\ \ i = 1, \dots, \ell$$

6: віднормувати ваги об'єктів:

$$w_i\ = w_i/\textstyle\sum_{j=1}^{\ell} w_j\,,\ \ i = 1, \dots, \ell$$

Рекомендації



Базові класифікатори

Дерева пошуку рішень використовуються найчастіше

Порогові правила "пні" (data stumps)

$$\mathcal{B} = \{b(x) = [f_j(x) \lessgtr \theta] | j = 1, \dots, n, \theta \in \mathbb{R}\}$$

Для SVM бустинг не ефективний

Відсів шуму

Відкинути об'єкти з найбільшими w_i

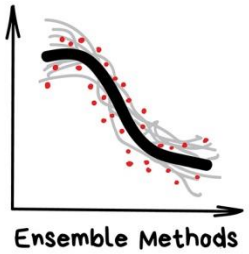
Модифікація формули для α_T на випадок $N = 0$

$$\alpha_T = \frac{1}{2} \ln \frac{1 - N(b_T, ; \widetilde{W}^\ell) + \frac{1}{\ell}}{N(b_T; \widetilde{W}^\ell) + \frac{1}{\ell}}$$

Додатковий критерій зупинки

Збільшення частоти помилок на контрольній вибірці

Бустинг – невипадковий ліс

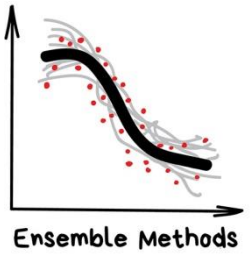


Побудуємо схожий ліс, але набір даних буде невипадковим. Перше дерево ми побудуємо так само, як і раніше, на випадкових даних і випадкових критеріях. А потім проженемо через це дерево контрольну вибірку: інші кліпи, за якими у нас є всі дані, але які не беруть участь у навчанні. Подивимося, де дерево помилилося:

Клип	Есть 1 000 000?	Что ответило дерево
1	Нет	Нет
2	Нет	Да
3	Нет	Нет
4	Да	Да
5	Нет	Да
6	Нет	Нет
7	Да	Нет
8	Нет	Нет
9	Нет	Нет
10	Нет	Да
11	Нет	Да
12	Да	Да
...	...	...
10 000	Нет	Нет

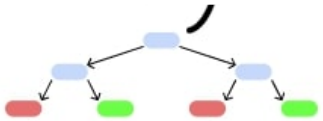
Тепер робимо таке дерево. Звернімо увагу на місця, де перше дерево помилилося. Дамо цим помилкам більшу вагу при підборі даних і критеріїв для навчання. Завдання — зробити дерево, яке виправить помилки попереднього.

Клип	Есть 1 000 000?	Что ответило дерево
1	Нет	Нет
2	Нет	Да
3	Нет	Нет
4	Да	Да
5	Нет	Да
6	Нет	Нет
7	Да	Нет
8	Нет	Нет
9	Нет	Нет
10	Нет	Да
11	Нет	Да
12	Да	Да
...	...	...
10 000	Нет	Нет



Дерево – зверну увагу

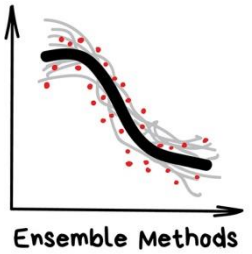
Зрозумів



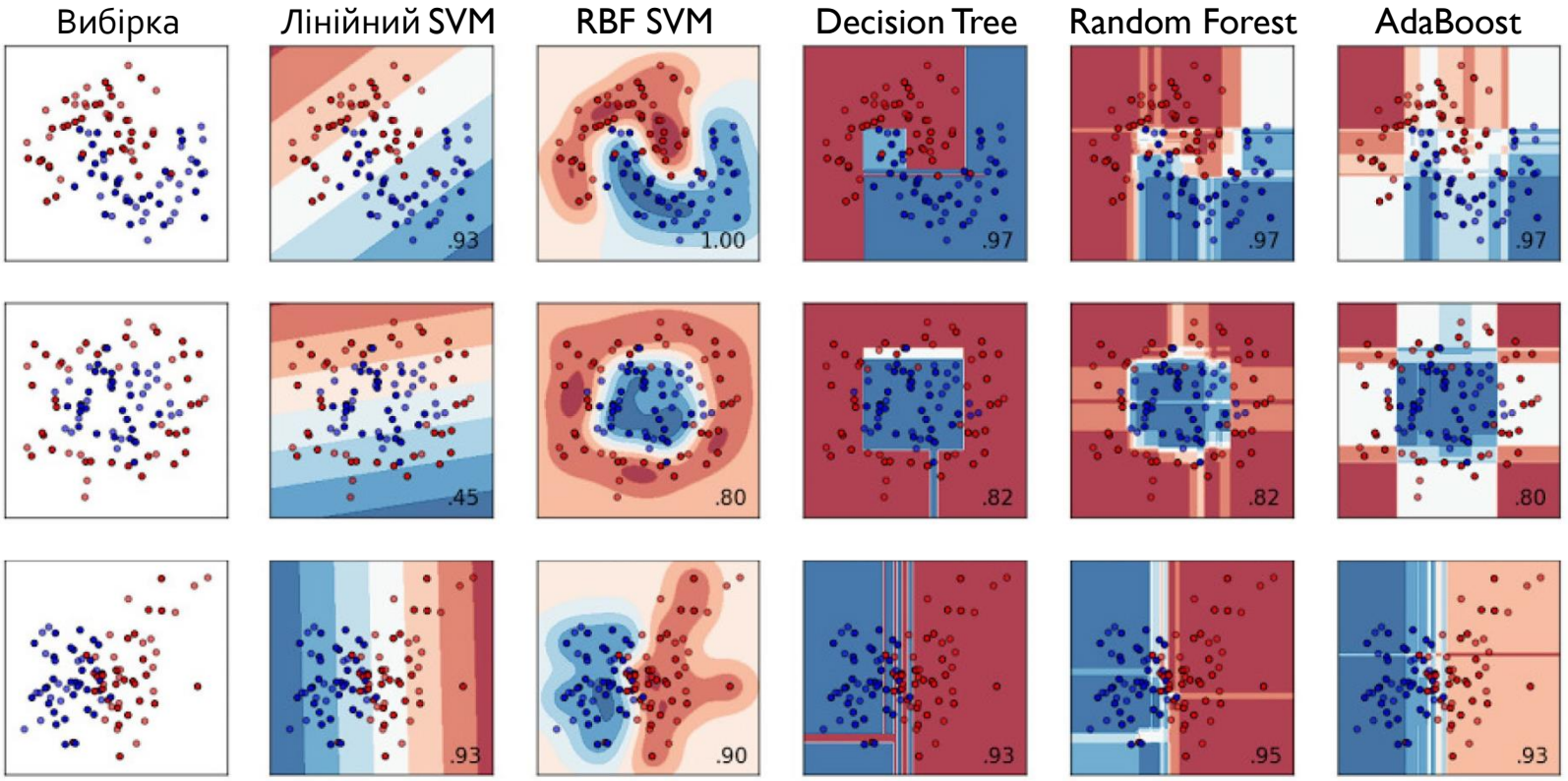
Але друге дерево наробить своїх помилок. Робимо третє, яке їх виправить. Потім четверте. Потім п'яте.

Робимо такі дерева, поки не досягнемо бажаної точності або поки точність не почне падати через перенавчання. Виходить, у нас багато дерев, кожне з яких не дуже сильне. Але разом вони складаються в ліс, який дає хорошу точність.

SVM, Random forest, AdaBoost

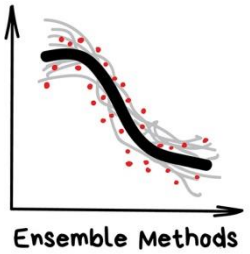


Експерименти на трьох дво-вимірних модельних вибірках

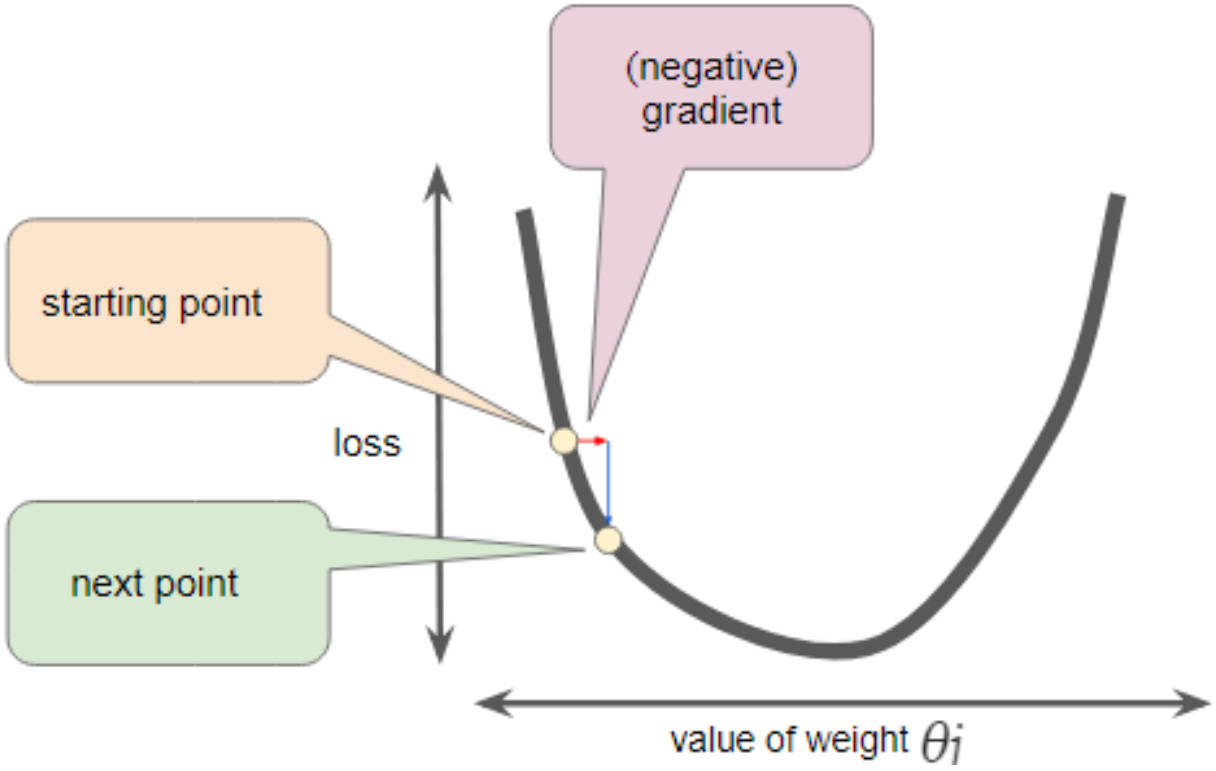


Розв'язки можуть виглядати дивно, проте випадковий ліс (random forest) та бустинг – одні з найпопулярніших методів в машинному навчанні

Градієнтний Бустінг



Градієнтний Бустинг відрізняється від Адаптивного тим, що, на відміну від **AdaBoost**, що змінює ваги при кожній ітерації, Градієнтний намагається навчати нові моделі за помилкою минулих (рухаючись до мінімуму функції втрат).



Дякую за увагу