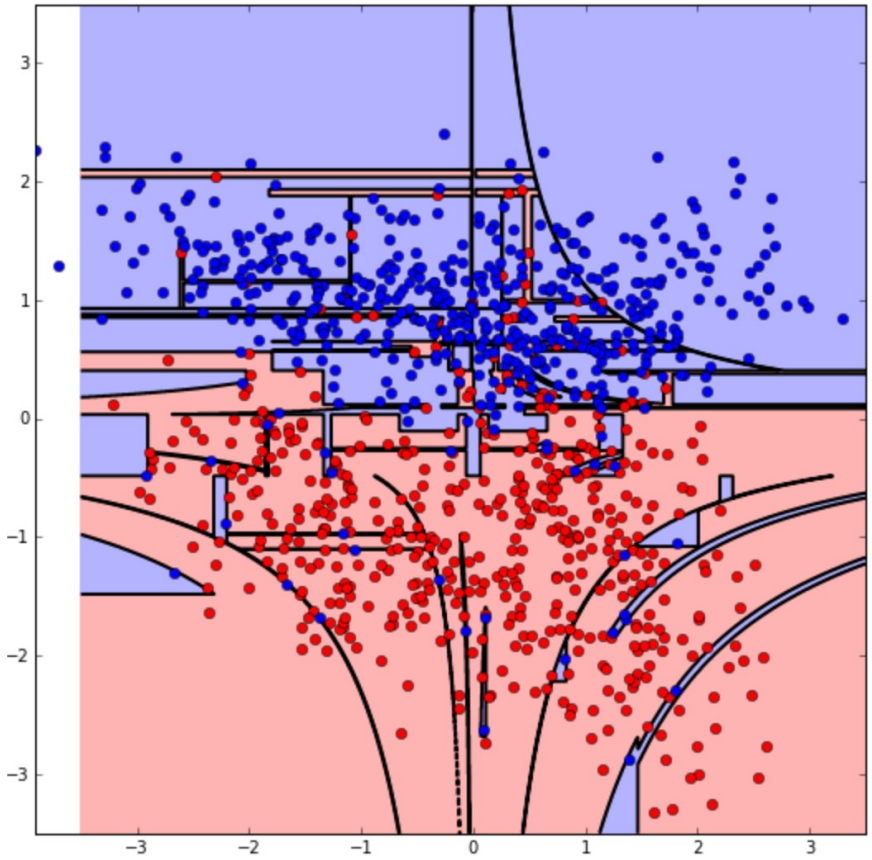
Лекція 13
Композиції дерев (ліси)

§61 Композиції дерев

- У попередньому занятті вивчалися дерева рішень й було встановлено, що вони **здатні відновлювати дуже складні закономірності**, але схильні до перенавчання. Інакше кажучи, **дерева занадто легко підганяються під навчальну вибірку й стають непридатними для побудови прогнозів**.
- Але виявляється, що **дерева рішень дуже добре підходять для об'єднання в композиції** й побудови одного неперенавченого алгоритму на основі великої кількості дерев рішень.

Основні недоліки дерев рішень

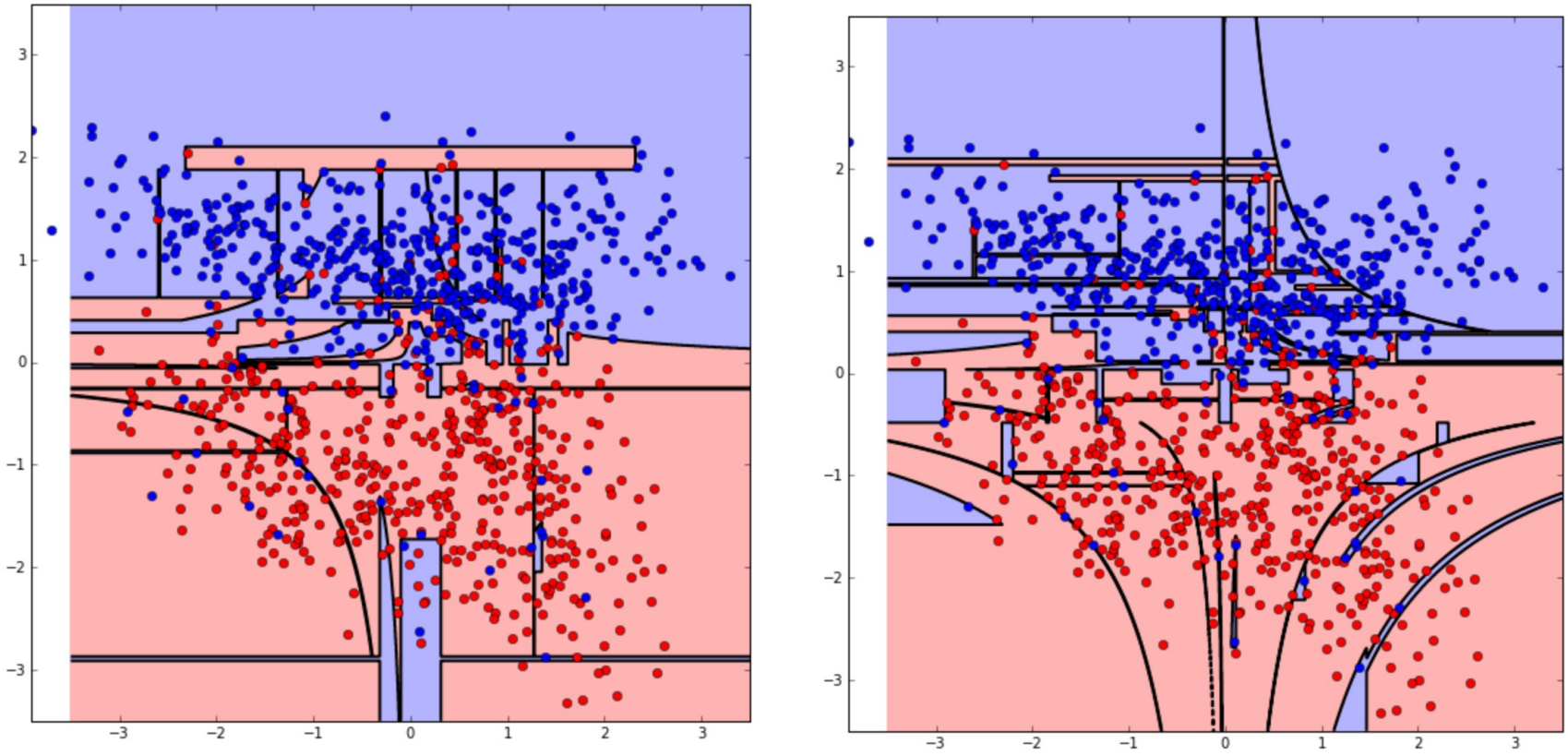
- Взяли складну вибірку й навчити на ній дерево рішень до кінця:



Навіть якщо якийсь об'єкт попадає в «гущавину іншого класу», роздільна поверхня намагається «уловити» його й видати на ньому правильну відповідь.

- Незначно змінили навчальну вибірку, **викинули пару об'єктів**, то навчене на цій вибірці дерево зовсім інше:

Роздільна поверхня вкрай нестійка до зміни вибірки



Дерево рішень має наступні серйозні недоліки:

- сильно перенавчається
- сильно змінюється при невеликій зміні вибірки

Композиція алгоритмів

Композиція — це об'єднання *N* алгоритмів $b_1(x), ..., b_N(x)$ в один.

Ідея:

- навчити алгоритми $b_1(x), ..., b_N(x)$,
- усереднити отримані від них відповіді:

$$a(x) = \text{sign} \frac{1}{N} \sum_{n=1}^{N} b_n(x),$$

Алгоритм ***a(x)*** **–** називається **композицією *N* алгоритмів** $b_1(x), ..., b_N(x)$

$b_1(x), ..., b_N(x)$ – називаються **базовими алгоритмами**.

Приклад: при вирішенні задачі класифікації із двома класами використовувалися 6 різних базових алгоритмів, які на деякому об'єкті *x* видали наступні відповіді:

$$-1, -1, 1, -1, 1, -1.$$

Відповідь композиції:

$$a(x) = \operatorname{sign}\left(-\frac{2}{6}\right) = -1.$$

Тобто **об'єкт був віднесений до класу –1**, тому що за цей варіант «проголосувало» більшість базових алгоритмів.

Рандомізація

- Як знайти різні базові алгоритми, тобто **як зробити усі дерева різними**?
- Використати рандомізацію, тобто навчати базові алгоритми **на різних підвибірках навчальної вибірки**

Оскільки дерева рішень сильно змінюються навіть від невеликих змін навчальної вибірки, така рандомізація значно підвищує різницю базових алгоритмів.

Бутстреп — один з популярних підходів до побудови підвибірок.

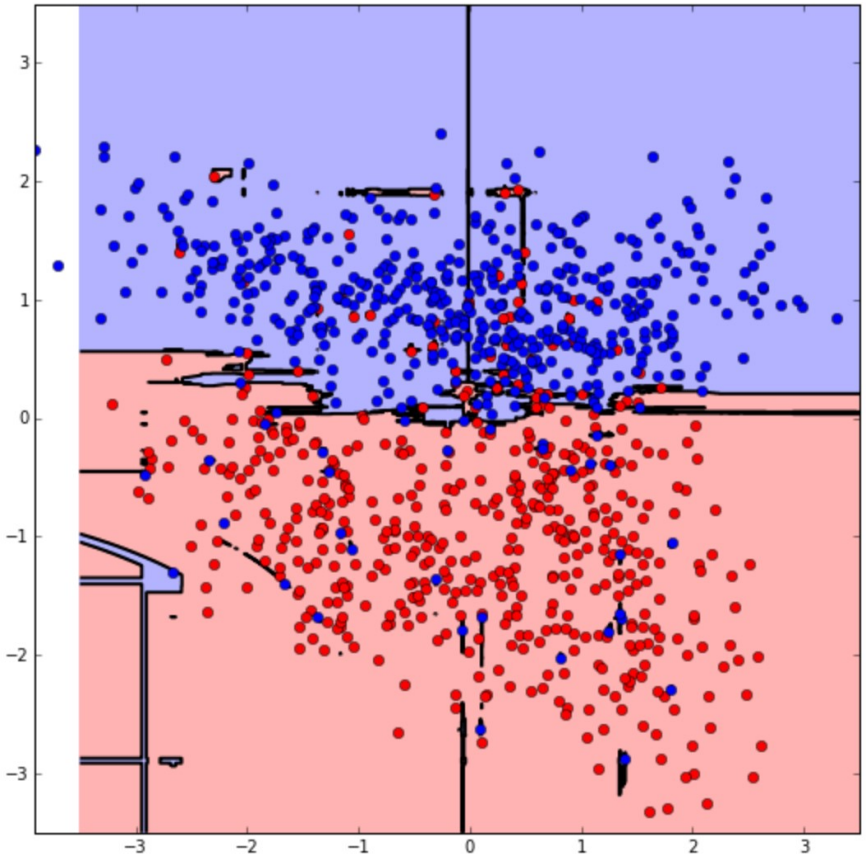
Статистичний бутстреп (бутстреп, бутстреппінг, англ. bootstrap, bootstrapping) — практичний комп'ютерний метод визначення статистик імовірнісних розподілів, заснований на багаторазовій генерації виборок методом Монте-Карло на базі наявної вибірки (Вікіпедія)

Сутність: З навчальної вибірки довжини *A* **вибирають із поверненням** *A* об'єктів. При цьому нова вибірка також буде мати розмір *A*, але **деякі об'єкти в ній будуть повторяться**, а деякі об'єкти з вихідної вибірки в неї не потраплять. Можна показати, що в бутстрепованій вибірці буде знаходитись в **середньому 63% різних об'єктів вихідної вибірки**.

- Інший підхід до рандомізації - **генерація випадкової підмножини навчальної вибірки**.
- Розмір цієї випадкової **підмножини є гіперпараметром**. Наприклад, можна випадково взяти половину вихідної вибірки й навчити на ній базовий алгоритм. Цей підхід трохи програє бутстрепу, тому що містить гіперпараметр, у той час як бутстреп без якого-небудь налаштування видає підвибірку.

Композиція дерев

Якщо за **допомогою бутстрепа побудувати 100 базових дерев рішень і об'єднати їх у композицію**, то отримаємо:



В цілому добре розділяє два класи. Збільшенням кількості базових алгоритмів можна усунути похибки, що залишилися.

§62 Зсув і розкид

Чому усереднення алгоритмів дозволяє підвищити якість?

Розкладання похибки: шум, зсув і розкид

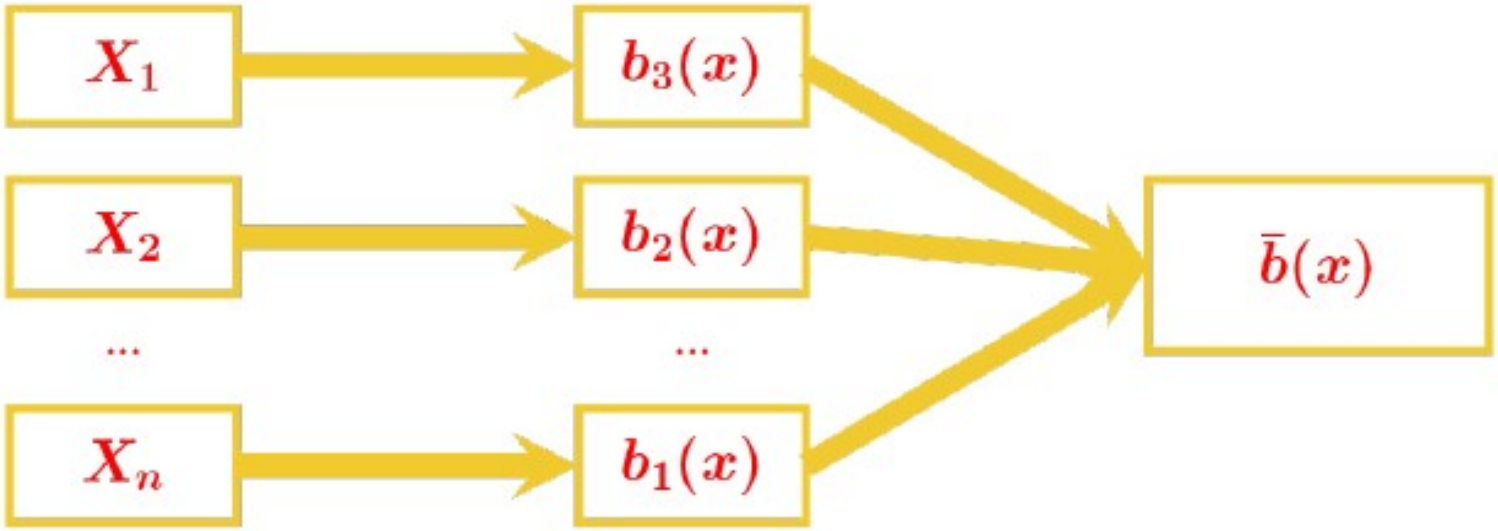
Похибка алгоритму на нових тестових даних складається із трьох компонентів: **шуму, зсуву й розкиду**.

- **Шум** — компонент похибки алгоритму, що буде проявлятися навіть на ідеальній моделі в цій задачі. Інакше кажучи, **шум є характеристикою даних** і буде проявлятися, яка б модель не використовувалася.

Розглядаємо модель

- Генеруємо багато навчальних вибірок
- На кожній навчаємо один з базових алгоритмів

- **Зсув — середнє відхилення** прогнозу заданої моделі від прогнозу ідеальної моделі, яке усереднено за різними навчальними вибірками.



- **Розкид — дисперсія відповідей моделей**, навчених по різних навчальних вибірках. Розкид характеризує те, наскільки сильно прогноз алгоритму залежить від конкретної навчальної вибірки.

Приклад:

Задача регресії: потрібно апроксимувати істину залежність (зображена на правому графіку зеленим) поліномом третього порядку за навчальною вибіркою. Навчальна вибірка це собою 10 випадкових точок істиною залежності, до яких був доданий випадковий шум.

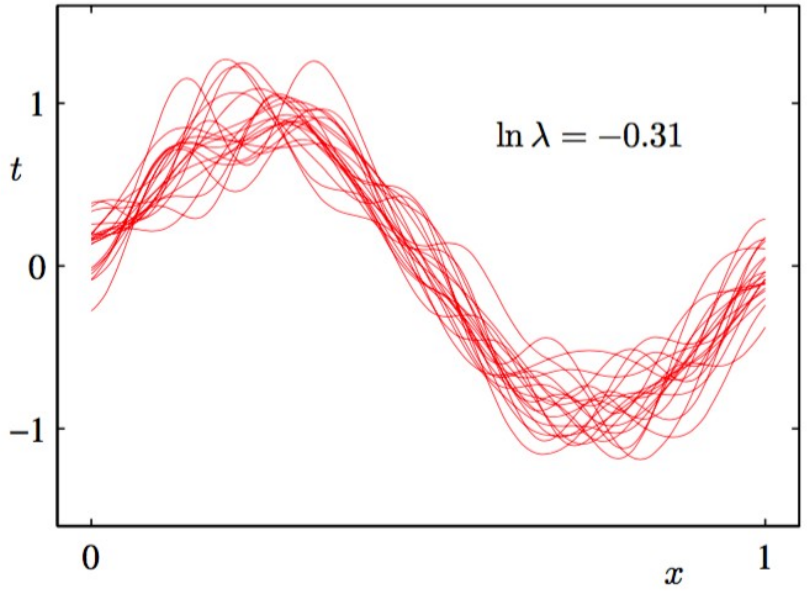


Рис. 62.1 Результат роботи кожного базового алгоритму

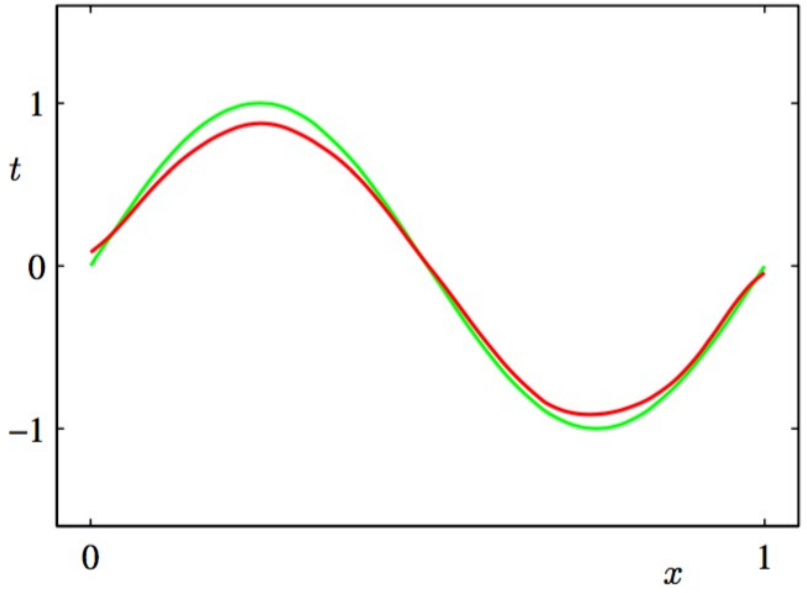


Рис. 62.2 Істина залежність – зелена лінія, композиція алгоритмів – червона лінія

Лінійні моделі:

- У випадку нелінійних задач, яких переважна більшість, **зсувів** при використанні таких алгоритмів буде **великими**.
- **Розкид**, навпаки, буде **маленьким** через мале число параметрів, порівнянного з кількістю ознак.

Дерева рішень:

- Характеризуються **низьким зсувом**, тобто здатні відновлювати складні закономірності,
- **Мають великий розкид**: дерева рішень сильно змінюються навіть при невеликих змінах навчальної вибірки.

Зсув і розкид композиції алгоритмів

Розглядаємо композицію базових алгоритмів, зсув композиції **збігається зі зсувом окремого базового алгоритму**.

Дерева характеризуються **низьким зсувом**, то це ж буде вірно й для **композиції дерев**.

Розкид композиції вже відрізняється від розкиду одного базового алгоритму:

$$\begin{pmatrix}\text{розкид}\\ \text{композицій}\end{pmatrix}=\frac{1}{N}\begin{pmatrix}\text{розкид одного}\\ \text{базового алгоритму}\end{pmatrix}+\begin{pmatrix}\text{кореляція між}\\ \text{базовими алгоритмами}\end{pmatrix}$$

Якщо базові алгоритми незалежні, тобто їхні прогнози не корелюють між собою, вираз спрощується:

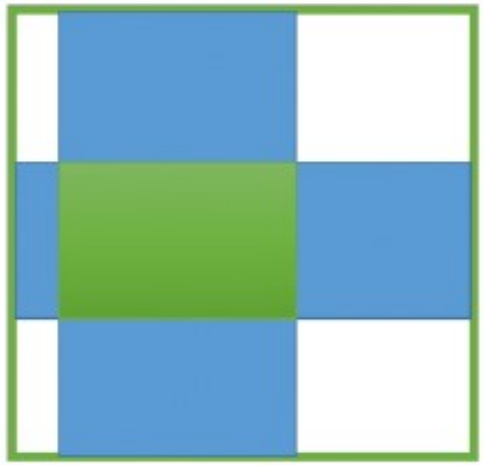
$$\begin{pmatrix}\text{розкид}\\ \text{композицій}\end{pmatrix}=\frac{1}{N}\begin{pmatrix}\text{розкид одного}\\ \text{базового алгоритму}\end{pmatrix}.$$

Фактично, композиція достатньої кількості некорельованих алгоритмів може дати ідеальний алгоритм.

Зменшення кореляції базових алгоритмів

Існують наступні два підходи зі зменшення кореляції базових алгоритмів:

1. **Бегінг**: Навчання базових алгоритмів відбувається на випадкових підвибірках навчальної вибірки. Причому чим менше розмір випадкової підвибірки, тим більш незалежними є базові алгоритми.
2. **Метод випадкових підпросторів**: вибирається випадкова підмножина ознак (стовпців матриці «об'єкти-ознаки») і черговий базовий алгоритм навчається тільки на цих ознаках. Частка обраних ознак є гіперпараметром цього методу.



Два даних підходи - бегінг і метод випадкових підпросторів - можна поєднувати й використовувати одночасно.

§63 Випадкові ліси

У цьому підрозділі мова йтиме про випадкові ліси, які є одним із кращих способів об'єднання дерев у композиції.

Випадковий ліс

Раніше були отримані наступні результати:

- Похибка може бути розкладена на шум, зсув і розкид.
- Зсув композиції близький до зсуву одного базового алгоритму.
- Розкид при побудові композиції зменшується, причому тем сильніше, ніж менш корельовані базові алгоритми.

Розглянутих минулого разу **способів зниження кореляції** між базовими алгоритмами (бегінг і метод випадкових підпросторів) **виявляється недостатньо.** Щоб базові алгоритми були ще менш скорельовані, має сенс зробити **випадковим їхній процес побудови**.

Рандомізація процесу побудови дерев рішень

Процес побудови дерев рішень є жадібним алгоритмом, що працює до виконання критерію зупинки.

Нехай на деякому кроці алгоритму необхідно розбити вершину *m*, у якій є вибірка X_m, на дві. Як умова розбивки використовується порівняння *j-го* ознаки з порогом *t*:

$$[x^j \leq t].$$

Параметри *j* і *t* вибираються виходячи з умови мінімізації функції похибки $Q(X_m, j, t)$:

$$Q(X_m, j, t) \to \min_{j,t}.$$

Рандомізувати процес побудови можна, якщо в задачі пошуку оптимальних параметрів **вибирати *j* з випадкової підмножини ознак розміру *q*.** Виявляється, що цей підхід дійсно дозволяє зробити дерева менш корельованими.

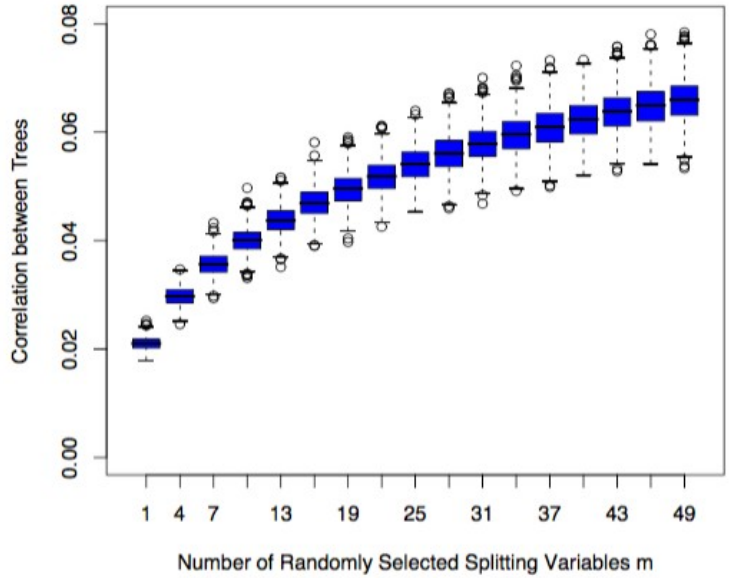


Рис. 8.1: Залежність кореляції між деревами від параметра *q*

За графіком видно, що **чим менше «простір для вибору кращої розбивки»**, тобто чим менше *q*, **тим менше кореляції** між деревами рішень.

Для *q* є деякі рекомендації, які непогано працюють на практиці:

- У задачі регресії має сенс брати $q = d/3$, тобто використовувати третину від загального числа ознак.
- У задачі класифікації має сенс брати $q = \sqrt{d}$.

Алгоритм побудови випадкового лісу

Щоб побудувати випадковий ліс із *N* дерев рішень, необхідно:

1. Будуємо за допомогою бутстрепа *N* випадкових підвибірок $\tilde{X}_n$.
2. Кожна підвибірка $\tilde{X}_n$ використовується як навчальна вибірка для побудови відповідного дерева рішень $b_n(x)$. Причому:

- Дерево будується, поки в кожному листочку виявиться не більше $n_{\min}$ об'єктів. Дуже часто дерева будують до кінця ($n_{min} = 1$), щоб одержати складні й перенавчені дерева рішень з низьким зсувом.
- Процес побудови дерева рандомізоровано: на етапі вибору оптимальної ознаки, за яким буде відбуватися розбивка, вона шукається не серед усієї множини ознак, а серед випадкової підмножини розміру *q*.
- Випадкова підмножина розміру *q* вибирається заново щораз, коли необхідно розбити чергову вершину.

3. Побудовані дерева поєднуються в композицію:

- В задачах регресії $a(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x);$
- В задачах класифікації $a(x) = \operatorname{sign}\frac{1}{N}\sum_{n=1}^{N} b_n(x).$

Особливість випадкових лісів: **вони не перенавчаються за умови збільшення числа базових алгоритмів**.

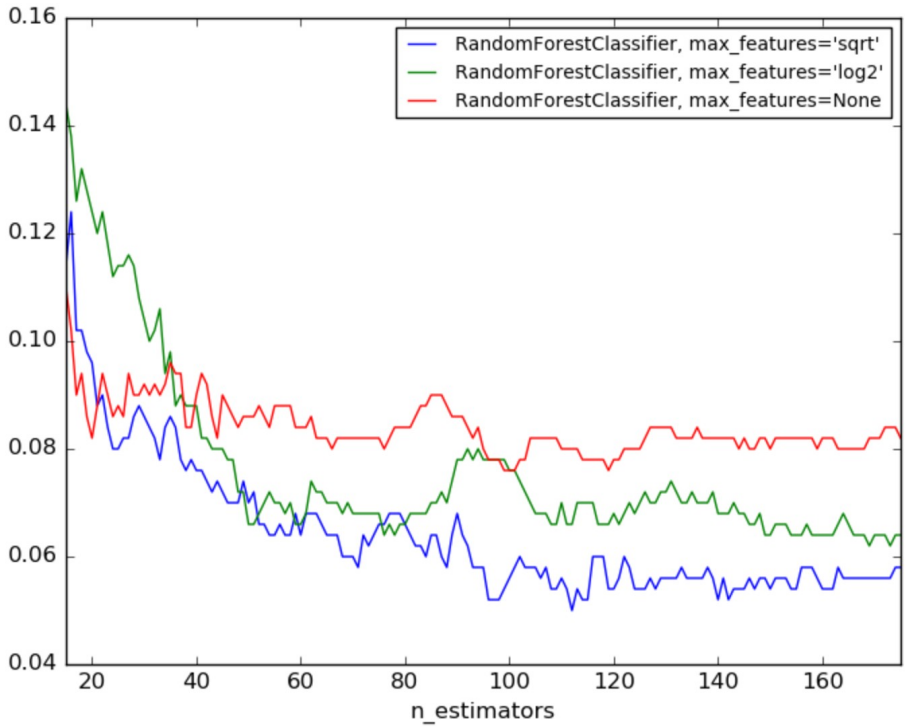


Рис. 8.2: Залежність похибки випадкового лісу від кількості базових алгоритмів

За графіком видно, що похибка на тесті спочатку зменшується з ростом числа базових алгоритмів, а потому виходить на асимптоту.

§64 Трюки з випадковими лісами

Можливість розпаралелювання

- Оскільки кожне дерево навчається незалежно від всіх інших базових дерев рішень, його можна **навчати на окремому ядрі або окремому комп'ютері**. Ідеальне розпаралелювання: швидкість обчислень пропорційна кількості задіяних обчислювальних ядер.

Оцінювання якості випадкового лісу

- Кожне дерево з випадкового лісу навчається на **бутстрепованій вибірці**, у яку попадають приблизно **63% о**б'єктів повної вибірки. Таким чином, близько **37% о**б'єктів вибірки не використовувалися при навчанні цього дерева, а значить їх можна використовувати для оцінки узагальнюючої здатності випадкового лісу.

Такий підхід зветься **out-of-bag** і дозволяє оцінювати якість лісу **без використання відкладеної вибірки або крос-валідації.** Формула для оцінки якості випадкового лісу з *N* дерев у рамках підходу out-of-bag має вигляд:

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

Доля деревьев, которые не обучались на x_i

Для кожного об'єкта x_i з навчальної вибірки обчислюється **середній прогноз по деревам, у навчальну вибірку яких не входить об'єкт** x_i:

$$\frac{1}{\sum_{n=1}^{N}[x_i \notin X_n]} \sum_{n=1}^{N}[x_i \notin X_n] b_n(x_i).$$

Для отриманого прогнозу обчислюється значення похибки. Як оцінка якості випадкового лісу використовується сума таких значень для всіх елементів вибірки.

§65 Випадкові ліси в sklearn

Див. JNotebook «4_2_sklearn_random_forest.ipynb»

§66 Композиції простих алгоритмів

Обговоримо, чому випадкові ліси підходять не для всіх задач.

Недоліки випадкового лісу

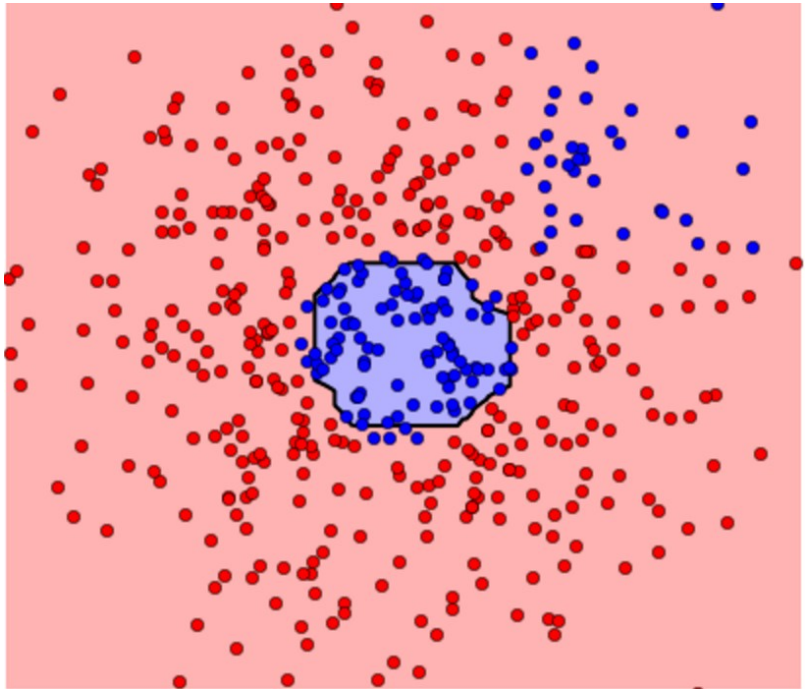
Випадковий ліс:

- композиція глибоких дерев,
- вони будуються незалежно один від одного.

Є проблема:

- **1.** Дерева повинні бути глибокими, тому такий підхід **вимагає дуже багато обчислювальних ресурсів**

Якщо обмежити глибину дерев рішень у випадковому лісі?



Неглибокі дерева не здатні вловлювати всі закономірності в даних. У цьому випадку синій клас складається із двох груп об'єктів, але неглибоке дерево змогло вловити тільки центральну групу. На об'єктах із другої групи таке дерево помиляється.

Є проблема:

- Процес побудови дерев є ненаправленим: кожне наступне дерево в композиції ніяк не залежить від попередніх.
- Через це для рішення складних задач необхідно величезна кількість дерев.

Бустінг: основна ідея

Бустінг - це підхід до побудови композицій, у рамках якого:

- Базові алгоритми будуються послідовно, один за одним.
- Кожний наступний алгоритм будується так, щоб виправляти похибки вже побудованої композиції.
- Можна використовувати прості базові алгоритми, наприклад неглибокі дерева.

Бустінг на прикладі задачі регресії

Розглянемо задачу регресії,

Використовуємо середньоквадратичну похибку:

$$MSE(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell}(a(x_i) - y_i)^2 .$$

Для початку необхідно навчити перший простий алгоритм (наприклад, неглибоке дерево рішень, градієнтний спуск):

$$b_1(x) = \text{argmin}_b \frac{1}{\ell}\sum_{i=1}^{\ell}(b(x_i) - y_i)^2 .$$

Відповіді	y_1	y_2	...	y_ℓ
Прогнози	$b_1(x_1)$	$b_1(x_2)$	...	$b_1(x_\ell)$

Другий алгоритм повинен бути навчений таким чином, щоб **композиція першого й другого алгоритмів**:

$$b_1(x_i) + b_2(x_i) = y_i$$

мала найменшу з можливих похибку на навчальній вибірці.

Ответы	y_1	y_2	...	y_ℓ
Прогнозы	$b_1(x_1)$	$b_1(x_2)$	...	$b_1(x_\ell)$
Поправка	$y_1 - b_1(x_1)$	$y_2 - b_1(x_2)$	...	$y_\ell - b_1(x_\ell)$

Алгоритм $b_2(x)$ налаштовується наступним чином

$$b_2(x) = \operatorname*{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} \Big(b(x_i) - (y_i - b_1(x_i)) \Big)^2$$

Продовжуючи за аналогією на *N* кроці черговий алгоритм $b_N(x)$) буде визначатися в так:

$$b_N(x) = \text{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} \left(b(x_i) - \left(y_i - \sum_{n=1}^{N-1} b_n(x_i) \right) \right)^2 .$$

Процес триває доти, поки похибка композиції не буде нас влаштовувати.

§67 Градієнтний бустінг

Градієнтний бустінг є одним із кращих способів напрямленої побудови композиції на сьогоднішній день. **Як він працює?**

В градієнтному бустінгу створюється композиція

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

як **є сумою**, а не усередненням базових алгоритмів $b_n(x)$. Це пов'язано з тим, що алгоритми навчаються послідовно й кожний наступний коректує похибки попередніх.

Є функція втрат

$$L(y,z)$$

середньоквадратична похибка (у задачі регресії):

$$L(y,z) = (y-z)^2$$

логістична функція втрат (у задачі класифікації):

$$L(y,z) = \log(1+\exp(-yz)).$$

Ініціалізація

Побудуємо перший базовий алгоритм $b_0(x)$ (простий алгоритм). Наприклад:

$$b_0(x) = 0$$

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

$$b_0(x) = \underset{y \in \mathbb{Y}}{\operatorname{argmax}} \sum_{i=1}^{\ell} [y_i = y]$$

Навчання базових алгоритмів

Навчання базових алгоритмів відбувається послідовно. Нехай до деякого моменту навчені $N - 1$ алгоритмів:

$$a_{N-1}(x) = \sum_{n=1}^{N-1} b_n(x).$$

Для наступного алгоритму $b_N(x)$ навчаємо так, щоб зменшити похибку композиції на навчальній вибірці:

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + b(x_i)\right) \to \min_b.$$

Як це зробити?

Спочатку вирішимо більше просту задачу: ***які значення*** $s_1,...,s_l$ повинен приймати алгоритм $b_N(x_i) = s_i$ на об'єктах навчальної вибірки, щоб похибка на навчальній вибірці була мінімальною:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s_1,...,s_\ell}$$

($s_1,...,s_l$ – вектор зміщень)

Маємо задачу мінімізації:

$$F(s) = \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s}$$

Шукаємо вектор зміщень

Необхідно знайти такий вектор зміщень *s*, що буде мінімізувати функцію *F*(*s*). Оскільки напрямок найшвидшого зменшення функції задається напрямком **антиградієнта**, його можна прийняти як **вектор *s***:

$$s = -\nabla F = \Big(-L'_z\big(y_1, a_{N-1}(x_1)\big), \ldots, -L'_z\big(y_\ell, a_{N-1}(x_\ell)\big) \Big)$$

Сдвиг по первому объекту

$$s = -\nabla F = \Big(-L'_z\big(y_1, a_{N-1}(x_1)\big), \ldots, -L'_z\big(y_\ell, a_{N-1}(x_\ell)\big) \Big)$$

Сдвиг по ℓ-му объекту

Компоненти вектора зміщень s, фактично, є тими значеннями, які на об'єктах навчальної вибірки повинен приймати новий алгоритм $b_N(x)$.

Навчання $b_N(x)$ є задачею навчання на розмічених даних, у якій $\{(x_i, s_i)\}_{i=1}^{l}$

Це є стандартною задачею машинного навчання

Використаємо квадратичну функція похибки:

$$b_N(x) = \underset{b}{\operatorname{argmin}} \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2$$

Вся інформація про вихідну функцію втрат $L(y, z)$ міститься у виразі для вектора оптимального зміщення s.

Для більшості задач при навчанні $b_N(x)$ можна використовувати квадратичну функцію втрат.

§68 Опис алгоритму градієнтного бустінга

1 **Ініціалізація:** побудова простого алгоритму $b_0(x)$.

2 **Крок ітерації**:

- **Обчислюється вектор зміщення**

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{n-1}(x_1)), \\ \dots \\ -L'_z(y_\ell, a_{n-1}(x_\ell)) \end{pmatrix}.$$

- **Будується алгоритм**

$$b_n(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2,$$

- Алгоритм $b_n(x)$ **додається в композицію**

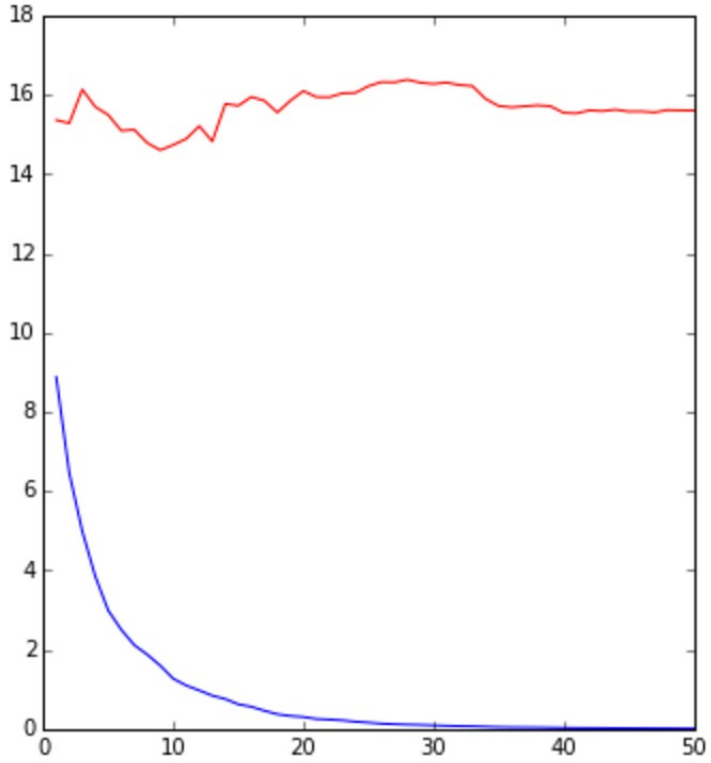
$$a_n(x) = \sum_{m=1}^{n} b_m(x)$$

3 Якщо критерій зупинки виконаний, **зупинити ітераційний процес**.

§65 Проблема перенавчання градієнтного бустінга

Проблема перенавчання градієнтного бустінга

Розглнемо **залежність похибки градієнтного бустінга від числа використаних дерев** на навчальній і контрольній вибірках.



Похибка залежно від числа дерев: **синя лінія - похибка на навчальній вибірці. Червона лінія - похибка на контрольній вибірці**.

Чому так?

В градієнтному бустінгу **використовуються дуже прості базові алгоритми**, наприклад невисокі дерева рішень, які не можуть добре апроксимувати вектор антиградієнта на навчальній вибірці.

Вектор, побудований алгоритмом, не буде вказувати напрямок найшвидшого зменшення похибки, тобто **замість градієнтного спуска можна одержати випадкове блукання**. Через це й отримуємо не дуже гарну якість на контролі.

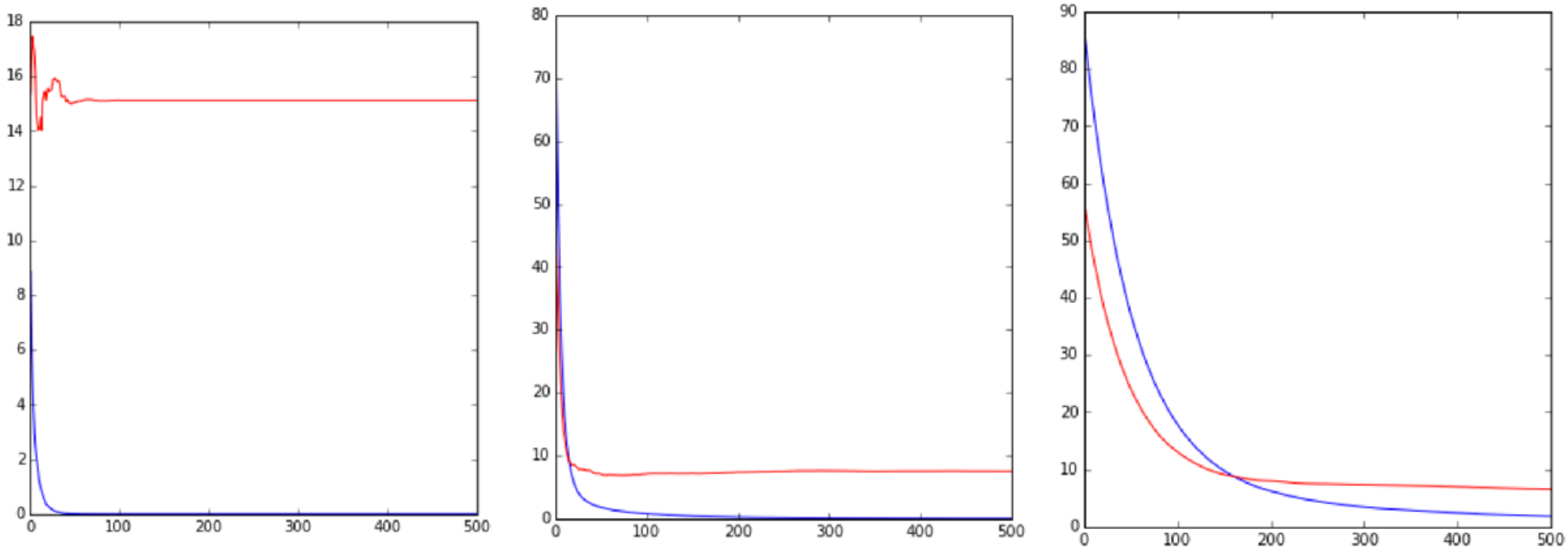
Скорочення розміру кроку

Не будемо «не довіряти» напрямку, який побудував базовий алгоритм і лише мало-мало зміщуватись убік цього вектора:

$$a_N(x) = a_{N-1}(x) + \eta b_N(x)$$

де $\eta \in (0,1]$ — довжина кроку.

Розглянемо, як впливає довжина кроку на якість градієтного бустінгу?



(a) Випадок $\eta = 1$. (b) Випадок $\eta = 0.1$. (c) Випадок $\eta = 0.01$.

Якість градієнтного бустінга на навчальній вибірці й контролі при різних η.

Підбор гіперпараметрів

Чим менше крок, тим більше потрібно базових алгоритмів

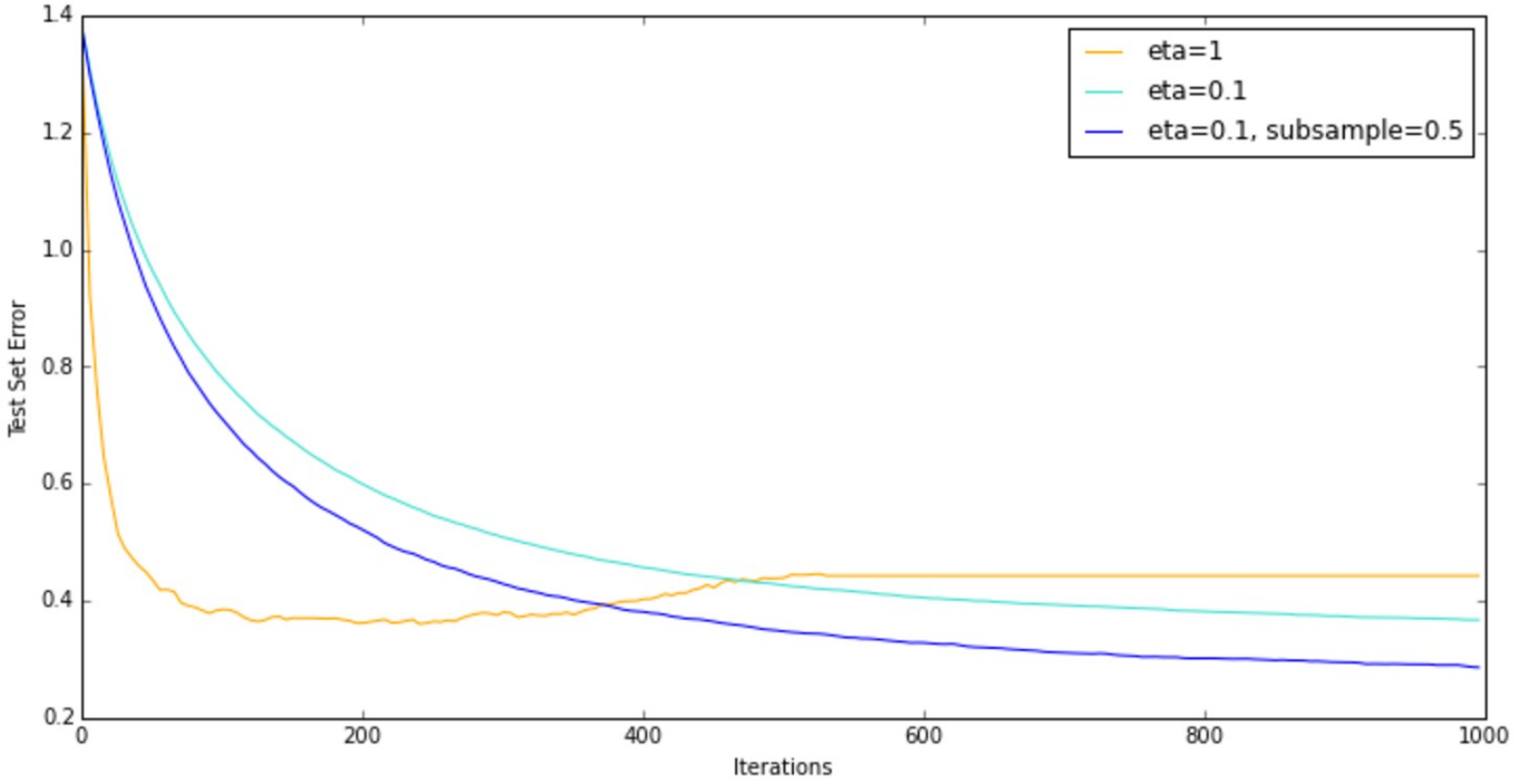
Скорочення кроку - гіперпараметр

Дві стратегії перебору:

1. Зафіксувати η, підбирати N – число ітерацій.
2. Зафіксувати N, підбирати η

Стохастичний градієнтний бустінг

Бегінг - кожний базовий алгоритм навчається **не на всій вибірці, а на деякій її випадковій підвибірці**. Такий підхід ще називається **стохастичним градієнтним бустінгом**.



Похибка на контрольній вибірці залежно від числа ітерацій.

§70 Градієнтний бустінг для регресії й класифікації

Градієнтний бустінг

1 **Ініціалізація:** побудова простого алгоритму $b_0(x)$.

2 **Крок ітерації**:

- **Обчислюється вектор зміщення**

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{n-1}(x_1)), \\ \dots \\ -L'_z(y_\ell, a_{n-1}(x_\ell)) \end{pmatrix}.$$

- **Будується алгоритм**

$$b_n(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2,$$

- Алгоритм $b_n(x)$ **додається в композицію**

3 Якщо критерій зупинки виконаний, **зупинити ітераційний процес**.

- У градієнтному бустінгу як базові алгоритми використовуються дерева рішень.
- Дерева рішень не дуже глибокі (глибина від 2 до 8)
- Скороченння кроку
- Навчання на підвибірках

Градієнтний бустінг для регресії

Типовий функціонал похибки в регресії - це середньоквадратична похибка:

$$\mathbf{MSE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

При цьому функція втрат, що вимірює похибку для одного об'єкта:

$$L(y, z) = (z - y)^2$$

Градієнт

$$L'_z(y, z) = 2(z - y)$$

Вектор зміщень

$$s = \big(-2(a_{N-1}(x_1) - y_1), \ldots \\ \ldots, -2(a_{N-1}(x_\ell) - y_\ell) \big)$$

Градієнтний бустінг для класифікації

Задачі бінарної класифікації $(Y = \{-1,+1\})$

Функція втрат є логістична функція втрат:

$$\sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i a(x_i)\right)\right),$$

де $a(x) \in \boldsymbol{R}$ — оцінка належності до позитивного класу. Якщо $a(x) > 0$, класифікатор відносить об'єкт x до класу +1, а при $a(x) \le 0$ – до класу –1. Причому, чим більше $a(x)$, тим більше класифікатор упевнений у своєму виборі. **Функція втрат та градієнт** записується в так:

$$L(y,z) = \log(1 + \exp(-yz)), \qquad L'_z(y,z) = -\frac{y}{1 + \exp(yz)}.$$

Вектор зміщень *s* у цьому випадку буде мати вигляд:

$$s = \begin{pmatrix} \frac{y_1}{1+\exp(y_1 a_{N-1}(x_1))} \\ \dots \\ \frac{y_\ell}{1+\exp(y_\ell a_{N-1}(x_\ell))} \end{pmatrix}.$$

Новий базовий алгоритм буде налаштовуватися таким чином, щоб вектор його відповідей на об'єктах навчальної вибірки був якнайближче до *s*. Після того, як алгоритм $a_N(x)$ був обчислений, можна оцінити ймовірності належності об'єкта *x* до кожного із класів:

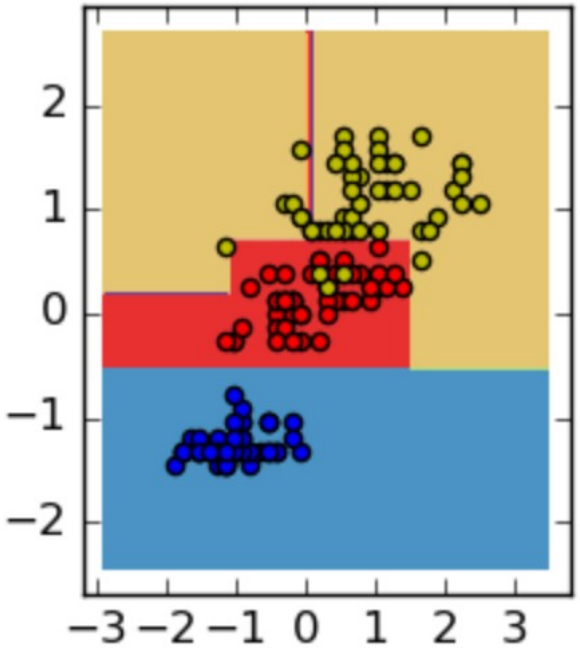
$$P(y=1|x) = \frac{1}{1+\exp\left(-a_N(x)\right)}, \qquad P(y=-1|x) = \frac{1}{1+\exp\left(a_N(x)\right)}.$$

§71 Градієнтний бустінг для дерев рішень

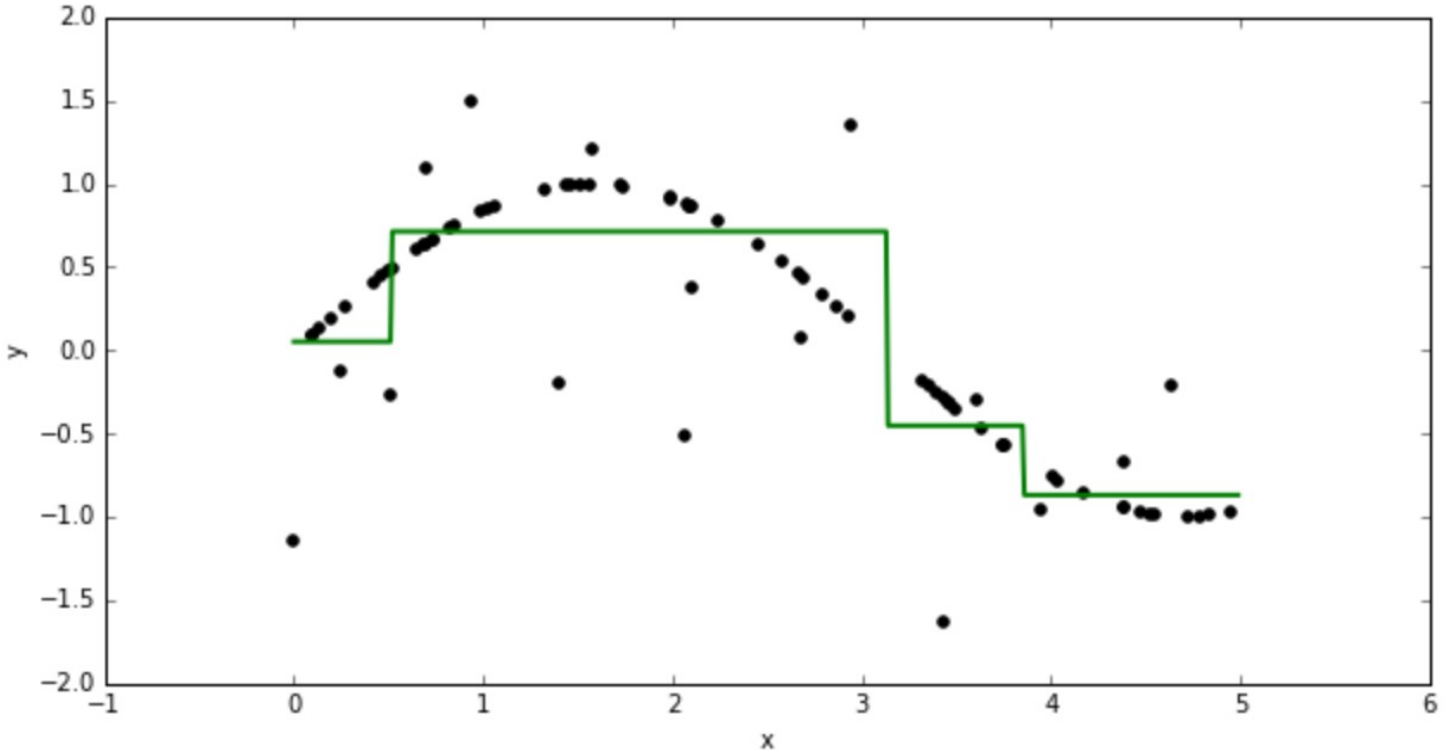
Як застосовувати градієнтний бустінг, якщо базовий алгоритм - це дерева рішень.

Поверхні, які відновлюють дерева рішень

Згадаємо, як виглядають області, що розділяють класи у задачі класифікації:



У **задачі регресії функція**, що відновлює дерево рішень – кусково стала:



Таким чином, дерево рішень, таким чином, розбиває весь простір об'єктів на *J* областей: $R_1, R_2, ... R_J$

У кожній області отримуємо константну відповідь $b_1, b_2, ..., b_J$

Дерево рішень *b*(*x*) можна записати так:

$$b(x) = \sum_{j=1}^{J} [x \in R_j] b_j$$

Градієнтний бустінг для дерев рішень

У градієнтному бустінгу кожний новий базовий алгоритм $b_N(x)$ додається до вже побудованої композиції:

$$a_N(x) = a_{N-1}(x) + b_N(x)$$

Якщо базові алгоритми - це дерева рішень

$$b_N(x) = \sum_{j=1}^{J} [x \in R_{Nj}] b_{Nj},$$

тоді нова композиція $a_N(x)$ буде виглядати в так:

$$a_N(x) = a_{N1}(x) + \sum_{j=1}^{J} [x \in R_{Nj}] b_{Nj}$$

Останній вираз можна проінтерпретувати не тільки як доданок одного дерева рішень, а інакше, як **доданок *J* дуже простих алгоритмів**, кожний з яких повертає постійне значення в деякій області й нуль у всьому іншому просторі.

Можна підібрати кожний прогноз b_{Nj} , де *N* — номер дерева, *j* — номер листка в цьому дереві, таким чином, щоб він був оптимальним з погляду вихідної функції втрат:

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x) + \sum_{j=1}^{J} [x \in R_{Nj}]\, b_{Nj}\right) \to \min_{b_1,\ldots,b_J}$$

Можна показати, що **ця задача розпадається на *J* підзадач**:

$$b_{Nj} = \operatorname{argmin}_{\gamma \in R} \sum_{x_i \in R_j} L\left(y_i, a_{N-1}(x) + \gamma\right).$$

Отже, **структура базового дерева рішень** (структура областей ***Rj***) у градієнтному бустінгу налаштовується мінімізацією середньоквадратичної похибки в одному листку.

Це **дозволяє істотно збільшити швидкість збіжності градієнтного бустінга**.

Наприклад, у задачі класифікації з логарифмічною функцією втрат, практично оптимальні значення для прогнозів у листах мають вигляд:

$$b_{N_j} = -\frac{\sum_{x_i \in \mathbb{R}_j} s_i}{\sum_{x_i \in \mathbb{R}_j} |s_i|(1 - |s_i|)}.$$

§72 Комп’ютерний проект 6: Градієнтний бустінг своїми руками (КП06_Прізвище_grad_boosting.ipynb)