МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра системного аналізу і управління

Т.А. Желдак, О.Б. Владико

МАШИННЕ НАВЧАННЯ

Конспект лекцій
для здобувачів ступеня магістра
зі спеціальності 124 Системний аналіз

Дніпро
НТУ «ДП»
2024

Машинне навчання [Електронний ресурс] : конспект лекцій для здобувачів ступеня магістра зі спеціальності 124 Системний аналіз / уклад.: Т.А. Желдак, О.Б. Владико ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 162 с.

Укладачі:
Т.А. Желдак, канд. техн. наук, доц.,
О.Б. Владико, канд. техн. наук, доц.

Затверджено науково-методичною комісією зі спеціальності 124 Системний аналіз (протокол № 3 від 10. 05. 2024) за поданням кафедри системного аналізу та управління (протокол № 5 від 10. 05. 2024).

Розглянуто теоретичний матеріал з дисципліни «Машинне навчання», опанування якого сприятиме формуванню у здобувачів ступеня магістра компетентностей щодо застосування та використання сучасних технологій обробки даних для вирішення задач класифікації, регресійного аналізу, прогнозування та ухвалення рішень. Сформульовано питання для самоконтролю й критерії оцінювання результатів навчання за дисципліною. Надано перелік рекомендованих джерел.

Відповідальний за випуск завідувач кафедри системного аналізу та управління Т.А. Желдак, канд. техн. наук, доц.

Зміст

стор.

Вступ

Машинне навчання – це один з методів функціонування штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися.

Коли Алан Т'юрінг працював над першими комп'ютерами, він намагався розшифрувати повідомлення німецьких військових, закодовані машиною Енігма. Пошук розшифровки вимагав перебору маси варіантів. Люди з цим завданням справлялися погано, проте машина могла вирішити її порівняно швидко. Очевидно, далеко не для кожного завдання, з яким люди справляються важко, можна написати програму для ефективного пошуку рішення. Більше того, є цілий клас завдань (так звані NP-важкі завдання), які не можна вирішити за розумний час. Можна навіть явно довести, жоден комп’ютер тут дива теж не зробить. Найцікавіше це те, що бувають і завдання, які для людей особливих труднощів не складають, але які чомусь вкрай важко запрограмувати, наприклад:

• перекласти текст з однієї мови іншою;

• діагностувати хворобу за симптомами;

• порівняти, який із двох документів в інтернеті краще підходить під пошуковий запит;

• сказати, що на зображенні;

• оцінити за якою ціною вдасться продати квартиру.

Усі ці завдання мають багато спільного. По-перше, їх рішення можна записати як функцію, яка відображає об'єкти або приклади (samples) у передбачення (targets).

Наприклад, хворих треба відобразити в діагнози, а документи в оцінку релевантності. По-друге, навряд чи ці завдання мають єдино вірне, ідеальне рішення.

Навіть професійні перекладачі можуть по-різному перекласти той самий текст, і обидва переклади будуть вірними. Тож найкраще у цих завданнях — ворог гарного. Зрештою, і лікарі іноді роблять помилки в діагнозах, і ви не завжди можете сказати, що саме зображено на картинці. По-третє, у нас є багато прикладів правильних відповідей (скажімо, перекладів пропозиції іншою мовою або підписів до заданої картинки), а приклади неправильних відповідей (якщо вони потрібні), як правило, не складно сконструювати.

Ми назвемо функцію, що відображає об'єкти в передбаченні, – моделлю, а наявний у нас набір прикладів - навчальною вибіркою або датасетом. Навчальна вибірка складається з:

• об'єктів (наприклад, завантажені з інтернету картинки, історії хворих, активність користувачів сервісу тощо);

• і відповідей (підписи до картинок, діагнози, інформація про звільнення користувачів з сервісу), які ми також будемо іноді називати таргетами.

Лекція №1. Машинне навчання. Вступ

Результати навчання, які формуються:

ДРН1: Розробляти та застосовувати методи, алгоритми та інструменти для навчання нейронних мереж.

ДРН2: Вміння застосовувати знання машинного навчання з вчителем та без нього в системах підтримки прийняття рішень та повсякденній практиці.

1.1. Основи машинного навчання.

Машинне навчання – це цілий світ і при розробці цього курсу ми активно користувалися багатьма джерелами. Найпростіший і очевидніший приклад – це класифікація зображень.

Наприклад, потрібно вміти розрізняти кішок від собак, або ідентифікувати людину за відбитком пальця, або знайти аномалії на рентгенівських медичних знімках і т. п. Тут скрізь вихідні дані (зображення) мають складну структуру і незрозуміло, як математично описати універсальний алгоритм, який би видавав коректні результати (у разі класифікації).

Тому йдуть на деяку хитрість – створюють такий алгоритм, щоб він сам у процесі навчання сформував потрібні критерії для вирішення завдання. Зокрема, це добре роблять глибокі нейронні мережі, особливо в області розпізнавання образів на зображеннях. Головне тут зрозуміти, що не потрібно «пхати» алгоритми машинного навчання в першу чергу в завдання, можливо, вона має набагато простіше (в обчислювальному плані) рішення.

Зазвичай виділяють такі категорії машинного навчання:

- **завдання класифікації;**
- **завдання регресії;**
- **завдання ранжирування**.

Завдання класифікації - це коли потрібно вхідні дані віднести до того чи іншого класу (наприклад, зображення розділити на кішок і собак, або розпізнати вимовлене слово, або за медичними даними видати діагноз тощо)

У **завданнях регресії** зазвичай роблять прогнози у вигляді речових чисел на основі вхідних даних (наприклад, оцінка курсу валют за попередніми показаннями, або прогноз обсягу продаж товару певного виду, або просто побудувати прогноз деякої функції за виміряними емпіричними даними і т.д.). рис. 1.

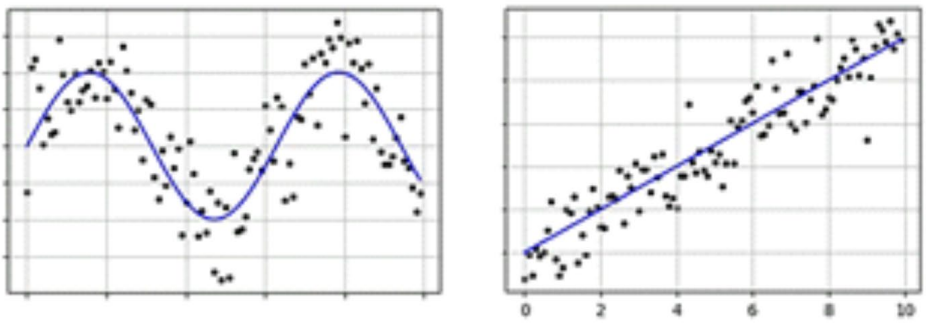


Рис. 1.1 Регресія

Нарешті, третій тип завдань передбачає впорядкування (за деяким критерієм) вхідного набору даних. Класичний приклад – це пошукові системи, які ранжують пошукову вибірку за релевантністю для кожного конкретного користувача.

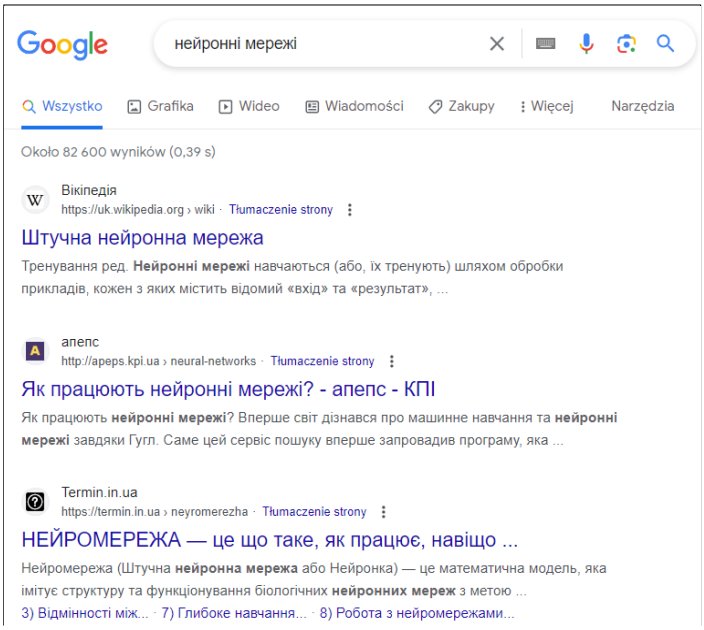


Рис. 1.2. Ранжирування

Усі ці завдання, зазвичай, вирішуються із застосуванням алгоритмів машинного навчання, так як вхідні дані мають досить складну структуру, отже, потрібно побудувати нетривіальні критерії їхньої обробки. Зробити це людині часто просто не під силу, тому потрібно підключати процес навчання так званої навчальної вибірки.

Навчальна вибірка. І ось тут ми підійшли до першого важливого моменту – що являє собою навчальна вибірка? Спочатку – це вихідні дані, прямі результати вимірів. Це може бути те саме зображення, звуковий сигнал, вимірювання росту, ваги людини і т.п. Зазвичай всі ці дані представляються набором деяких чисел (все ж таки ми маємо справу з обчислювальною

технікою, а вона сприймає лише числа). Всі ці виміри зручно представляти у вигляді векторів.

Позначимо їх наступним чином:

$$x_i = [x_{i1}\ x_{i2}\ \ldots\ x_{in}\ ] = [x_{i1}, x_{i2}, \ldots, x_{in}]^T.$$

Тут представлено вимір якогось *i*-го об'єкта. Наприклад, так можна уявити *i*-е зображення, якщо всі його пікселі витягнути в один рядок – в один вектор або амплітуди звукового сигналу, або пари вимірювань ваги і зростання. Я, гадаю, зрозуміло, що вектором можна уявити будь-який набір вимірів.

Ми всі ці набори можемо скласти в матрицю:

$$x_i = [x_1\ x_2\ \ldots\ x_l\ ] = [x_{11}\ x_{21}\ldots x_{l1}\ \ x_{12}\ x_{22}\ldots x_{l2}\ \ \ldots\ \ldots\ldots\ \ x_{1n}\ x_{2n}\ldots x_{ln}\ \ ].$$

В даному випадку матриця X складена з векторів l і кожен вектор містить n чисел. До речі, загалом, розмірність цих векторів може змінюватися, але зазвичай, у завданнях машинного навчання їх намагаються робити рівними. Так із ними простіше працювати.

Але матриця X – це ще не Навчальна вибірка. Нам тут для повноти картини не вистачає вихідних даних:

$$Y = [y_1\ y_2\ \ldots\ y_m\ ] = [y_1, y_2, \ldots, y_m]^T.$$

Їх ще називають цільовими (target). Що то за виходи?

Наприклад, у завданнях класифікації можуть приймати одне з наступних значень:

- y O{-1, +1} – за бінарної класифікації;
- y O{1,2, …, M} – при M-класової класифікації (класи не перетинаються);
- y O$\{0, 1\}^M$ при M-класової класифікації з класами, що перетинаються (тут M – це розмірність простору).

У задачах регресії кожен вхід пов'язаний з одним або декількома речовими числами:

- - y O $\mathbb{R}$ – одновимірне завдання регресії;
- - y O $\mathbb{R}^m$ – m-мірне завдання регресії.

У завданнях ранжирування:

- Y – деяке впорядковане безліч значень.

І ось ця сукупність даних $X^l = \{(x_i, y_i)^l_{i=1}\}$ вже може виступати в якості навчальної вибірки, так як кожному вхідному вектору (спостереження) x_i поставлено у відповідність значення (або вектор) y_i.

На рівні математики вилучення ознак із сирих вихідних даних можна подати у вигляді функціонального (у широкому значенні цього слова) перетворення: $\{f_1(x), f_2(x), \ldots, f_k(x)\}$.

Тут x – це вектор результатів вихідних вимірів, який потім, перетворюється на набір ознак, причому k – будь-яке число, необов'язково дорівнює n (вихідному числу ознак – довжина вектору x).

В результаті, можна записати матрицю F, що містить ці виділені ознаки:

$$F = \|f_1(x_i)\|_{i\times k} = = [f_1(x_1) \ldots f_1(x_l) \ \ldots \ldots \ f_k(x_1) \ldots f_k(x_l)\ ].$$

Отримуємо новий простір ознак F, який спрощує роботу алгоритмів машинного навчання.

1.2. Постановка задачі машинного навчання

Вважатимемо, що набір цих ознак і подається на вхід алгоритмів рис. 3.

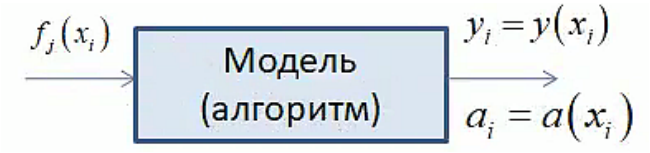


Рис. 1.3. Модель (алгоритм)

Причому, в окремому випадку, коли $f_j(x_i) = x_{ij},\ j = 1, 2, \ldots$ маємо постійні вихідні дані вимірів $\{x_i\}$. Виходить, що ідеальний алгоритм має вміти відображати входи $\{x_i\}$ у відповідні виходи $\{y_i\}$. На рівні математики це можна записати через функціональну залежність між входами та виходами: $y_i = y\ (x_i)$

Причому, функціональна залежність у сенсі слова – це може бути будь-який алгоритм, який пов'язує вхід із виходом. Але ми цей взаємозв'язок не знаємо. Метою навчання якраз і є знайти таку модель, функцію яка вирішує (**decision function**), яка б наближала відповіді: $a_i = a\ (x_i)$, до необхідних $\{y_i\}$ по всьому безлічі можливих вхідних даних X (як для навчальної вибірки, але всіх можливих спостережень тієї ж природи). Це **загальна постановка завдання машинного навчання**.

Звісно, у такому вигляді зовсім незрозуміло, як її вирішувати, як шукати правило перетворення $a(x)$?

Необхідна конкретизація. По суті весь курс машинного навчання - це і є різні варіації рішення поставленого завдання. Напевно, одним із найпростіших підходів (і найчастіше використовуваних), уявити функціонал $a(x)$ у вигляді деякої обраної нами параметричної функції: $a(x) = g(x, \theta)$ з настоюваним набором параметрів θ.

Тобто, ми зводимо завдання навчання до пошуку невідомих параметрів θ і робимо це (у найпростішому, але поширеному випадку) за навчальною вибіркою.

Причому вид самої функції $g(x, \theta)$ може бути як завгодно складним (в математичному сенсі) і в загальному випадку складатися з композиції інших, більш простих функцій. Тобто, вид функції $g(x, \theta)$ має відбивати характер (природу, модель) зміни даних між входом і виходом, а параметри підганяють її під конкретний набір даних.

Щоб це було зрозуміліше, давайте розглянемо класичний приклад такого завдання – лінійну регресію, коли входи і виходи мають яскраво виражену функціональну залежність виду: $y_i = kx_i + b + \varepsilon_i$, (тут $\{\varepsilon_i\}$ – гаусівський (нормальний) шум з нульовим середнім та деякою невеликою дисперсією).

Оскільки ми можемо прогнозувати випадкові відхилення, то найрозумніше описати модель даних як лінійної функції з двома невідомими параметрами k, b: $g(x, k, b) = kx + b$

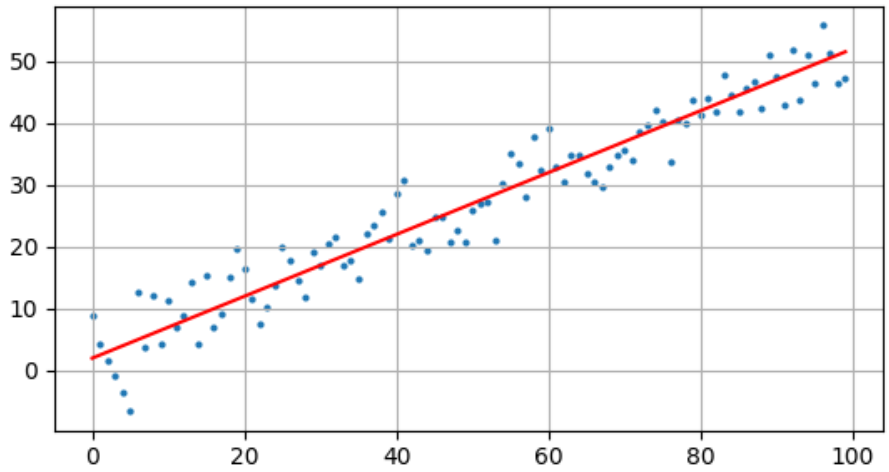


Рис. 1.4. Лінійна регресія

В результаті ми маємо вектор параметрів $\theta = [k, b)]^T$, які визначають конкретний нахил та зсув лінійної функції.

Тобто, вихідна функція $g(x, k, b)$ описує весь клас прямих, а за конкретних $\theta = [k, b)]^T$ отримуємо певну пряму для даних навчальної вибірки.

Ось принцип **параметричної оптимізації**, який розширюється довільні функціональні залежності виходів від входів.

Етап навчання. Добре, вирішальна функція $a(x) = g(x, \theta)$ у вигляді параметричної функції визначено. Як тепер нам знайти значення параметрів θ на безлічі входів та виходів $X \times Y$ навчальної вибірки? Очевидно, вони мають бути підібрані так, щоби зменшити помилки між заданими виходами $y(x)$ та тими, які виходять у нашій моделі $a(x)$: $a(x) - y(x)$, $\forall x$.

Але сама по собі помилка як оптимізована величина не дуже зручна, так як у точці мінімуму (нуля) вона не утворює точки екстремуму. Математично було б краще використовувати функцію, яка б зростала зі збільшенням помилки та зменшувалась би з її зменшенням. Наприклад, можна вибрати такі:

$L(a, x) = |a(x) - y(x)|$ – абсолютна помилка;

$L(a, x) = (a(x) - y(x))^2$ – квадратична помилка.

Подібні функції отримали назву **функцій втрат** $L(a, x)$ (loss function), які фактично обчислюють міру втрат (невідповідності) між нашою моделлю та навчальною вибіркою. Звичайно, таких функцій є величезна кількість і з деякими з них ми будемо познайомимося в міру проходження цього курсу.

Однак, сама по собі **функція втрат** – це випадкова величина, яка залежить від алгоритму $a(x)$ та поточного вектору вхідного x. Тому оптимізувати одне якесь конкретне значення функції втрат – **неправильно**. Потрібно зробити так, щоб на всій навчальній множині, в середньому, помилка була б мінімальною.

В результаті ми приходимо до поняття середнього емпіричного ризику:

$$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} L(a, x_i)$$

де $X^l = \left\{(x_i, y_i)_{i=1}^{l}\right\}$ – навчальна множина.

Фактично це і є показник якості, який потрібно мінімізувати, підбираючи значення вектору параметрів θ.

Наприклад, якщо повернутися до нашої задачі лінійної регресії і як функція втрат вибрати квадрат помилки, то отримаємо функціонал якості у вигляді:

$$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} (a(x_i) - y(x_i))^2 = \frac{1}{l} \sum_{i=1}^{l} (kx_i + b - y(x_i))^2.$$

У разі параметри k, b легко обчислюються з рішення наступної системи лінійних рівнянь:

$\{\frac{\partial Q(a,X^l)}{\partial k} = \sum_{i=1}^{l} (kx_i + b - y(x_i)) \cdot x_i = 0 \; \frac{\partial Q(a,X^l)}{\partial b} = \sum_{i=1}^{l} (kx_i + b - y(x_i)) = 0$ **– метод найменших квадратів (МНК)**

Отже, резюмуючи матеріал цього пункту, можна відзначити такі чотири пункти:

1. Загальне завдання машинного навчання ставиться як пошук моделі $a(x)$ яка найкраще описує природу залежності вхідних даних $\{x_i\}$ та цільових вихідних значень $\{y_i\}$.

2. Завдання пошуку найкращої моделі часто зводять до завдання параметричної оптимізації функції виду $a(x) = g(x, \theta)$.

3. Для знаходження відповідних параметрів θ вводиться функція втрат $L(a, x)$ та визначається середній емпіричний ризик $Q(a,X^l)$. Мінімізуючи цей показник якості, отримуємо набір параметрів θ з навчальної вибірки.

4. На основі знайденої залежності $a(x) = g(x, \theta)$ надалі обчислюються вихідні значення $a_k = a(x'_k)$ при пред'явленні нового вхідного вектору x'_k тієї ж природи, що й під час навчання.

Ці чотири етапи є загальним принципом, що лежить в основі всіх алгоритмів машинного навчання

1.3. Лінійні моделі. Поняття перенавчання

На попередньому параграфі ми побачили, що завдання машинного навчання – це завдання оптимізації, зокрема мінімізації емпіричного ризику:

$$Q(a, X^l) = \frac{l}{l}\sum_{i=l}^{l} \vdots\, L(a, x_i)$$

залежить, загалом, від моделі a (x) та виду функції втрат $L(a, x)$.

Тобто, нам потрібно обрати таку модель, щоб функціонал набував найменшого значення на навчальній вибірці.

Математично це можна записати так:

$$\mu\left(X^l\right) = arg\ arg\ min_{a\in A} Q(a, X^l)$$

Тут $\mu(X^l)$ – знайдена модель на етапі навчання за всіма можливими моделями aOA; *argmin* – визначає значення аргументу, у якому функція приймає найменше значення.

У разі параметричних моделей $a(x) = g(x, \theta)$ цей вираз може бути записано щодо вектору параметрів θ:

$$\mu\left(X^l\right) = arg\ arg\ min_{\theta} Q(a, X^l).$$

Після навчання ми приймаємо $a = \mu(X^l)$ і використовуємо знайдену залежність у продакшені (при практичній реалізації).

На рисунку нижче червоними точками показано значення функції, що беруть участь у розрахунках параметрів $\{\theta_i\}$, а зеленими – точки, в яких будується прогноз щодо отриманої моделі.

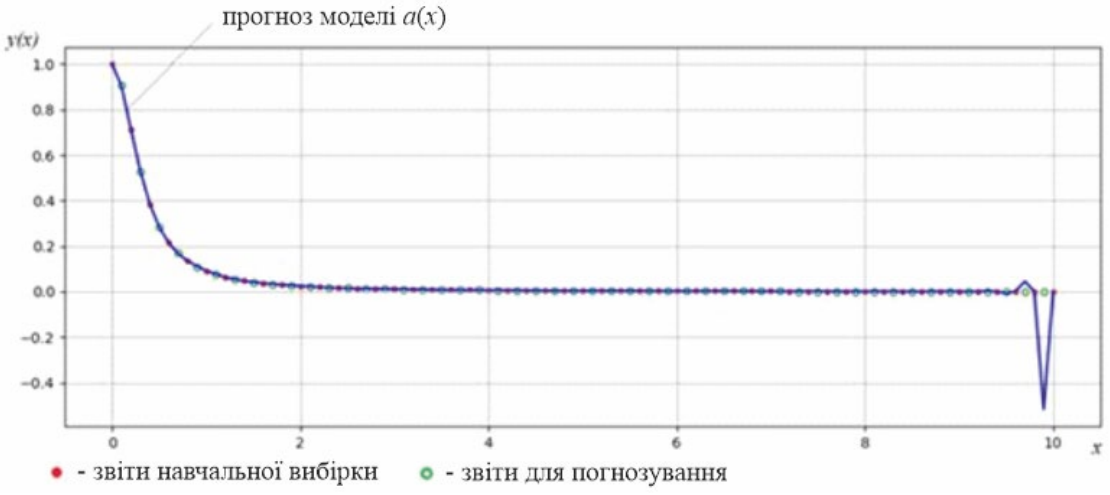


Рис. 1.8. Прогноз моделі

Як бачите, при великих значення x поліном поводиться абсолютно непередбачуваним чином. Це відомий факт, знайомий всім математикам. Стосовно нашого завдання це означає, що така модель погано будуватиме прогнози в нових, невідомих їй точках. Тобто, з одного боку, поліном високого ступеня добре описує точки навчальної вибірки, але непридатний для побудови прогнозів у нових точках.

А ось із поліномами менших ступенів таких проблем, як правило, немає.

Наведемо ще один графік, де показано, як змінюється значення емпіричного ризику для двох моделей:

$a_1(x)$ – обчислена за всіма точками функції $y(x)$;

$a_2(x)$ – обчислена по половині точок, взятих через відлік.

В результаті, маємо два вирази для емпіричних ризиків:

$$Q_1(a_1, X^l) = \frac{1}{l}\sum_{i=1}^{l} (a_1(x_i) - y(x_i))^2$$

$$Q_2(a_2, X^l) = \frac{1}{l}\sum_{i=1}^{l} (a_2(x_i) - y(x_i))^2$$

та їх графіки при різних ступенях поліномів $n = 1, 2, \ldots, 64$:

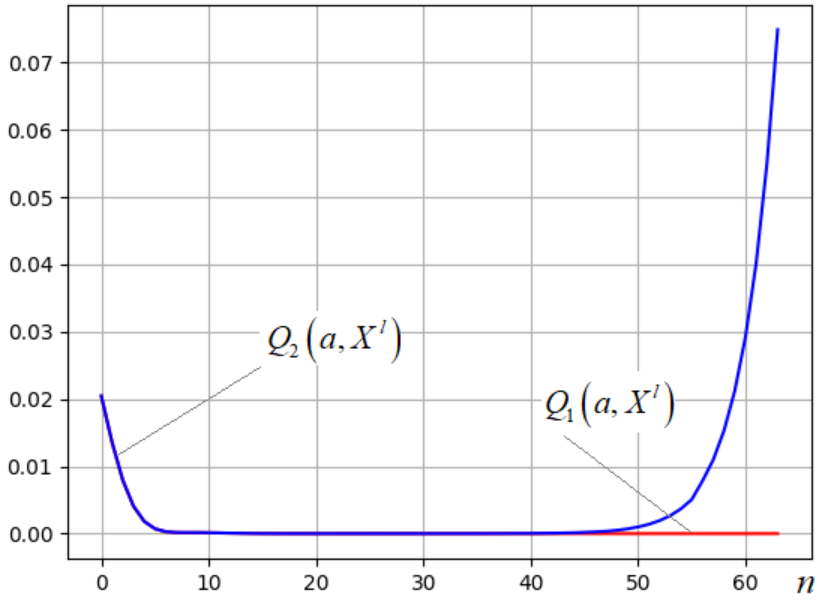


Рис. 1.9 Ефект перенавчання (overfitting)

Тут синій графік – це $Q_2(a, X^l)$, а червоний – $Q_1(a, X^l)$.

Добре видно, що спочатку збільшення ступеня поліномів призводить до поліпшення ступеня апроксимації функції, а потім, при великих ступенях, синій графік різко зростає і розходиться з червоним. Це саме пов'язано з непередбачуваністю поведінки поліномів з великими ступенями і погіршенням їх передбачуваної здатності.

З погляду машинного навчання – це яскравий приклад ефекту **перенавчання** (overfitting). Тобто перенавчання – це невідповідність знайденої моделі $a(x)$ закону зміни даних, частина яких була представлена у навчальній вибірці. При цьому дані навчальної вибірки описуються моделлю добре. В результаті, така перенавчена модель не має достатніх узагальнюючих здібностей – її не можна розширити на довільний набір даних X тієї ж природи, що й у навчальній вибірці. А це саме те, що ми хочемо від нашої моделі – застосовувати її для нових спостережень та отримувати коректні результати. Тобто тут важливо не просто провести функцію з емпіричних залежності, а виділити їхню модель, закон природи. У цьому полягає головне завдання машинного навчання.

Треба сказати, що будь-яка модель $a(x)$, Знайдена за емпіричними даними, має той чи інший ступінь перенавченості. Позбутися повністю цього ефекту неможливо, ми завжди будемо так чи інакше підлаштовуватися під дані навчальної вибірки. І все, що можемо зробити – це мінімізувати цей ефект і, як наслідок, підвищити узагальнюючу здатність нашої моделі.

1.4. Способи оцінювання ступеня перенавчання моделей

Нам потрібно вміти оцінювати якість знайдених моделей $a(x)$. Очевидно, що вони, по-перше, повинні показувати хороші результати на виборці навчальної і, по-друге, узагальнюватися на інші спостереження $\{x'_k\}$, відсутні під час навчання. Тобто модель не повинна бути перенавченою. І тут постає закономірне питання, як визначати якість знайдених моделей?

Перший показник (середні втрати на вибірці) ми вже вміємо визначати – це величина емпіричного ризику: $Q(a, X^l) = \frac{1}{l}\sum_{i=1}^{l} L(a, x_i)$.

А ось другий – ступінь перенавченості, дещо складніший. Тут є низка загальноприйнятих підходів. Розглянемо їх у порядку.

Оцінка узагальнюючої здібності по відкладеній вибірці (hold-out).

Перший варіант досить простий і очевидний. Потрібно всю вихідну розмічену вибірку розділити на дві частини: власне, навчальну та відкладену (тестову, hold-out).

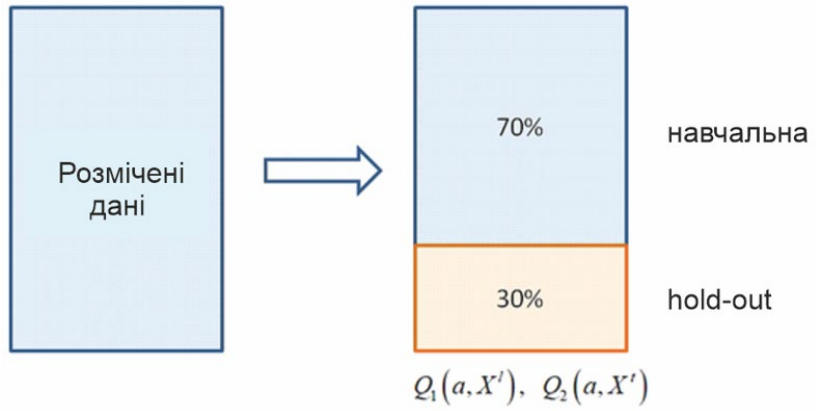


Рис. 1.10 Оцінка узагальнюючої здібності по відкладеній вибірці (hold-out)

Зазвичай, такий поділ робиться випадковим чином (вектори даних відбираються випадково, без повторень), наприклад, щодо 70% на 30%, тобто 70% залишають для навчання, а 30% – для тестування отриманої моделі. Потім, після процесу навчання для знайденої моделі $a(x)$ обчислюються емпіричні ризики для навчальної та тестової вибірок: $Q_1(a, X^l)$, $Q_2(a, X^t)$.

Якщо показники виявляються близькими, робиться висновок про прийнятність моделі. Інакше вона вважається перенавченою і її слід скоригувати.

Недоліком такого підходу є залежність отриманої оцінки якості від способу розбиття вибірки на навчальну та відкладену. Може випадково вийти так, що на обох модель покаже хороші результати, а на практиці в процесі її застосування якість буде помітно нижчою. Це означає, що ми неправильно визначили її узагальнюючі здібності і таку модель потрібно виправляти.

Ковзний контроль (leave-one-out).

Напевно, найоб'єктивніший підхід до оцінки пере навченості моделі на основі розбиття навчальної вибірки – це так званий ковзний контроль (leave-one-out). Ідея дуже проста. Давайте з усієї вибірки прибирати по черзі по одному вектору спостережень і навчатися щоразу на вибірці, що залишилася.

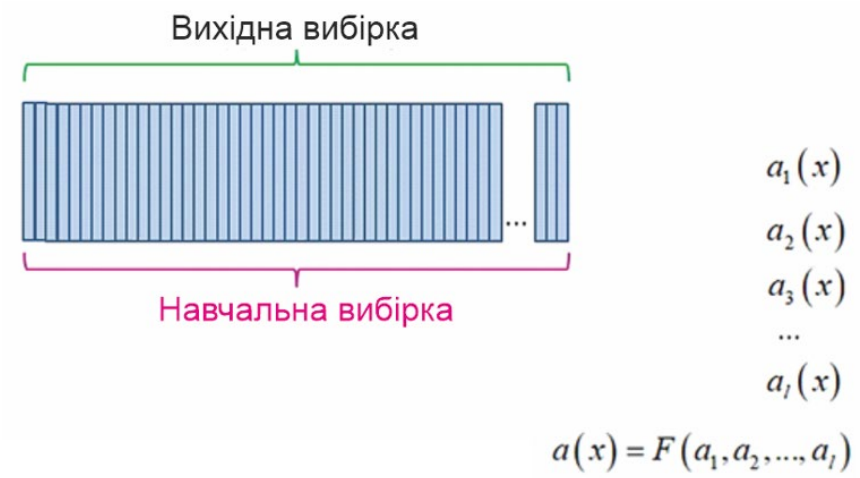


Рис. 1.11 Ковзний контроль (leave-one-out)

У результаті, у нас вийде безліч різних моделей: $a_1(x)$, $a_2(x)$, $a_3(x)$,…, $a_i(x)$. Потім, на їх основі, сформуємо загальну модель: $a(x) = F(a_1, a_2,…, a_l)$ яка б мала кращі узагальнюючі здібності. Як це можна зробити, я скажу трохи згодом. А тут зазначимо, що такий підхід хоч і позбавлений нестачі попереднього (нам тут уже не потрібно вирішувати які спостереження залишити в навчальній вибірці, а які перемістити у відкладену), але він, по-перше, обчислювальний, так як процес навчання повторюється велику кількість разів, а по-друге, нам потрібно вирішити, як на основі отриманих моделей сформувати одну єдину, щоб вона добре вирішувала поставлене завдання. Тому такий підхід на практиці для оцінки перенавченості, як правило, не використовують, а користуються іншим, дещо грубішим методом, але більш простим з погляду обчислень.

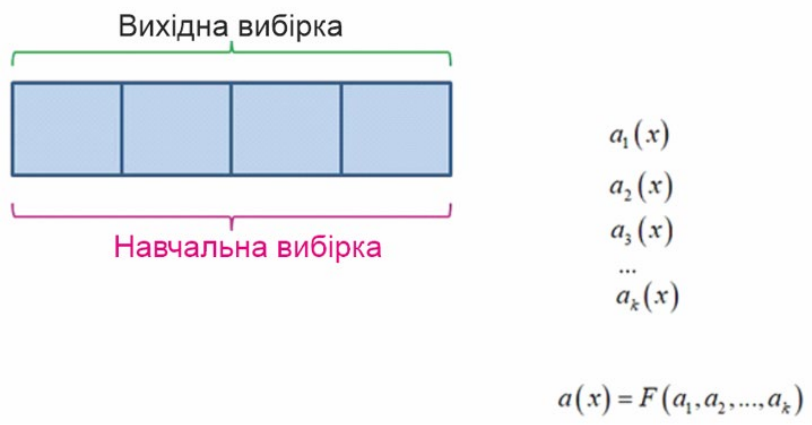


Рис. 1.12 Крос-валідація (cross-validation)

Крос-валідація (cross-validation). Цей метод називають крос-валідацією (англійською cross-validation або *k*-fold). Тут вся вибірка розбивається на окремі групи спостережень і кожної ітерації, одна група використовується як перевірна, інші складають навчальне безліч.

До кожного навчального набору формується своя модель. У результаті, у нас вийде кілька різних моделей (за кількістю груп): $a_1(x)$, $a_2(x)$,…, $a_k(x)$

Потім, на їх основі формується загальна модель: $a(x) = F(a_1, a_2,\ldots, a_k)$

Тобто крос-валідація працює за тим же принципом, що й ковзний контроль, тільки тут ми оперуємо не окремими векторами, а групами векторів. Це значно скорочує обсяг обчислень і може бути використано в практичних завданнях на етапі побудови моделі.

1.4. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Що не відноситься до завдань машинного навчання?
2. Що таке гіперпараметр?
3. Що не відноситься до цільових даних (target)?
4. Чим завдання класифікації відрізняється від задачі регресії?
5. Наведіть приклади задач класифікації та регресії.
6. Що таке алгоритм (модель)?
7. Опишіть загальну постановку завдання машинного навчання.
8. Надайте формулу описання лінійної регресії.
9. Критерій якості моделі.
10. Що таке перенавчання моделі?

1.4. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.
2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

Лекція №2. Задачі бінарної класифікації

Результати навчання, які формуються:

ДРН2: Вміння застосовувати знання машинного навчання з учителем та без нього в системах підтримки прийняття рішень та повсякденній практиці.

ДРН3: Знати математичну основу, ідею та алгоритми класифікації об'єктів у дійсному, категорійному та змішаному просторах.

2.1. Рівняння гіперплощини у задачах бінарної класифікації

Давайте, для простоти, поставимо завдання двокласової класифікації точок на площині, які лінійно розділяються. У цьому випадку кожен вхідний вектор матиме два виміри: $x_i = [x_{i1}, x_{i2}]^T$, $i = 1, 2,\ldots$

А цільові вихідні значення – одне із двох значень:

$$x_i = \{+1, якщо\ x_i \in C_1 \quad -1, якщо\ x_i \in C_2\ .$$

Графічно, розподіл двох класів можна уявити, таким чином:

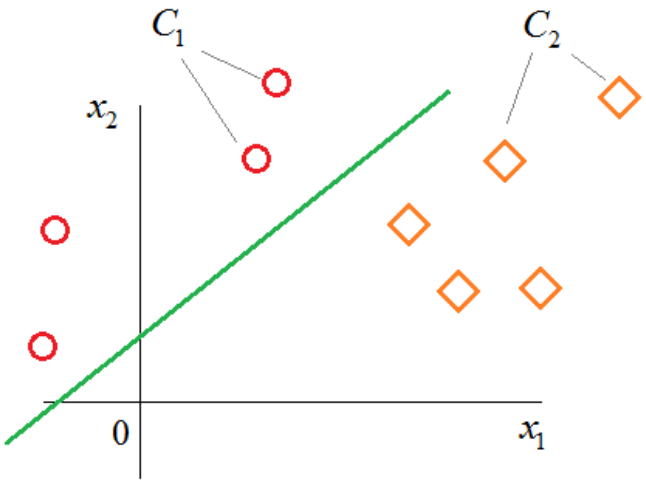


Рис. 2.2

Як модель у цій задачі виступає рівняння поділяючої прямої: $a(x) = kx + b$. Давайте тепер поставимо питання, що це за безліч точок, що лежать на прямій $a(x)$? Очевидно, це точки, у яких координати пов'язані виразом: $x_2 = a(x_1)$.

Розпишемо його, отримаємо:

$x_2 = kx_1 + b$

$x_2 - kx_1 - b = 0$

І в загальному вигляді можна записати так: $w_1x_1 + w_2x_2 - w_0 = 0$.

Якщо виділити два вектори: $w = [w_1, w_2]^T$; $x = [x_1, x_2]^T$ то цю ж формулу можна переписати у вигляді: $\langle w, x\rangle - w_0 = 0$.

Тут $\langle w, x\rangle = w^T \cdot x = w_1x_1 + w_2x_2$ – скалярний добуток двох векторів.

А вільний коефіцієнт w_0 лише визначає усунення прямої по осі ординат.

Давайте поки що його прирівняємо нулю $w_0 = 0$ і вважатимемо, що пряма проходить через початок координат.

А як вектор: $w = [-1, 1]^T$, тобто $w_1 = -1$; $w_2 = 1$.

При цих параметрах отримуємо наступне рівняння прямої:

$-1 \cdot x_1 + 1 \cdot x_2 = 0$

$x_2 = x_1$ Ю $a(x) = x$

Тобто це пряма, яка проходить під 45 градусів через початок координат, а вектор w ортогональний цієї прямої:

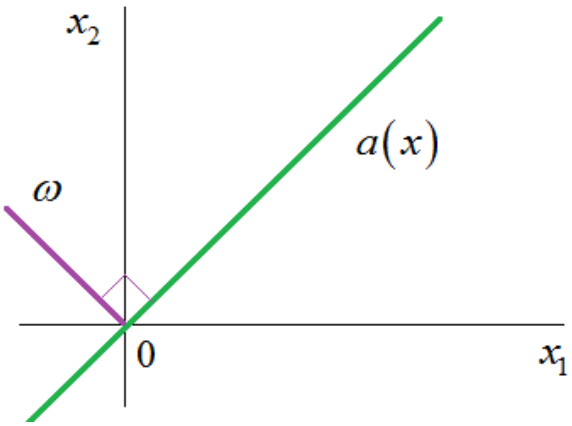


Рис. 2.3

Можна помітити, що вектор w завжди ортогональний розділяє прямий $a(x)$, оскільки, згадуючи шкільну математику, скалярний добуток можна розписати і так: $\langle w, x \rangle = w_1 x_1 + w_2 x_2 = |w| \cdot |x| \cos \alpha = 0$.

Якщо довжини векторів відмінні від нуля (як у нашому випадку) $|w| \neq 0$, $|x| \neq 0$, то отримаємо косинус кута між ними рівний нулю. А це справедливо тільки для ортогональних векторів, тобто при $\alpha = 90^0$.

Картина не зміниться, якщо вільний коефіцієнт

$w_0 \neq 0$. У цьому випадку пряма обертатиметься не щодо точки 0, а в точці вище або нижче по осі ординат. Головне, що тут, як і раніше, вектори $w \perp x$ (Залишаються ортогональними один одному).

Це дуже важливий момент при вирішенні задач класифікації вхідних даних за допомогою лінійної моделі: $a(x, w, w_0) = \langle w, x \rangle - w_0$.

І ви зараз побачите чому. Але спочатку зазначимо, що цей вільний член також можна внести в скалярний добуток, якщо вихідні двовимірні вектори доповнити третім виміром з константним значенням -1: $w = [1, x_{i1}, x_{i2}]^T$

Тоді вектор w набуває вигляду: $w = [w_0, w_1, w_2]^T$ і ми отримуємо рівняння безлічі точок на площині в тривимірному просторі: $a(x, w) = \langle w, x \rangle = -w_0 + w_1 x_1 + w_2 x_2 = 0$.

Як бачимо, тут, як і раніше, вектори $w \perp x$ і графічно все можна уявити, таким чином:

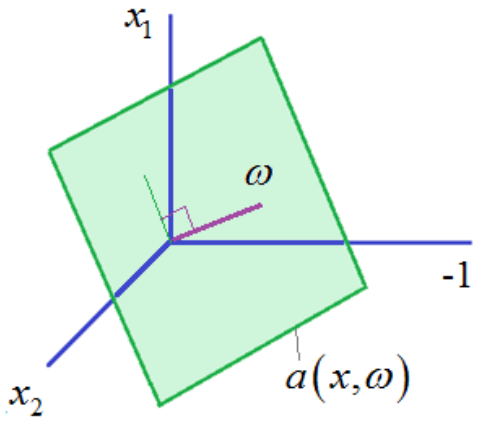


Рис. 2.4

Причому, гіперплощина в тривимірному просторі ознак буде обертатися щодо нуля, хоча спочатку у двовимірному у нас було зміщення по осі ординат. Цей приклад добре показує, що для будь-якого вектору розмірності n ми завжди можемо додати один новий вимір, щоб сформувати гіперплощину через нуль.

Добре, ми з вами зрозуміли, як описується роздільна гіперплощина в задачах класифікації лінійно роздільних образів і що вектор параметрів w, який її задає, завжди перпендикулярний цій площині. Але як нам тепер, використовуючи цю модель $a(x, w)$, відокремлювати об'єкти одного класу від іншого?

Робиться це дуже просто. Давайте я покажу це на прикладі двовимірного простору, але та сама ідея буде застосовна до простору будь-якої розмірності.

Дивіться, якщо модель $a(x, w)$ сформована вірно і лінія дійсно ділить образи на два класи, то для будь-якої точки першого класу C_1 скалярний добуток з параметрами вектор w буде позитивним, а для точок класу C_1 – негативним:

$$\{\langle x,w\rangle > 0, x \in C_1 \; \langle x,w\rangle < 0, x \in C_2$$

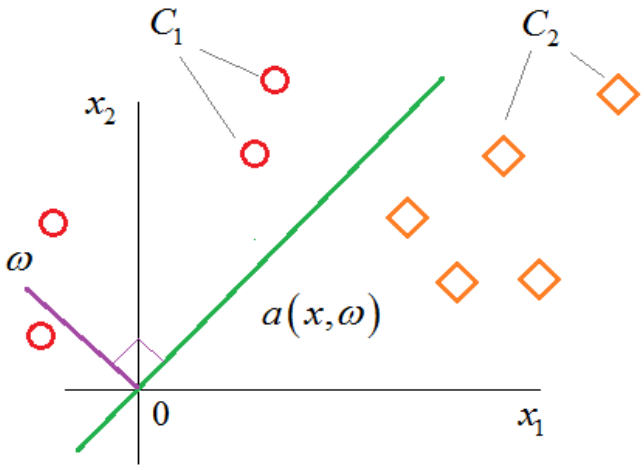


Рис. 2.5

Чому так відбувається, добре видно із рисунка. Радіус-вектори точок для класу C_1 утворюють гострий кут із вектором w. Оскільки скалярне добуток – це, фактично, обчислення косинуса кута між векторами, то гострих кутів будемо отримувати позитивні значення. Крапки ж класу C_2 утворюють тупі кути з вектором w. Отже, їм матимемо негативні значення. Бачите, як математично можна дуже просто розділити точки, що лежать по різні боки від прямої чи гіперплощини, так як у багатовимірному просторі ознак ми матимемо той самий принцип.

У результаті алгоритм класифікації образів за допомогою моделі $a(x, w)$ можна записати у вигляді: $a(x, w) = sign\ (\langle w, x\rangle)$.

Тут $sign()$ – знакова функція, яка повертає +1 для позитивних чисел та -1 – для негативних: $sign(x) = \{-1, x < 0 \ +1, x > 0$.

Ви можете спитати, а що робити, якщо скалярний добуток дасть точно 0. У нулі ця функція не визначена. Насправді, у практиці, ймовірність того, що точка виявиться точно на роздільній гіперплощині, майже дорівнює нулю. Але для надійності ми можемо покласти, якщо виявиться: $\langle w, x\rangle = 0$, то видаємо відмову у класифікації (значення 0), так як тут справді неясно, до якого класу віднести цей образ.

2.2. Функції втрат у задачах лінійної бінарної класифікації.

У попередньому пункті ми з вами розглянули найпростіший приклад навчання моделі бінарної лінійної класифікації у такому вигляді:

$$a(x, w) = sign(\langle w, x\rangle) = \{-1, x \in C_1 \ +1, x \in C_2$$

Де $w = [\ w_0, w_1, \ldots, w_n]^T$ – вектор настроюваннях (у процесі навчання) параметрів.

І все завдання звели до пошуку всього одного коефіцієнта w_1: $w = [\ w_1, -1]^T$.

Крім того, ми фактично придумали алгоритм для знаходження цього єдиного коефіцієнта. Однак, хотілося б звести це завдання до математичної оптимізації за деяким критерієм якості, а також узагальнити на довільне число коефіцієнтів.

Почнемо із визначення показника якості. На попередньому занятті він у нас обчислювався за кількістю неправильно класифікованих спостережень (вхідних векторів):

$$Q(a, X^l) = \sum_{i=1}^{l} [M_i < 0] \rightarrow min$$

Нагадаю, що тут як відступ ми використовуємо величину:

$$M_i = \langle w, x\rangle \cdot y_i$$

а як функція втрат:

$$L_i(w) = [M_i < 0].$$

де квадратні дужки – це індикатор помилки. Відповідно до нотації Айзерсона вони повертають 1 для *True* та 0 – для *False*.

Якщо зобразити цю функцію втрат, що залежить від відступу, отримаємо ступінчастий графік:

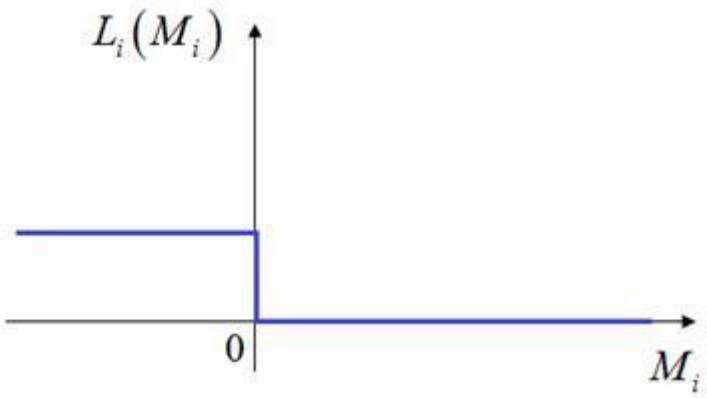


Рис. 2.8

Як ви розумієте, він не має похідних у точках перегину, та й у всіх інших похідні дорівнюватимуть 0. Тобто, ця функція не підходить для завдання математичної оптимізації при пошуку вагових коефіцієнтів *w*. Візьмемо якусь схожу, але функцію, що диференціюється. І таких функцій вигадували безліч.

Ось деякі, найвідоміші:

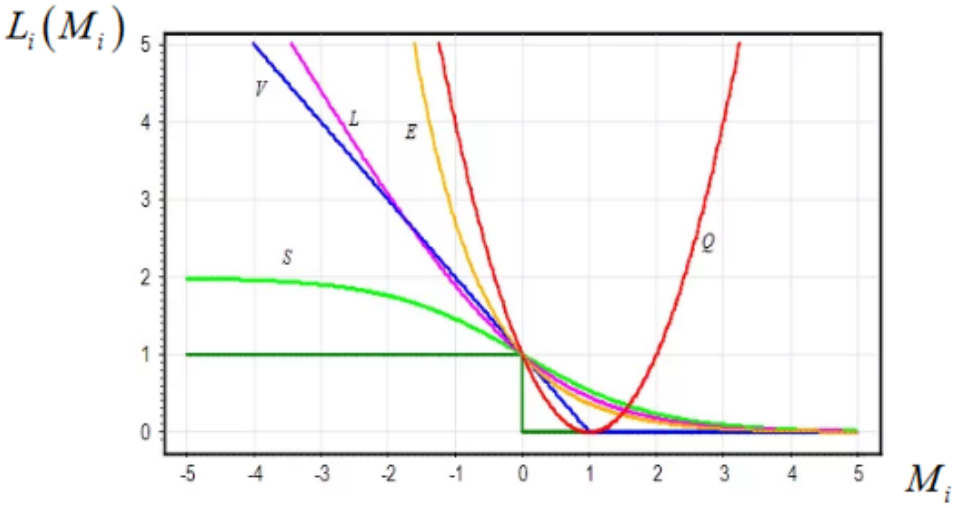


Рис. 2.9

- $V(M) = (1 - M)_+$ – шматкова-лінійна (SVM);
- $H(M) = (-M)_+$ – кусочно-лінійна (Hebb's rule);
- $L(M) = \log_2(1+e^{-M})_+$ – логарифмічна (LR);
- $Q(M) = (1 - M)^2$ – квадратична (FLD);
- $S(M) = 2 \cdot (1 + e^{M})^{-1}$ – сигмоїдна (ANN);
- $E(M) = e^{M}$ – експонентна (AdaBoost).

Особливість всіх цих функцій (крім того, що вони всюди диференційовані) у тому, що вони за значенням більші, або рани вихідної

порогової. Отже, мінімізуючи будь-яку з них, ми автоматично мінімізуватимемо втрати і при пороговій функції.

Давайте подивимося, як це можна використовувати для квадратичної функції втрат:

$L_i(w) = (1 - M)^2 = (1 - w^T \cdot x_i \cdot y_i)^2$

Тоді функціонал якості запишеться у вигляді:

$$Q(w) = \sum_{i=1}^{l} (1 - w^T \cdot x_i \cdot y_i)^2 \to min_w$$

Тут ми можемо знайти оптимальний вектор коефіцієнтів w суто математично:

$$\frac{dQ(w)}{dw} = -2 \cdot \sum_{i=1}^{l} (1 - w^T \cdot x_i \cdot y_i) \cdot x_i^T \cdot y_i = 0$$

Звідки

$$\sum_{i=1}^{l} x_i^T \cdot y_i - w^T \sum_{i=1}^{l} x_i \cdot x_i^T \cdot y_i^2 = 0$$

$$w^T = \sum_{i=1}^{l} x_i^T \cdot y_i \cdot (\sum_{i=1}^{l} x_i \cdot x_i^T)^{-1}$$

2.4. Стохастичний градієнтний спуск SGD та алгоритм SAG

На попередньому занятті ми з вами запровадили гладкі функції втрат для вирішення задачі бінарної класифікації. І побачили, що за квадратичної функції рішення можна знайти за формулою:

$w^T = \sum_{i=1}^{l} x_i^T \cdot y_i \cdot (\sum_{i=1}^{l} x_i \cdot x_i^T)^{-1}$.

Однак далеко не завжди реальні завдання вдається так елегантно і просто вирішити. Головні проблеми виникають через погану обумовленість зворотної матриці (тобто вона може не існувати) і з дуже великою розмірністю простору ознак. Наприклад, у нейронних мережах розмір вектору коефіцієнтів досягає 100 000, 1 000 000 і це ще не найбільші мережі. Звертати такі великі матриці обчислювально дуже складно. Тому потрібні інші чисельні підходи до завдання оптимізації показника якості (емпіричного ризику):

$Q(w) = \sum_{i=1}^{l} L_i(w) \to min_w$.

Для диференційованих функцій найпростішим і найпоширенішим методом їх оптимізації (пошуку екстремуму) є градієнтні алгоритми.

Ідея градієнтних алгоритмів дуже проста. Якщо взяти яку-небудь функцію, що диференціюється, з однією точкою мінімуму, наприклад,

параболу: $Q(w) = w^2 + 2$, то можна поступово уточнювати деяке початкове значення коефіцієнта $w^{(0)}$, переміщаючись у напрямку антиградієнта функції:

$$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla Q(w^{(t)}).$$

Так як антиградієнт показує напрямок до найближчої точки екстремуму функції, то при правильному доборі кроку η_t можна досить точно знайти точку мінімуму функції.

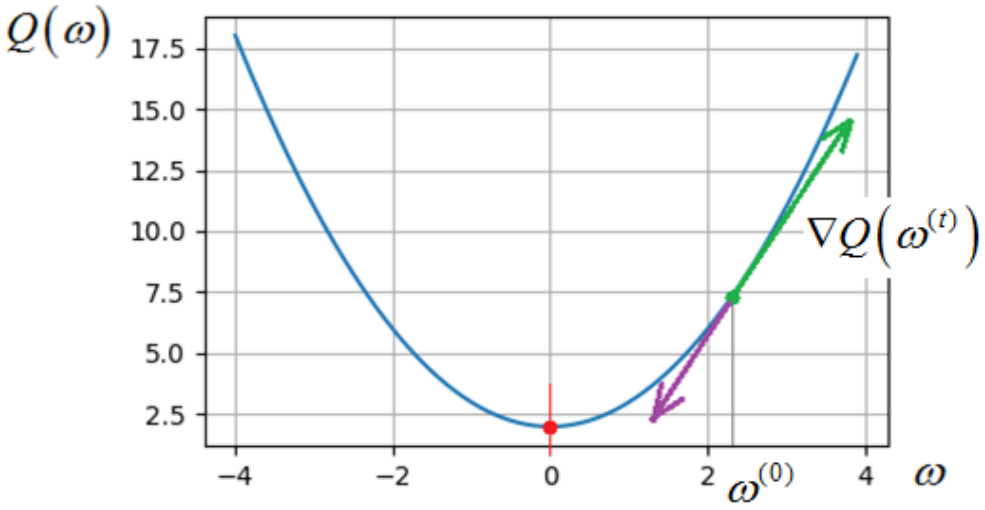


Рис. 2.10

Тому я не буду вдаватися в деталі його класичної реалізації (там нічого складного немає), а відзначу принцип його роботи для вектору параметрів:

$$w = [w_1, w_2, \ldots, w_n]^T.$$

У цьому випадку ми маємо багатовимірну функцію і на кожній ітерації нам потрібно обчислювати похідні по кожному з значень:

$$\nabla Q(w_t) = \left[\frac{\partial Q(w_t)}{\partial w_1^{(t)}} \ \ldots \ \frac{\partial Q(w_t)}{\partial w_n^{(t)}}\right].$$

Отримаємо вектор із приватних похідних, яким потім уточнимо значення вектору коефіцієнтів на новій ітерації:

$$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla Q(w^{(t)})$$

Тобто загальний принцип зберігається, просто ми переходимо у векторну форму опису градієнтного алгоритму.

Стохастичний градієнтний спуск (SGD). Якщо розписати цю формулу ще докладніше, то замість градієнта функціоналу у нас вийде сума градієнтів функції втрат за навчальною вибіркою:

$$w^{(t+1)} = w^{(t)} - \eta_t \cdot \sum_{i=1}^{l} \nabla L_i(w^{(t)}, x_i).$$

Виходить, щоб зробити один крок класичного градієнтного спуску, нам потрібно обчислити похідні за функціями втрат для всіх об'єктів $\{x_i\}$ навчальної вибірки та скласти їх. Уявляєте, який тут може виходити обсяг обчислень, якщо наша вибірка складається із сотень тисяч, а то й мільйонів векторів $\{x_i\}$?

А такі розміри вибірок у завданнях машинного навчання є звичайною справою. Тому за практичної реалізації вихідний градієнт (всю суму цілком) замінюють псевдоградієнтом (субградієнтом), який обчислюється значно простіше. І єдина вимога до псевдоградієнту – його напрямок має в середньому, на кожному кроці, утворювати гострий кут із справжнім градієнтом.

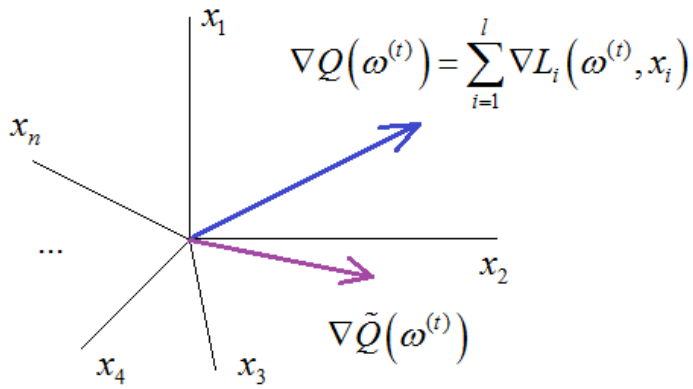


Рис. 2.11

Тут через С $Q(w^{(t)})$ позначений псевдоградієнт.

Виникає питання, що нам взяти як псевдоградієнт, тобто як скоротити обсяг обчислень для істинного градієнта і при цьому зберегти збіжність алгоритму до точки локального мінімуму? Найпростіше і радикальне, що можна зробити, на кожному кроці з усієї суми брати лише одне спостереження (одне доданок) і по ньому обчислювати поточне наближення:

$$w^{(t+1)} = w^{(t)} - \eta_t \cdot \mathrm{C}L_k\,(w^{(t)}, x_k),$$

де С $\tilde{Q}(w^{(t)}) = CL_k\,(w^{(t)}, x_k)$ – псевдоградієнт; k – випадковий індекс вектору з навчальної вибірки.

Чи буде у цьому випадку наш псевдоградієнт утворювати в середньому гострий кут із справжнім градієнтом? Іншими словами, чи будемо ми в середньому рухатися у напрямку локального мінімуму функціоналу якості?

Так, можна показати, що за законом великих чисел, за великої кількості кроків (наближень) цей псевдоградієнт сходить до справжнього градієнта. Отже, рано чи пізно ми досягнемо за його допомогою точки мінімуму.

Ця ідея покладена основою методу стохастичного градієнтного спуску (Stochastic Gradient Descent – *SGD*). Іноді його називають методом Роббінса-Монро.

Алгоритм можна записати у вигляді наступного псевдокоду:

Вхід: вибірка X^l, крок збіжності η, швидкість «забуття» λ.

Вихід: вектор вагових коефіцієнтів w.

1) ініціалізація ваг w деякими початковими значеннями;

2) початкове обчислення функціоналу якості $\underline{Q}(w) = \frac{1}{l}\sum_{i=1}^{l} L_i(w)$;

3) цикл;

4) випадковий вибір спостереження x_k з X^l;

5) обчислення функції втрат $\varepsilon_k = L_k(w)$;

6) крок псевдоградієнтного алгоритму: $w = w - \eta \cdot C L_k(w)$;

7) перерахунок функціоналу якості: $\underline{Q} = \lambda\varepsilon_k + (1 - \lambda)\,\underline{Q}$;

8) поки $\underline{Q}$ та/або w не досягнуто заданих значень.

Тут у вас може викликати здивування формула перерахунку середнього показника якості $\underline{Q} = \lambda\varepsilon_k + (1 - \lambda)\,\underline{Q}$.

Дивіться, у процесі навчання алгоритму нам потрібно вміти оцінювати якість знайдених вагових коефіцієнтів, а це є функціонал $\underline{Q}(w)$.

Наприклад, коли він досягає певного значення, процес навчання можна зупинити. Але формально він обчислюється як середнє арифметичне від функцій втрат по всій вибірці. А це знову великий обсяг обчислень, якого саме хочемо уникнути. Тому в алгоритмах SGD функціонал $\underline{Q}(w)$ обчислюється з урахуванням експоненціального середнього.

Звідки взялася ця рекурентна формула?

Давайте уявімо, що ми обчислюємо просте арифметичне середнє по m величинам:

$$\underline{Q}_m = \frac{1}{m}\varepsilon_m + \frac{1}{m}\varepsilon_{m-1} + \frac{1}{m}\varepsilon_{m-2} + \ldots + \frac{1}{m}\varepsilon_1$$

Можна помітити, що

$$m \cdot \underline{Q}_m - \varepsilon_m = \varepsilon_{m-1} + \varepsilon_{m-2} + \ldots + \varepsilon_1$$

$$\underline{Q}_{m-1} = \frac{1}{m-1} \cdot \left[m \cdot \underline{Q}_m - \varepsilon_m\right] = \frac{1}{m-1}\underline{Q}_m - \frac{1}{m-1}\varepsilon_m$$

звідки

$$\underline{Q}_m = \frac{m-1}{m} \cdot \underline{Q}_{m-1} + \frac{m-1}{m} \cdot \frac{1}{m-1}\varepsilon_m$$

$$\underline{Q}_m = \frac{1}{m}\varepsilon_m + \left(1 - \frac{1}{m}\right)\underline{Q}_{m-1}.$$

Ось формула рекурентного перерахунку середнього арифметичного з появою нового доданку ε_m. Експоненціальна ковзна середня працює за тим же принципом, тільки замість кроку $1/m$, що змінюється, береться постійна величина лямбда:

$$\underline{Q}_m = \lambda\varepsilon_m + (1 - \lambda)\,\underline{Q}_{m-1}.$$

Якщо розписати цю формулу, то вона буде еквівалентна наступній сумі:

$$\underline{Q}_m = \lambda\varepsilon_m + (1 - \lambda)\,\lambda\varepsilon_{m-1} + (1 - \lambda)^2\,\lambda\varepsilon_{m-2} + \ldots .$$

Ми тут бачимо експоненційно спадні коефіцієнти при $\{\varepsilon_k\}$. Значення параметру λ прийнято підбирати відповідно до правила:

$$\lambda = \frac{2}{N+1},$$

де N – інтервал згладжування.

Наприклад, за $N = 99$ отримуємо $\lambda = 0{,}02$.

Я думаю, що тепер вам має бути все зрозумілим у роботі алгоритму стохастичного градієнта. Останнє, що тут важливо відзначити, що є інші варіанти обчислення псевдоградієнту. Ми говорили, що достатньо брати по одному випадковому спостереженню та обчислювати поточний напрямок руху. Однак, у цьому випадку збіжність алгоритму нагадуватиме броунівський рух (випадкове блукання). Щоб якось згладити цей тренд, можна брати не одне, а кілька спостережень на кожній ітерації, їх підсумовувати та отримувати точніші напрямки руху. Це дещо збільшить обсяг обчислень, але у ряді завдань дає кращі результати швидкості збіжності, ніж класичний *SGD* із одним спостереженням. Наприклад, так робиться при навчанні нейронних мереж, коли градієнти обчислюються за пакетами спостережень (міні-батч). Це також буде алгоритм *SGD*.

2.5. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Дайте пояснення алгоритму класифікації образів за допомогою моделі $a(x, w)$.
2. Приведіть графік функції втрат шматково-лінійну (SVM).
3. Приведіть графік функції втрат експонентну (AdaBoost).
4. Дайте пояснення градієнтним алгоритмам.
5. Дайте пояснення стохастичному градієнтному спуску (SGD).

2.6. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.
2. Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale FPGA-based convolutional networks. In R. Bekkerman, M. Bilenko, and J. Langford, editors, Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press.
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

Лекція №3. Формула Байєса

Результати навчання, які формуються:

ДРН4: Використовувати методи добування знань, кластеризації та класифікації.

3.1. Байєсівський висновок. Наївна байєсівська класифікація

На цьому занятті ми ближче познайомимося з формулою Байєса, яку у найпростішому варіанті для двох випадкових подій A і B можна записати у вигляді:

$$P(A|B) = \frac{P(A)\cdot P(B|A)}{P(B)}$$

Припустимо, що ми розглядаємо деяке завдання класифікації. Для простоти, бінарну (двокласову): y O{-1, +1}.

І, звичайно ж, є набір даних навчальної вибірки:

$$X^l = (x_i, y_i)_{i=1}^{l} \sim p(x, y).$$

Ці дані підпорядковуються певній спільній щільності розподілу ймовірностей (ПРВ) $p(x, y)$. Умовно вона повністю зображена нижче у тривимірному вигляді на рис. 3.1:

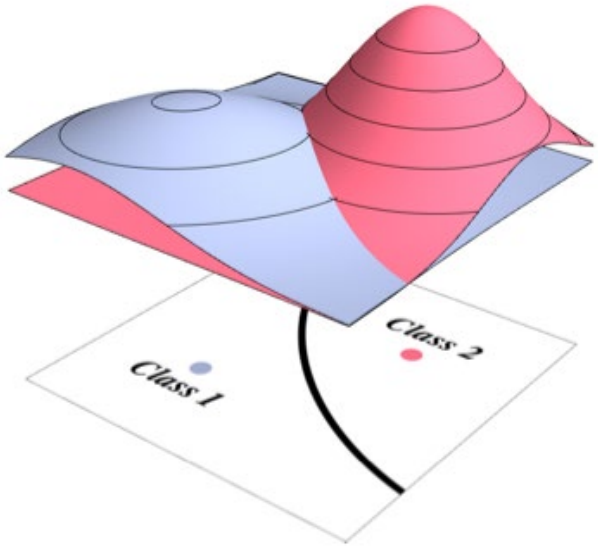


Рис. 3.1

Наше завдання віднести кожен конкретний вхідний вектор (образ) x до одного із двох класів y O{-1, +1}.

Як це правильно зробити? Так, віднести вектор x до того класу, ймовірність якого виявиться найбільшою. Тобто нам потрібно знайти ймовірності для класів y O{-1, +1}, а потім вибрати найбільше значення:

$P(y \mid x) = ?$

Використовуючи теорему Байєса, цей вираз можна розписати, таким чином:

$$P\,(y \mid x) = \frac{P(y) \cdot p(x \mid y)}{p(x)}$$

Я тут через велику букву P позначив ймовірність події, а через малу букву p – ПРВ.

У цій формулі множник $P(y)$ – це апріорна ймовірність появи класу y (частка відповідного класу у навчальній вибірці); $p(x \mid y)$ - Розподіл образів x для певного класу y. На рисунку образно розподіл першого класу показано блакитним графіком, а другого класу – червоним графіком. Зрештою, $p(x)$- Розподіл образів всіх класів безвідносно до їх приналежності тому чи іншому класу.

Припустимо, ми знаємо величини $P(y)$ та розподілу $p(x \mid y)$. Тоді, правило вибору класу (модель класифікації) можна записати, таким чином:

$a(x) = arg\, arg\, max_{y \in Y} P(y|x) = arg\, arg\, max_{y \in Y} P(y) \cdot p(x \mid y)$

Зверніть увагу, я тут відкинув розподіл $p(x)$, оскільки воно не залежить від параметра y, отже, не впливає положення точки максимуму.

Ось так, з позиції теореми Байєса ставиться завдання класифікації в машинному навчанні. Причому кількість класів може бути не два, а скільки завгодно, обмежень тут ніяких немає.

Оптимальний байєсівський класифікатор. Взагалі, на практиці до цієї моделі додають ще множники $\lambda_y \geq 0$, які визначають величину штрафу за неправильної класифікації для кожного класу y:

$a(x) = arg\, arg\, max_{y \in Y} \lambda_y \cdot P(y) \cdot p(x|y)$

Існує теорема, яка доводить, що в цьому випадку мінімізуватимуться середні помилки, тобто, функціонал виду:

$\underline{R}(a) = \sum_{y \in Y} \lambda_y \int [a(x) \neq y] \cdot p(x, y) dx$

Мінімізація цього критерію відповідає оптимальному **байєсівському класифікатору**.

За цим посиланням ви також знайдете приклад простої байєсівської класифікації чоловіків та жінок за двома ознаками: висота та вага, певній спільній щільності розподілу ймовірностей (ПРВ)

Виходить, що якби ми знали спільну ПРВ $p(x, y)$, то завдання класифікації в машинному навчанні було б повністю вирішено. Вся проблема полягає в тому, що ця ПРВ невідома і нам її доводиться якось оцінювати. Або ж, вирішити простішу задачу, знайти оцінки $\hat{P}(y)$, $\hat{p}(x|y)$ з навчальної вибірки.

Звичайно, після того, як у класифікатор байєсівского підставляються оцінки ПРВ, то він перестає бути оптимальним. Тому оптимальність тут радше представляє теоретичний інтерес. Насправді, реальні класифікатори можуть лише наближатися до оптимального.

Наївний байєсівський класифікатор. Головна проблема у реалізації байєсівського класифікатора – це оцінити умовні ПРВ $\hat{p}(x|y)$. І, в загальному вигляді, це завдання набагато складніше, ніж побудова класифікатора з позиції

вибору параметричної функції та подальшого пошуку вектору параметрів θ: $a(x) = g(x, \theta)$.

Тобто так, як це ми робили із самого початку. Отже, байєсівський (імовірнісний) підхід до завдань машинного навчання – це лише цікавий теоретичний трюк? Не зовсім. По-перше, як ми вже бачили, він дає найкраще розуміння роботи алгоритмів машинного навчання. А по-друге, ми можемо спростити завдання оцінки багатовимірної умовної ПРВ $\hat{p}(x|y)$, поклавши, що всі ознаки: $x = [\xi_1, \xi_2, \xi_n]^T$ незалежні між собою.

І тут багатовимірна ПРВ розписується через добуток відповідних одновимірних ПРВ:

$$p(x \mid y) = p(\xi_1 \mid y) \cdot p(\xi_2 \mid y) \cdot \cdots \cdot p(\xi_n \mid y) = \prod_{i=1}^{n} p(\xi_i|y)$$

Таке, взагалі кажучи, сильне припущення про незалежність ознак, призводить до алгоритму, відомого під назвою наївний байєсівський класифікатор: $a(x) = arg\, arg\, max_{y \in Y} \lambda_y \hat{P}(y) \prod_{i=1}^{n} \hat{p}(\xi_i|y)$

Або, еквівалентний висновок часто роблять за логарифмом від добутку величин (уникають добутку): $a(x) = arg\, arg\, max_{y \in Y} \left(ln \lambda_y \hat{P}(y) + \sum_{i=1}^{n} ln\, \hat{p}(\xi_i|y) \right)$.

Як бачите, ми тут оперуємо лише одновимірними ПРВ і це спрощує завдання, так як відновити одномірну густину набагато простіше, ніж багатовимірну.

І ще одне важливе зауваження. Часто в сторонніх бібліотеках алгоритм наївного класифікатора байєсівського реалізують з використанням гауссівських ПРВ $p(\xi \mid y)$.

Звичайно, це досить поширений випадок, але не завжди ознаки підкоряються нормальному закону розподілу. Якщо у вашому завданні суттєво інша ПРВ ознак, то бібліотечні реалізації можуть призводити до незадовільних результатів. Звертайте на це увагу. Інакше, ви ризикуєте зробити висновок про непридатність наївного байєсу для вирішення вашого завдання, тоді як насправді потрібно просто вибрати інший вид ПРВ.

3.2. Гаусівський байєсівський класифікатор

Давайте уявімо, що безліч об'єктів навчальної вибірки підпорядковується гаусівському (нормальному) закону розподілу:

$$X^l = \left\{ (x_i, y_i)_{i=1}^{l} \right\} \sim N(m, \sigma)$$

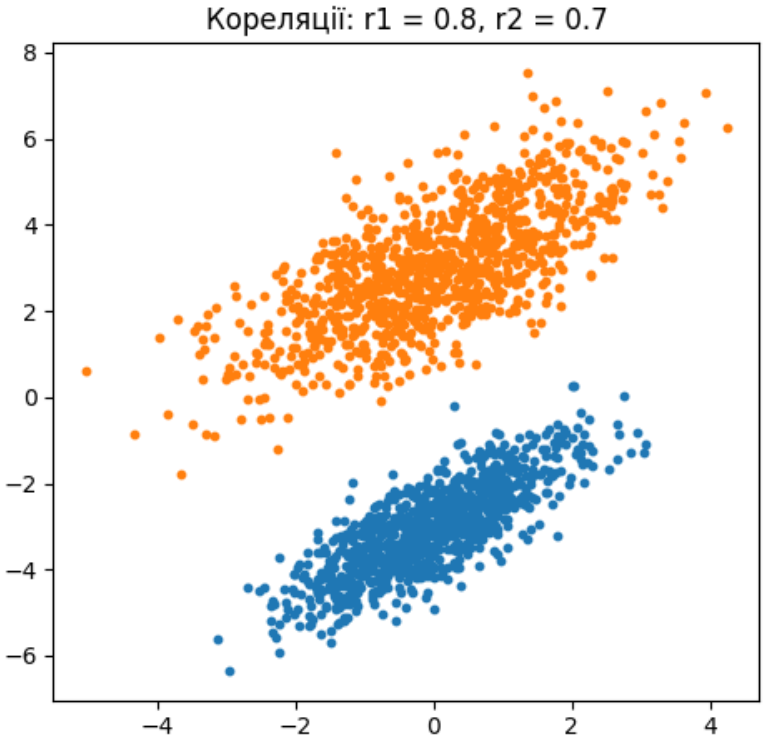


Рис. 3.2

Тоді, відповідно до теореми Байєса, алгоритми машинного виведення слід будувати, виходячи з формули:

$$P(y \mid x) = \frac{P(y) \cdot p\,(x \mid y)}{p(x)}$$

Тут $P(y)$ – (як і раніше) апріорна ймовірність появи класу y; $p(x|y)$ – умовна щільність розподілу (функція правдоподібності) образів x для заданого класу y.

Оскільки ми вважаємо, що ці вибірки підпорядковуються гауссівському розподілу, то формально:

$$p(y \mid x) = \frac{1}{(2\pi \cdot det\Sigma_y)^{\frac{n}{2}}} \cdot exp\left\{-\frac{1}{2}(x-\mu_y)^T \Sigma_y^{-1}(x-\mu_y)\right\}$$

Тут $\mu_y = \left[\mu_y^1, \mu_y^2, \ldots, \mu_y^n\right]^T$ – вектор математичних очікувань образів x, що належать класу y; $\sum_y$ – коваріаційна матриця образів x, які відносяться до класу y:

$$\sum_y = E\left\{\left[\left(x^1-\mu_y\right)\left(x^2-\mu_y\right)\ldots\left(x^n-\mu_y\right)\right]\cdot\left[\left(x^1-\mu_y\right),\left(x^2-\mu_y\right),\ldots\left(x^n-\mu_y\right)\right]\right\}$$

де $E\{\ldots\}$ – оператор математичного очікування.

Алгоритм ухвалення рішення (класифікації) для багатовимірної гаусівської ПРВ записується так само, як і для оптимального байєсівського класифікатора: $a(x) = \lambda_y \cdot P(y) \cdot p(x \mid y)$

Нагадаю, що тут $\lambda_y \geq 0$ – штрафи, які ми накладаємо на неправильну класифікацію моделлю класу y.

Наприклад, коли ми розглядали завдання наївного байєсівського висновку, то вважали, що діагональна коваріаційна матриця з дисперсіями по головній діагоналі:

$$\sum_y = [\sigma_{x1}^2\, 0 \ldots 0 \;\; 0\, \sigma_{x2}^2 \ldots 0 \;\; \ldots \ldots \ldots \ldots \;\; 0\, 0 \ldots \sigma_{xn}^2 \;\;].$$

Для гаусівського розподілу це автоматично означає незалежність випадкових величин (у разі незалежність ознак):

$$p(y \mid x) = \frac{1}{(2\pi \cdot det\Sigma_y)^{\frac{n}{2}}} \cdot exp\left\{-\sum_{i=1}^{n} \frac{1}{\sigma_{xi}^2}(x - m_y)^2\right\} = \prod_{i=1}^{n} \left(x^i|y\right)$$

Але, як ви розумієте, це дуже сильне припущення, що межує з наївністю (звідси назва). Ознаки завдання машинного навчання часто виявляються певною мірою лінійно залежними. І, як знаємо з теорії ймовірностей, ступінь лінійної залежності двох будь-яких випадкових величин визначається коваріацією:

$$B\ (x,\ y)\ =\ E\{(x\ -\ m_x)\cdot(y\ -\ m_y)\}\ =\ \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x - m_x) \cdot \left(y - m_y\right) \cdot p(x, y)dxdy$$

або, нормованим коефіцієнтом – кореляцією:

$$R(x, y) = \frac{B(x,y)}{\sigma_x \cdot \sigma_y}, \quad -1 \leq R(x, y) \leq 1$$

Якщо значення коефіцієнта кореляції дорівнює нулю, то гаусівських випадкових величин це повну незалежність, а інших розподілів – лінійну незалежність.

Якщо коефіцієнт кореляції дорівнює 1 або -1, то СВ повністю лінійно залежні і одну величину можна обчислити по іншій. У більшості практичних завдань коефіцієнт кореляції по модулю лежить між нулем і одиницею, тобто величини лінійно незалежні, але лише певною мірою. У цьому випадку застосування наївного класифікатора байєсівського може бути під питанням, особливо, якщо коефіцієнти кореляції по модулю близькі до одиниці.

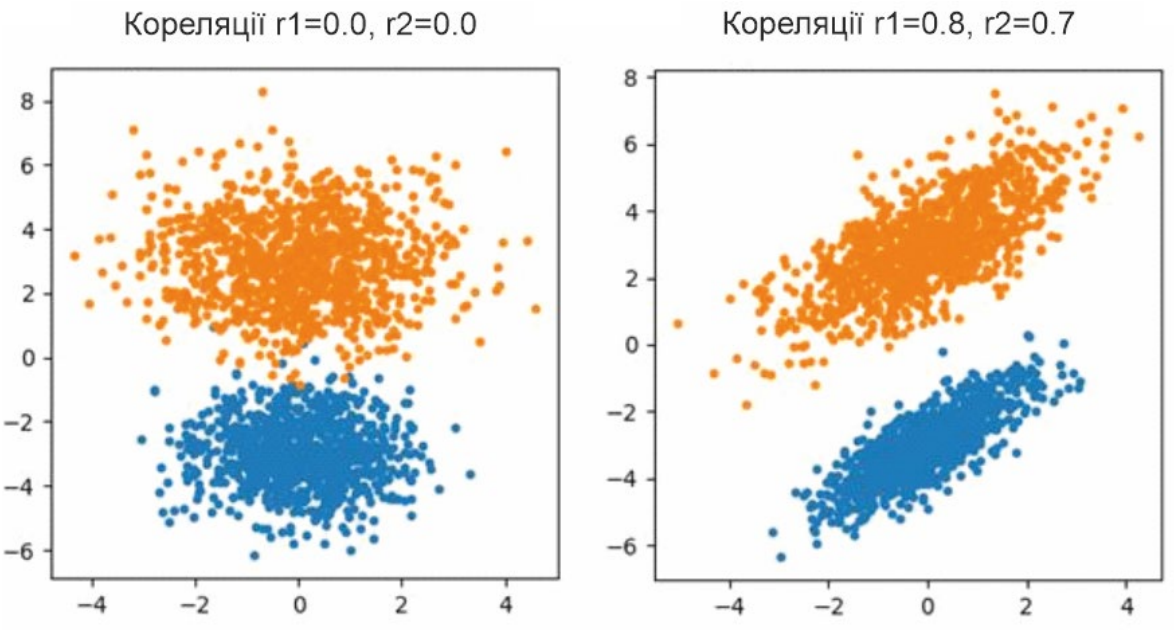


Рис. 3.3

Добре, але тоді питається, як ми можемо на практиці врахувати підступні зв'язки між ознаками і побудувати не наївний, а повноцінний класифікатор Байєса? Хороша новина, що у разі багатовимірного розподілу гаусівського це зробити відносно просто. Як ви вже здогадалися, для цього нам достатньо по навчальній вибірці побудувати оцінки МО та коваріаційних матриць окремо для кожного класу: $\hat{\mu}_y$, $\hat{\Sigma}_y$, y O Y

Формули для обчислення цих оцінок прості і загалом очевидні:

$$\hat{\mu}_y = \frac{1}{l_y}\sum_{i:y_i=y} x_i.$$

$$\Sigma_y = \frac{1}{l_y}\sum_{i:y_i=y} (x_i - \hat{\mu}_y)\cdot(x_i - \hat{\mu}_y)^T.$$

Існує теорема, яка доводить, що ці оцінки відповідають оцінкам максимальної правдоподібності для *n*-вимірних гаусівських розподілів. Тобто ці оцінки адекватні і, загалом, нічого кращого тут ми придумати не можемо.

Звичайно, щоб можна було довіряти цим оцінкам, кількість об'єктів кожного класу має бути якнайбільше. Інакше просто не зберемо достатньої статистики. Вважається, що мінімум це 100 об'єктів одного конкретного класу. Але, бажано мати від 1000 і більше.

Щоб усе це краще зрозуміти, давайте застосуємо цей підхід (гаусівський байєсівський класифікатор) до змодельованого завдання бінарної класифікації, в якій навчальна вибірка згенерована на основі нормального двомірного розподілу, з наступними параметрами:

$$r_1 = 0{,}8;\ \sigma_{x1}^2 = 1{,}0;\ \mu_{y1} = [0, -3]^T;\ \Sigma_{y1} = \begin{pmatrix} \sigma_{x1}^2 & \sigma_{x1}^2 \cdot r_1 \\ \sigma_{x1}^2 \cdot r_1 & \sigma_{x1}^2 \end{pmatrix}$$

$$r_2 = 0{,}7;\ \sigma_{x2}^2 = 2{,}0;\ \mu_{y2} = [0, -3]^T;\ \Sigma_{y1} = \begin{pmatrix} \sigma_{x2}^2 & \sigma_{x2}^2 \cdot r_2 \\ \sigma_{x2}^2 \cdot r_2 & \sigma_{x2}^2 \end{pmatrix}$$

Усього ми генеруємо за $N = 1000$ образів вибірки кожного класу. За цими даними обчислюємо оцінки МО та коваріаційних матриць: $\hat{\mu}_y$, $\hat{\Sigma}_y$, y O Y.

А потім застосовуємо алгоритм гаусівського байєсівського класифікатора:

$$a(x)=\left(ln\left(\lambda_y\hat{P}(y)\right)-\frac{1}{2}(x-\hat{\mu}_y)^T\cdot\hat{\Sigma}_y^{-1}(x-\hat{\mu}_y)-\frac{1}{2}ln\,det\hat{\Sigma}_y\right)$$

Тут від умовної багатовимірної ПРВ було взято натуральний логарифм. (**Додаток 1.**)

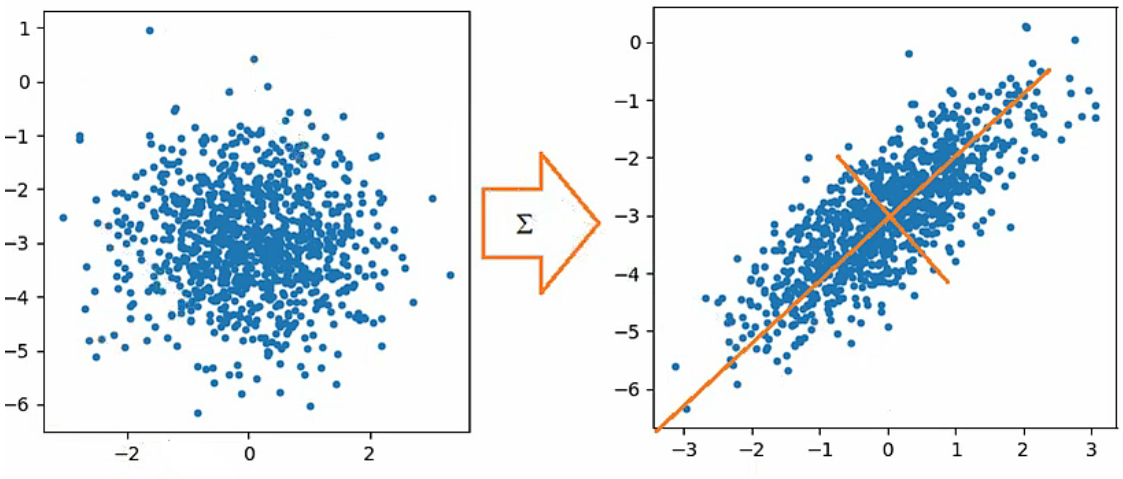


Рис. 3.4

Якщо ще раз подивитися на розподіл точок для двовимірного розподілу гаусівського з кореляцією 0,7, то можна побачити, так званий, їх еліпс розсіювання. І не більше цього еліпса можна назвати дві основні осі, вздовж яких розподілені точки:

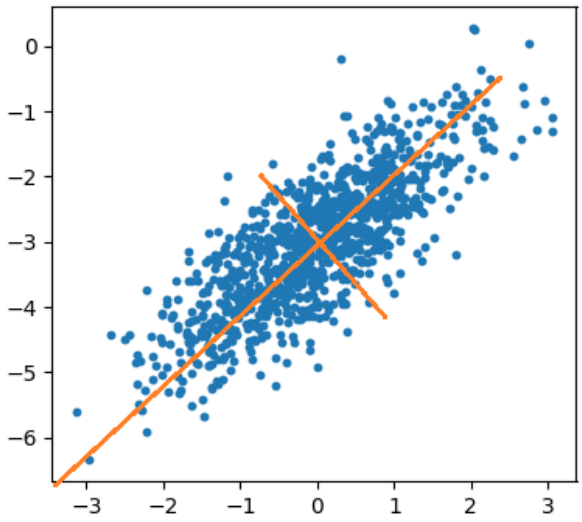


Рис. 3.5

Так ось, коваріаційну матрицю можна представити у вигляді так званого спектрального розкладання:

$$\sum = VSV^T,$$

Де V- $n \times n$ матриця, що складається із власних векторів матриці $\sum$;

S – діагональна матриця з дисперсіями (власними числами) по головній діагоналі.

Фактично, коваріаційна матриця визначає лінійні перетворення рівномірно розподіленої множини точок в корельовану множину точок:

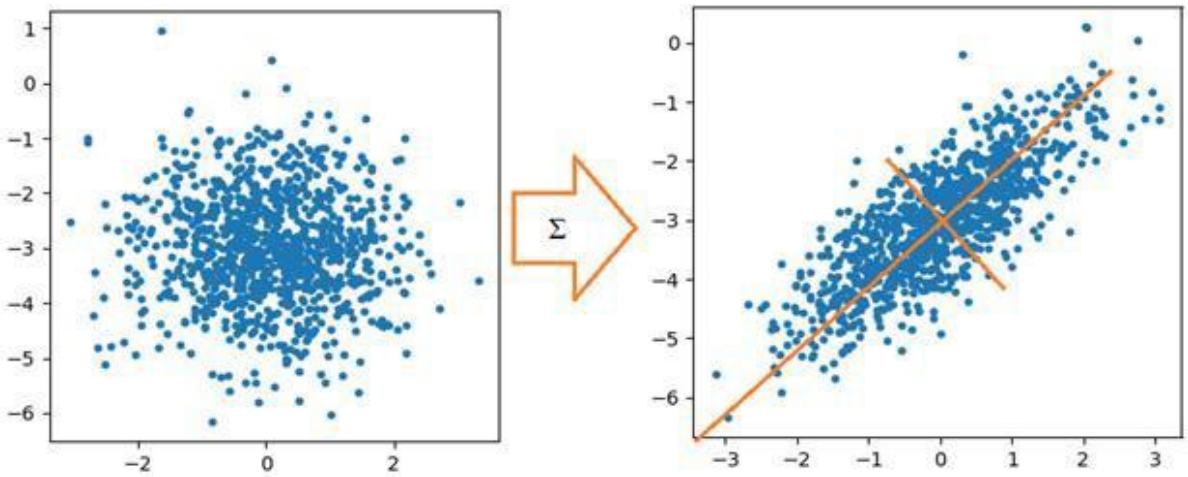


Рис. 3.6

Тобто, прописуючи зворотну матрицю в класифікаторі, ми, по суті, переходимо в новий простір (нову систему координат), де ознаки не корельовані і там, у цьому просторі реалізуємо звичайний, наївний байєс (обчислюємо суму). Ось у чому основна суть коваріаційної матриці в гаусівському байєсівському класифікаторі.

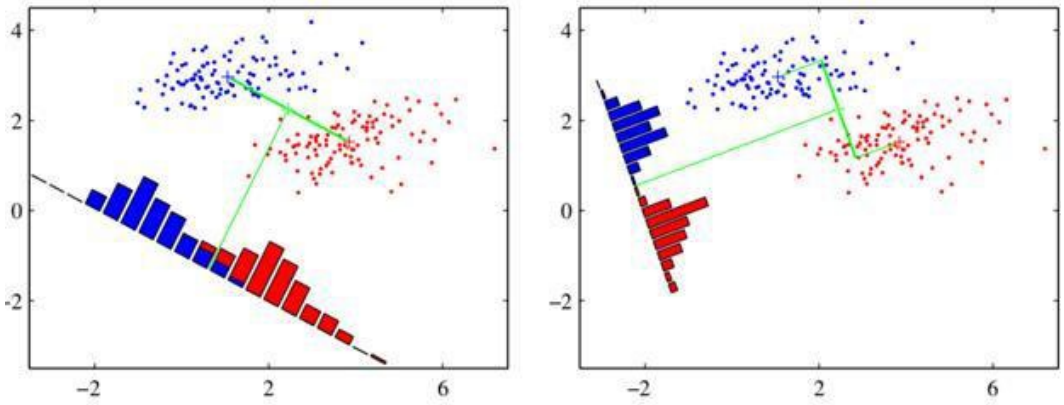


Рис. 3.7

Докладніше про спектральне розкладання, власні вектори та числа, ми з вами ще говоритимемо на майбутніх заняттях. Але, я думаю, що загальний принцип роботи байєсівських класифікаторів при гаусівських розподілах вам зрозумілий

3.3. Лінійний дискримінант Фішера

Досить цікавою окремою нагодою гаусівського байєсівського класифікатора, який ми розглянули на попередньому занятті, є лінійний дискримінант Фішера. Він будується виходячи з припущення про рівність всіх коваріаційних матриць класів:

$$\Sigma = \Sigma_{y1} = \Sigma_{y1} = \cdots = \Sigma_{yk}$$

Нагадаю, що модель будується за правилом:

$$a(x) = \lambda_y \cdot P(y) \cdot p(x \mid y)$$

де $\lambda \geq 0$ – штрафи за неправильну класифікацію класу y; $P(y)$ – апріорна ймовірність появи класу y;

$$p(y \mid x) = \frac{1}{(2\pi \cdot det\Sigma_y)^{\frac{n}{2}}} \cdot exp\left\{-\frac{1}{2}(x-\mu_y)^T \Sigma_y^{-1}(x-\mu_y)\right\}$$

у разі гаусового розподілу ознак вектору x.

Так от, при рівності всіх коваріаційних матриць різних класів, нам достатньо оцінити вектори математичних очікувань (МО): $\hat{\mu}_y = \frac{1}{l_y}\sum_{i,y\in Y} x_i$, $y \mathrm{O} Y$ і загальну коваріаційну матрицю вже по всіх об'єктах навчальної вибірки:

$$\hat{\Sigma} = \frac{1}{l}\sum_{i=1}^{l} (x_i - \hat{\mu}_{yi}) \cdot (x_i - \hat{\mu}_{yi})^T$$

Тобто, у нас МО різні (свої) для кожного класу, а матриця коваріаційна загальна. У результаті класифікатор набуває вигляду (з використанням логарифмічної функції):

$$a(x) = \left(ln\left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}\left(x-\hat{\mu}_y\right)^T \cdot \hat{\Sigma}^{-1} \cdot \left(x-\hat{\mu}_y\right)\right)$$

Зверніть увагу, ми тут прибрали доданок $-\frac{1}{2} ln\, det\hat{\Sigma}_y$, що є у гауссовском байєсівському класифікаторі, т.к. при рівності коваріаційних матриць, ця величина та сама для всіх класів, отже, її можна відкинути.

Далі вираз під argmax можна розписати, таким чином:

$$ln\, ln\left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}x^T\hat{\Sigma}^{-1}\left(x-\hat{\mu}_y\right) + \frac{1}{2}\hat{\mu}_y{}^T\hat{\Sigma}^{-1}\left(x-\hat{\mu}_y\right) =$$

$$= ln\, ln\left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}x^T\hat{\Sigma}^{-1}x + \frac{1}{2}x^T\hat{\Sigma}^{-1}\hat{\mu}_y + \frac{1}{2}\hat{\mu}_y{}^T\hat{\Sigma}^{-1}x_{\smile 0} - \frac{1}{2}\hat{\mu}_y{}^T\hat{\Sigma}^{-1}\hat{\mu}_y =$$

$$= ln\, ln\left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}\hat{\mu}_y{}^T\hat{\Sigma}^{-1}\hat{\mu}_y + x^T\hat{\Sigma}^{-1}\hat{\mu}_y - \frac{1}{2}x^T\hat{\Sigma}^{-1}x$$

Останній доданок $\frac{1}{2}x^T\hat{\Sigma}_x^{-1}$ не залежить від номера класу, отже, його також можна не враховувати.

Отримуємо класифікатор у вигляді:

$$a(x) = \left(ln\, ln\left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}\hat{\mu}_y{}^T\hat{\Sigma}^{-1}\hat{\mu}_y + x^T\hat{\Sigma}^{-1}\hat{\mu}_y\right)$$

Якщо тепер для кожного класу y ввести позначення:

$$\beta_y = ln\, ln\left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}\hat{\mu}_y{}^T\hat{\Sigma}^{-1}\hat{\mu}_y$$

$$\alpha_y = \hat{\Sigma}^{-1}\hat{\mu}_y$$

тоді класифікатор набуде вигляду:

$$a(x) = \left(ln\ ln\ \left(\lambda_y \hat{P}(y)\right) - \frac{1}{2}\hat{\mu}_y{}^T \hat{\Sigma}^{-1}\hat{\mu}_{y_{\omega_0}} + x^T \hat{\Sigma}^{-1}\hat{\mu}_{y_{\omega_0}} \right)$$

$$a(x) = \left(x^T \cdot \alpha_y + \beta_y\right).$$

Це вам нічого не нагадує? Так, це звичайний лінійний класифікатор, про який ми з вами раніше вже говорили. Він отримав назву **лінійного дискримінанта Фішера**. На честь вченого, який вперше вирішив це завдання далекого 1936 року $\sum^* = \sum + \tau I$.

Виходить, якщо розподіл безлічі об'єктів різних класів підпорядковуються гаусівському закону і мають однакові матриці коваріації, то оптимальний класифікатор для них буде лінійним (або кусочно-лінійним для безлічі класів). І це зрозуміло. Якщо ми подивимося на межу перетину двох нормальних двовимірних ПРВ з однаковими матрицями коваріації, то побачимо, що лінія з рівними ймовірностями класів буде лінійною:

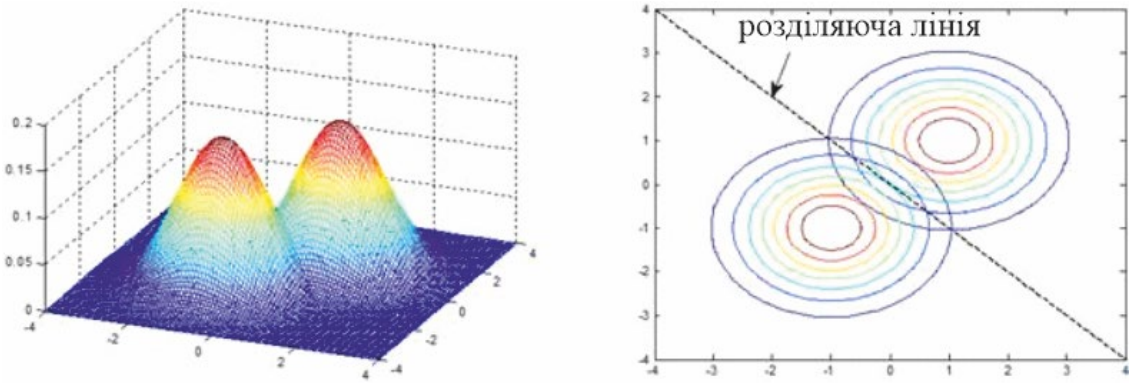


Рис. 3.8

Навіть якщо ми вибиратимемо різні штрафи, або класи матимуть різні апріорні ймовірності, то ця лінія просто зміщуватиметься праворуч або ліворуч – у бік від класу з більшим штрафом та/або більшою апріорною ймовірністю. Форма ж лінії, що розділяє, або гіперплощини залишатиметься незмінною. Тому алгоритм класифікації за таких вихідних даних завжди лінійний.

Основною перевагою лінійного дискримінанта Фішера в простоті його реалізації (малого обсягу обчислень) і, крім того, можливість обчислювати матрицю коваріаційну по всіх об'єктах вибірки, а не по об'єктах одного окремого класу.

Цей підхід допомагає, коли вибірка сильно не збалансована і представників окремого класу може бути недостатньо для хорошої оцінки матриці коварії. У дискримінанті Фішера така проблема, як правило, відсутня.

Проте, і в гауссівському байєсівському класифікаторі, і в лінійному дискримінанті Фішера залишається одна головна проблема зі зверненням матриці підступів $\sum^{-1}$.

Як ви вже знаєте, через лінійні або майже лінійні залежності між ознаками ця матриця стає погано обумовленою і зворотна матриця може навіть не існувати. Вирішити цю проблему можна двома поширеними способами. Один ми з вами зовсім недавно вже розглянули – це вважати всі ознаки незалежними, тоді матриця підступів вироджується у діагональну матрицю, яка завжди добре звертається. У підсумку ми приходимо до наївного байєсівського класифікатора.

У другому способі додають до елементів головної діагоналі коваріаційної матриці невеликі ненульові значення: $\sum^* = \sum + \tau I$, де $\tau > 0$ - невелике значення; I – одинична матриця.

Цей підхід відповідає $L2$-регуляризації і, звичайно ж, у байєсівських класифікаторах його також можна використовувати.

3.4. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Приведіть формулу Байєса.
2. Приведіть приклад простої байєсівської класифікації.
3. Дайте визначення наївного байєсівського класифікатора.
4. Дайте визначення гаусівського байєсівського класифікатора
5. Дайте визначення лінійного дискримінанта Фішера

3.5. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.
2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA

Лекція №4. Методи SVM, PCA

Результати навчання, які формуються:

ДРН1: Розробляти та застосовувати методи, алгоритми та інструменти для навчання нейронних мереж.

ДРН5: Вміти створювати ефективні алгоритми для обчислювальних задач системного аналізу та систем підтримки прийняття рішень.

ДРН6: Застосовувати знання машинного навчання в умовах слабо структурованих даних різної природи.

4.1. Введення метод опорних векторів (SVM)

Давайте знову повернемося до розгляду завдання бінарної класифікації образів з позиції гіперплощини, що розділяє. Для простоти припустимо, що у нас двовимірний простір ознак. Тоді кожен образ класу можна уявити крапкою на площині. Нехай образи навчальної вибірки розподіляються, таким чином:

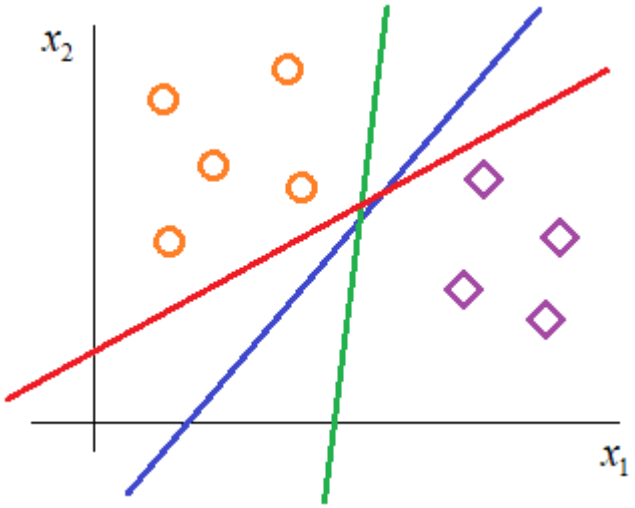


Рис. 4.1

Як бачите, для нашого прикладу можна побудувати безліч різних ліній, що розділяють (в загальному випадку, гіперплощин) так, що кожна з них буде коректною відокремлювати один клас від іншого. І тут постає питання, який поділ кращий?

У машинному навчанні ми виходимо з того, що модель, навчена на деякій вибірці, має добре працювати з іншими довільними наборами того ж розподілу. Тобто модель повинна мати хороші узагальнюючі здібності (не бути надто перенавченою). Якщо з цих позицій подивитися на проведені лінії розділення, то чисто інтуїтивно я вибрав би синю. Чому її? Дивіться, червона і зелена лінії насправді роблять додаткові припущення про розподіл образів обох класів:

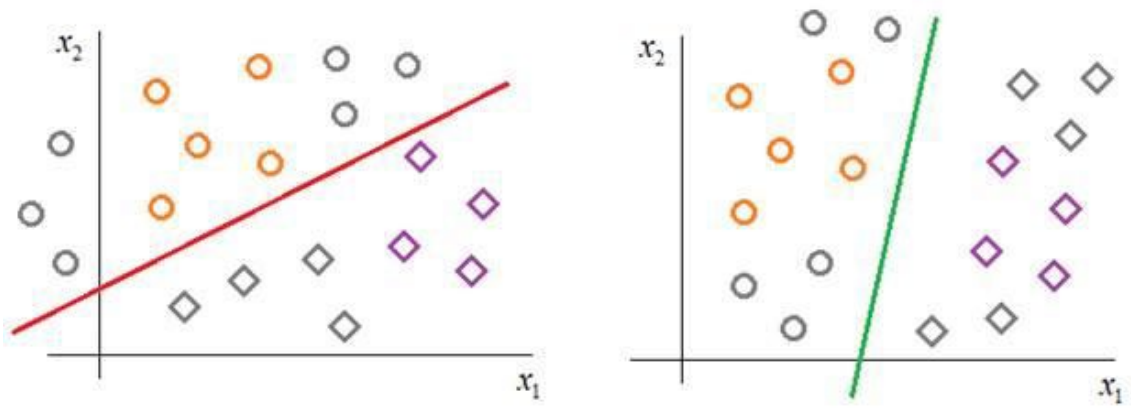


Рис. 4.2

Червона «думає», що образи розподілені трохи горизонтально, а зелена, що образи йдуть вертикальніше. Звичайно, і синя лінія "робить" своє припущення, але вона суто візуально більшою мірою зберігає вихідний розподіл. Ось ця ідея проведення роздільної гіперплощини, яка орієнтувалася тільки на розподіл навчальної вибірки і по можливості не робила б додаткових припущень про розподіл образів у класах, покладена в основу методу опорних векторів (Support Vector Machine – *SVM*).

Давайте тепер суто інтуїтивне міркування формалізуємо лише на рівні математики. І відповімо на перше запитання, що ж це за роздільна гіперплощина, яка «робить» мінімальні припущення про розподіл класів і, тим самим, призводить до кращої узагальнюючої здатності алгоритму класифікації? З точки зору *SVM*, оптимальна роздільна гіперплощина – це та, яка утворює найбільш широку смугу між об'єктами двох класів.

При цьому сама роздільна гіперплощина точно проходитиме посередині цієї смуги:

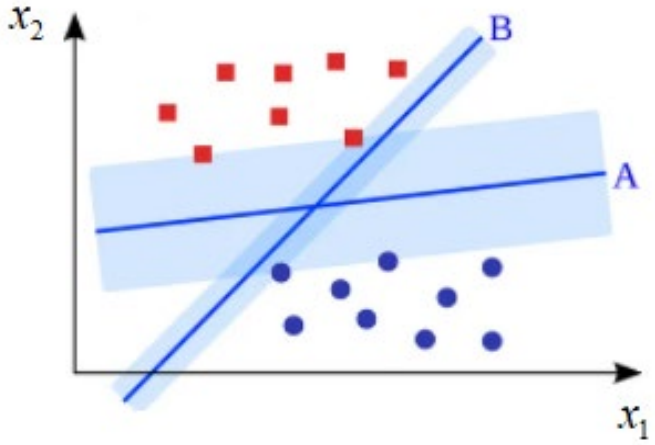


Рис. 4.3

Чому так? Тому що, чим ширша смуга, тим надійніше класифікатор відокремлюватиме образи одного класу від іншого.

Щоб описати цю ідею лише на рівні математики, ми спочатку маємо задати модель класифікатора, яка, фактично, визначає рівняння гіперплощини в просторі ознак.

Ми виберемо найпростішу лінійну модель виду:

$$a(x) = sign\ (w^T x - b) = sign\ (\langle w, x\rangle - b) = sign(w^* x - b).$$

Запишемо тут трьома способами, але всі вони означають лінійну комбінацію вектору параметрів з чином x плюс зміщення $-b$.

На виході модель видає значення: $a(x)$ О {-1; +1}

Далі, ми припустимо, що наша навчальна вибірка складається з лінійно роздільних образів (потім, цей випадок узагальнимо на лінійно нероздільний). Тоді ширина смуги визначатиметься розташуванням граничних векторів x у просторі ознак:

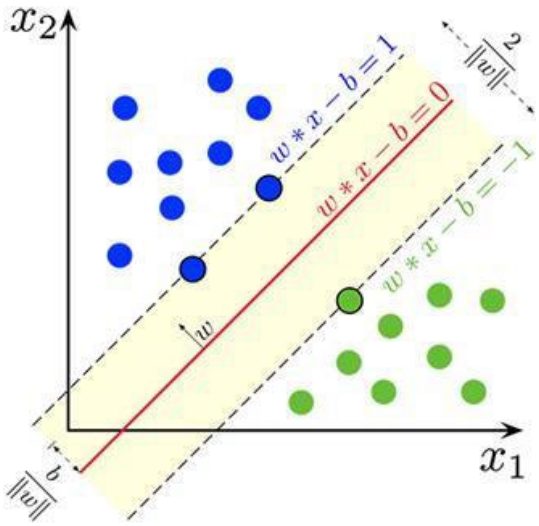


Рис. 4.4

Якщо взяти будь-які два образи різних класів, розташовані найближче до межі, що розділяє (тобто, що лежать на межі смуги), то ширину смуги можна обчислити як проекцію вектору $x_+ - x_-$ на вектор ω:

$$\langle w \circ x_+ - x_+\rangle = w^T (x_+ - x_+) = |w| \cdot |x_+ - x_+| \cdot cos\alpha$$

ширина смуги

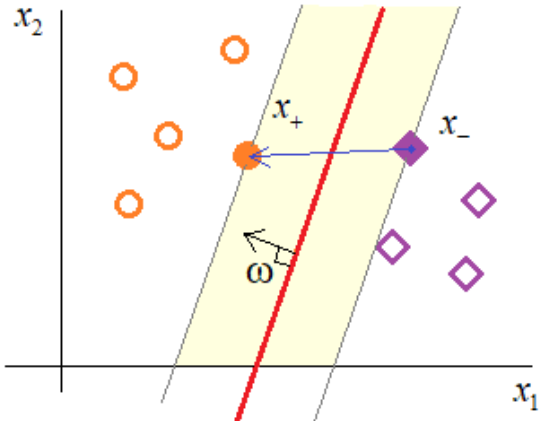


Рис. 4.5

Тобто отримуємо ширину, помножену на довжину вектору коефіцієнтів?

Тому, звичайно, маємо:

$$L = |x_+ - x_+| \cdot \cos\alpha = \frac{\langle w, x_+ - x_+ \rangle}{|w|} \to \max.$$

І цю величину слід максимізувати.

Але для зручності *SVM* в знаменнику записують не довжину вектору ω, а квадрат його норми: $||w||^2 = w^T \cdot w$ отримуємо вираз для максимізації ширини: $L = \frac{\langle w, x_+ - x_- \rangle}{||w||^2} \to \max$.

Принципово така заміна картини не змінює, зате ми точно не зіштовхуватимемося з обчисленнями квадратних коренів при вирішенні оптимізаційної задачі.

Зрештою, зробимо ще одне, останнє спрощення. Як ви пам'ятаєте із попередніх занять, у завданнях бінарної класифікації вводиться поняття відступу (margin): $M_i = y_i \cdot a(x_i) = y_i \cdot (\langle w, x_i \rangle - b)$, $i = 1, 2, \ldots, l$

Ця величина характеризує відстань від роздільної гіперплощини до обраного образу.

Причому: $M_i > 0$ при неправильній кваліфікації та $M_i < 0$ при неправильній кваліфікації.

Але, оскільки ми розглядаємо випадок лінійно-роздільних образів, то свідомо існують такі значення w і b, що:

$$M_i = y_i \cdot (\langle w, x_i \rangle - b) > 0, \quad i = 1, 2, \ldots, l.$$

Так ось, нам ніщо не заважає помножити цю величину на кілька: $\alpha > 0$

В результаті отримаємо: $\alpha M_i = y_i \cdot (\langle \alpha w, x_i \rangle - \alpha b) > 0$, $i = 1, 2, \ldots, l$.

Суть оптимізаційної задачі це не змінить. Але ми тепер можемо підібрати таке значення α і змінити параметри: $w = \alpha w$; $b = \alpha b$, щоб для будь-яких образів x_+, x_-, що лежать на межі смуги, величина: $M_i(x_+) = 1$; $M_i(x_-) = 1$.

Строго математично це нормування можна записати, таким чином:

M_i (w, b) =1. Очевидно, що для лінійно-роздільного випадку ми завжди можемо це зробити. Але тоді автоматично виходить, що: $\langle w, x_+ - x_- \rangle$ = $\langle w, x_+ \rangle$ - $\langle w \circ x_- \rangle$ = 1-(-1)=2 і ширина смуги визначатиметься виразом: $L = \frac{2}{||w||^2} \rightarrow$max.

В результаті для лінійно-роздільних образів ми отримуємо наступне оптимізаційне завдання:

$$\{\frac{2}{||w||^2} \rightarrow \max_{w,b} \qquad M_i(w,b) \geq 1\,, \quad i = 1,2,\ldots,l$$

Але її звично зводять до завдання мінімізації:

$$\{\frac{1}{2}||w||^2 \min_{w,b} \qquad M_i(w,b) \geq 1\,, \quad i = 1,2,\ldots,l$$

Тобто нам потрібно знайти такі ω і b, щоб мінімізувати квадратичну норму ваг і разом з тим забезпечити всі відступи більше одиниці, крім тих образів, що лежать безпосередньо на межах смуги (там відступ повинен дорівнювати одиниці).

Це називається **завданням квадратичного програмування**, коли ми мінімізуємо квадрати вагових коефіцієнтів при лінійних обмеженнях нерівностей.

4.2 Метод опорних векторів для лінійно неподільного випадку

Однак, як ви розумієте, у загальному випадку образи в навчальних вибірках рідко бувають роздільними лінійно. Тому при лінійно нероздільній вибірці ми не зможемо знайти параметри ω і b, які задовольняли б лінійним обмеженням на відступи:

$$M_i(w, b) \geq 1, \quad i = 1, 2,\ldots,l$$

Як же бути? Для цього вигадали наступну евристику. Давайте дозволимо класифікатор помилятися на деяку величину (slack variables):

$$\xi_i \geq 0, \quad i = 1, 2,\ldots,l$$

для кожного і-го образу: $M_i(w, b) \geq 1 - \xi_i, \quad i = 1, 2,\ldots,l.$

Величини slack variables ще можна сприймати як штраф за порушення вихідної нерівності. Зрозуміло, якщо все $\xi_i \rightarrow +\infty$, то ми можемо взяти будь-які ваги, наприклад 0 і тоді оптимізаційне завдання буде вирішено. Але, очевидно, це не те, що нам потрібне. Звичайно, ми дозволяємо помилятися, але величина цієї помилки має бути якомога меншою, тобто, нам потрібно знайти такі ω і b, щоб $\xi_i \rightarrow 0, \quad i = 1, 2,\ldots,l.$

Цю умову ми можемо врахувати в алгоритмі мінімізації, записавши її наступним чином:

$$\{\frac{1}{2C}||w||^2 + C \cdot \sum_{i=1}^{l} \xi_i \rightarrow \min_{w,b,\xi} M_i(w,b) \geq 1 - \xi_i\,,\ i = 1,2,\ldots,l\ \xi_i \geq 0\,,\ i =$$

$$= 1, 2, \dots, l$$

де C – гіперпараметр, що визначає ступінь мінімізації величин $\{\xi_i\}$.

Все ми з вами отримали еквівалентне завдання оптимізації для загального випадку лінійно нероздільної вибірки.

Тепер питання, як її вирішити? Дивіться, ми можемо останні дві нерівності у системі переписати у вигляді:

$$\{\xi_i \geq 1 - M_i(w,b)\,, \quad i = 1,2,\dots,l \; \xi_i \geq 0\,, \quad i = 1,2,\dots,l$$

І оскільки ми вирішуємо завдання пошуку мінімуму коефіцієнтів ω і b, то в цій нерівності логічно вибрати рівність, тобто:

$$\{\xi_i = 1 - M_i(w,b) \; \xi_i = 0\,, \; L_i(w,\, b) = \max(0,1 - M_i(w,\, b)), \; i = 1, 2,\dots,l$$

Цей вираз ще записують так: $L_i(w,\, b) = (1 - M_i(w,\, b))_+, \; i = 1, 2,\dots,l$

(Позитивне зрізання – все, що менше нуля дорівнює нулю).

В результаті вихідна система стає еквівалентною безумовної задачі мінімізації:

$$C \cdot \sum_{i=1}^{l} \left(1 - M_i(w,b)\right)_+ + \frac{1}{2}||w||^2 \to \min_{w,b}$$

або, при поділі на параметр C, маємо:

$$\sum_{i=1}^{l} \left(1 - M_i(w,b)\right)_+ + \frac{1}{2C}||w||^2 \to \min_{w,b}$$

Що вам нагадує цей вираз? Так, тут у нас першим доданком йде функціонал якості (емпіричний ризик), а друге – це $L2$-регулятор для коефіцієнтів ω. Причому функція втрат тут має такий вигляд (зелений графік) і називається hinge loss.

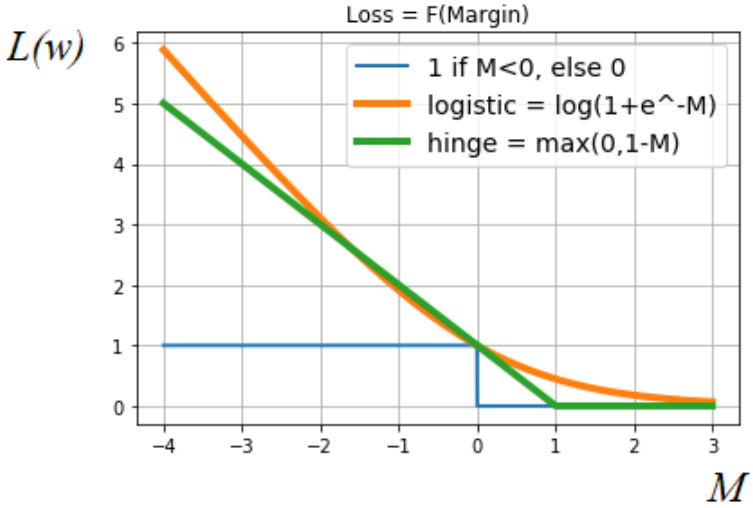


Рис. 4.6

Для порівняння тут також наведено логарифмічні функції втрат, які ми раніше з вами розглядали. Дивіться, що виходить. Фактично, *SVM* – це вирішення оптимізаційної задачі за функції втрат hinge. Вона починає

штрафувати, якщо мінімальний відступ для об'єкта стає менше одиниці. І ніяк не штрафує (нульовий штраф), якщо відступ більший або дорівнює 1. Тобто для hinge loss важлива саме ширина смуги між образами двох класів. На відміну, наприклад, від логістичної функції втрат, яка прагне максимально розвести образи класів щодо гіперплощини, що розділяє, тому що тут штраф поступово зменшується при збільшенні відступу і ніколи не йде в нуль.

Також з методу опорних векторів дуже добре видно геометричне значення $L2$-регулятор. Фактично він тут відповідає за максимізацію ширини смуги між сусідніми образами двох різних класів і це в результаті призводить до кращих узагальнюючих здібностей отриманого класифікатора.

На цьому ми завершимо перше заняття з SVM. Тут ми отримали загальний алгоритм знаходження параметрів лінійної моделі ω і b.

4.2. Реалізація методу опорних векторів (*SVM*)

Розглядаючи оптимізаційну задачу методу опорних векторів, ми отримали таку систему:

$$\{\frac{1}{2}||w||^2 + C \cdot \sum_{i=1}^{l} \xi_i \to \min_{w,b,\xi} M_i(w,b) \geq 1 - \xi_i, \quad i = 1,2,\dots,l \; \xi_i \geq 0,$$

$$i = 1,2,\dots,l \quad (*)$$

і з неї дійшли алгоритму для пошуку параметрів ω і b:

$$\sum_{i=1}^{l}\big(1 - M_i(w,b)\big)_+ + \frac{1}{2C}||w||^2 \to \min_{w,b}$$

Для мінімізації цього функціоналу звичайні градієнтні методи не підходять, функція втрат тут безперервна, але з гладка (похідні немає у точці перегину).

Як варіант можна скористатися субградієнтними методами, тобто обчислювати похідну за правилом:

$$CJ(w,\ b) = C\ (1 - M_i(w,\ b)) = \{-1\ , J(w,b)\ < 1, 0\ ,\ J(w,b)\ \geq 1,$$

Але спочатку розв'язання задачі оптимізації методу опорних векторів зводилося до розв'язання системи (*). Тобто задачі квадратичного програмування, мінімізації коефіцієнтів при лінійних обмеженнях у вигляді нерівностей. Такий підхід призводить до досить ефективних чисельних методів і, крім того, дозволяє виділити об'єкти (спостереження), на основі яких розраховуються коефіцієнти w і b. Це цікава додаткова інформація про структуру навчальної вибірки.

Ми не будемо наводити докладного висновку завдання квадратичного програмування для системи (*). Для тих, хто хоче

поринути у цю математику, скажу лише, що тут використовується умова Каруша-Куна-Таккера з пошуком сідлової точки функції Лагранжа.

У результаті ми приходимо до того, що коефіцієнти можуть бути обчислені за формулою: $w = \sum_{i=1}^{l} \lambda_i, y_i, x_i$, де $\{\lambda_i\}$ – деякі коефіцієнти, які також обчислюються в процесі вирішення даної оптимізаційної задачі.

Тобто, виходить, що оптимальний вектор видається у вигляді лінійної комбінації спостережень з навчальної вибірки. І якщо для будь-якого i-го спостереження $\lambda_i \neq 0$. Отже, він не використовується для обчислення вектору ω.

Що примітно, нульових значень виходить досить багато і з усієї навчальної вибірки залишається кілька, які і граю роль при розрахунку коефіцієнтів. Такі спостереження (вектори) одержали назву опорних. Звідси й походить назва метод опорних векторів.

Взагалі значення коефіцієнтів λ можна інтерпретувати, таким чином:

$\lambda_i = 0, \xi_i = 0, M_i \geq 0$ – периферійні (неінформативні) об'єкти;

$0 < \lambda_i < C; \xi_i = 0; M_i = 1$ – опорні граничні об'єкти;

$\lambda_i = C, \xi_i > 0, M_i < 1$ – опорні помилкові об'єкти.

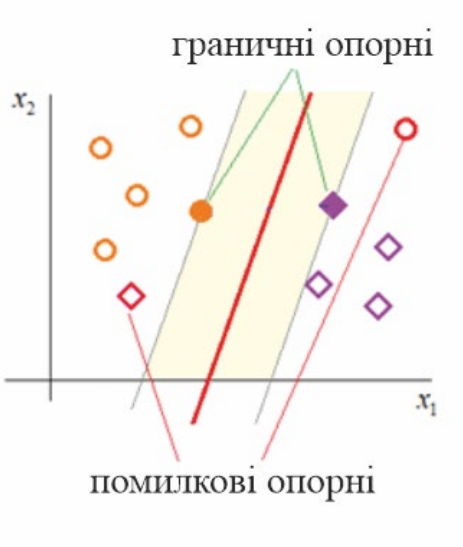


Рис. 4.7

4.3. Метод опорних векторів (SVM) із нелінійними ядрами

На попередньому занятті я наголосив, що для лінійного бінарного класифікатора виду:

$$a(x) = sign\ (\langle w, x\rangle - b)$$

коефіцієнти в методі опорних векторів можуть бути обчислені за формулою:

$$w = \sum_{i=1}^{l} \lambda_i, y_i, x_i$$

де $\{\lambda_i\}$ – деякі коефіцієнти. Причому ненульові коефіцієнти $\{\lambda_i \neq 0\}$ **відповідають опорним або хибним векторам (викидам).**

Якщо об'єднати ці дві формули, то виходить, що:

$$a(x) = sign\ (\sum_{i=1}^{l} \lambda_i, y_i \langle w, x \rangle - b)$$

(Тут сума взята не по всіх об’єктах, а тільки по опорних, для яких $\{\lambda_i \neq 0\}$).

Звідси добре видно, що класифікатор обчислює виважену суму скалярних творів опорних векторів. $\{x_i\}_{i=1}^{h}$ – з деяким вхідним вектором x віднімає зсув b і визначає знак.

Графічно це виглядає так:

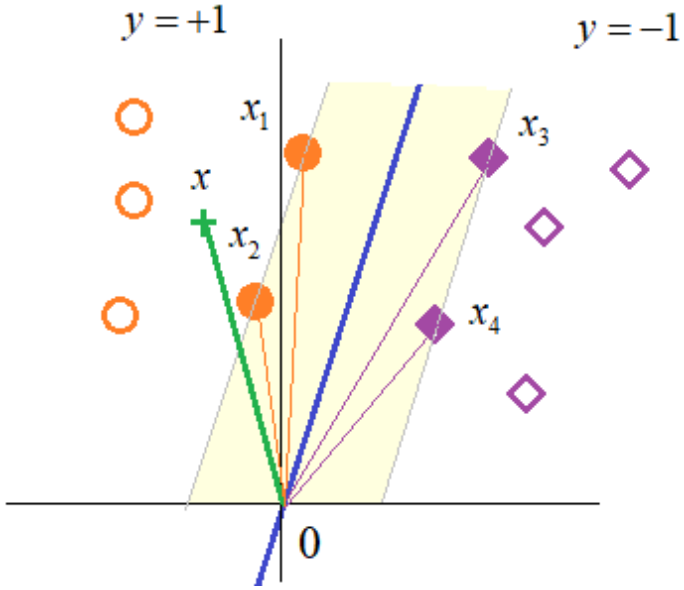


Рис. 4.8

Нехай у нас лінійно розділена вибірка з чотирма граничними векторами x_1, x_2, x_3, x_4. Зміщення для простоти дорівнює нулю ($b = 0$). Тоді для довільного вектору x обчислюватиметься наступна лінійна комбінація: $\lambda_1 \cdot \langle x_1, x \rangle + \lambda_2 \cdot \langle x_2, x \rangle - \lambda_3 \cdot \langle x_3, x \rangle - \lambda_4 \cdot \langle x_4, x \rangle$.

Або її можна записати в такому вигляді. Спочатку підсумовуємо опорні вектори для одного та другого класів образів:

$$w_+ = \lambda_1 x_1 + \lambda_2 x_2$$

$$w_- = \lambda_3 x_3 + \lambda_4 x_4$$

А потім, на який з них припадає найбільша проекція вектору x: $\langle w_+, x \rangle - \langle w_-, x \rangle$.

Класифікатор просто бере знак від цієї величини і видає своє рішення: $a(x) = sign(\langle w_+, x \rangle - \langle w_-, x \rangle)$.

Ось так можна інтерпретувати роботу лінійного бінарного класифікатора у методі опорних векторів.

SVM з нелінійними ядрами. Але повернемося до загальної формули моделі лінійного класифікатора: $a(x) = sign\ (\sum_{i=1}^{h} \lambda_i y_i \langle x_i, x\rangle - b)$ і поставимо питання, а якщо лінійні перетворення:

$$\langle x_i, x\rangle = x_i^T \cdot x,\ \ i = 1, 2, \ldots, h.$$

у вихідному просторі ознак замінити на якісь нелінійні? Наприклад, взяти і звести цей скалярний добуток у квадрат:

$$\langle x_i, x\rangle^2 = \langle x_i^T, x\rangle^2,\ i = 1, 2, \ldots, h.$$

Чи зможемо ми тоді повторити ті самі обчислення для знаходження опорних векторів і коефіцієнтів $\{\lambda_i\}$?

Виявляється так, для зазначеного перетворення і деяких інших нелінійних перетворень рішення системи шляхом Каруша-Куна-Таккера з допомогою функції Лагранжа залишається незмінним.

Взагалі існує теорема, яка свідчить:

Функція $K(x, x')$ є ядром (підходить для задачі SVM) тоді і лише тоді, коли вона симетрична: $K(x, x') = K(x, x')$ і невід'ємно визначено:

$$\int_x \int_x K(x, x')g(x)g(x')dxdx' \geq 0 \text{ для будь-якої } g\colon X \to \mathbb{R}.$$

Таким чином цю теорему надано лише формально. Насправді їй керуватися досить складно (як кажуть, вона конструктивна).

Але, зокрема, перетворення: $K(x, x') = \langle x, x'\rangle^2$, задовольняє цю теорему.

Давайте розпишемо його для двовимірних векторів:

$$u = [u_1, u_2]^T;\ v = [v_1, v_2]^T,$$

отримаємо:

$$K(u, v) = \langle u, v'\rangle^2 = (u_1v_1 + u_2v_2)^2 = u_1^2v_1^2 + u_1^2v_1^2 + 2u_1v_1\ u_2v_2 =$$
$$= \langle \left[u_1^2, u_2^2, \sqrt{2}u_1u_2\right]^T, \left[v_1^2, v_2^2, \sqrt{2}v_1v_2\right]^T \rangle.$$

Тобто зведення скалярного твору в квадрат двовимірних векторів – це аналог скалярного твору тривимірних векторів:

$$\psi(u) = \left[u_1^2, u_2^2, \sqrt{2}u_1u_2\right]^T,\ \psi(v) = \left[v_1^2, v_2^2, \sqrt{2}v_1v_2\right]^T.$$

Фактично, функція $\psi(x)$ формує новий тривимірний простір ознак, у якому починає працювати звичайний лінійний алгоритм опорних векторів.

Але з позиції вихідного двовимірного простір ознак ми отримуємо поліноміальні функції ступеня два. Причому поліноми тут не довільні, а складаються тільки з омонімів ступеня два (складених менших ступенів тут немає).

Я наведу характерні малюнки, взяті з офіційної документації *Scikit-Learn* роздільних гіперплощин для різних типів ядер:

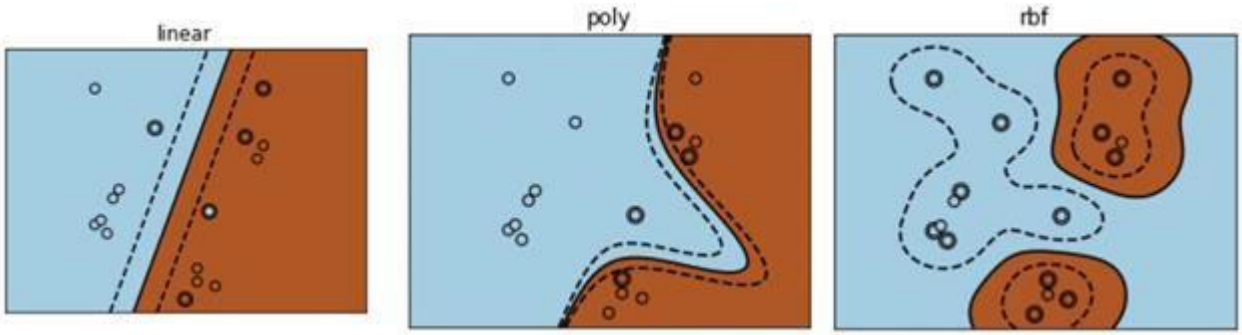


Рис. 4.9

Тут *linear* – звичайне скалярне утворення; *poly* – поліноміальне ядро, яке утворюється функціональними перетвореннями виду:

$$K(x, x') = \langle x, x' \rangle^d,$$

$$K(x, x') = (\langle x, x' \rangle + 1)^d,$$

де *rbf* – радіальні ядра, що визначаються виразом:

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right).$$

У практиці *SVM* ще можна зустріти ядро виду:

$$K(x, x') = th\ (k_1 \langle x, x' \rangle - k_0), \quad k_0, k_1 \geq 0.$$

На його основі виходить аналог двошарової нейронної мережі із сигмоїдальними функціями активації.

***SVM* як двошарова нейронна мережа**

Вас може трохи здивувати, причому тут нейронна мережа, коли ми говоримо про зовсім іншу структуру алгоритму на основі роздільної гіперплощини? Але насправді *SVM* можна у вигляді наступної обчислювальної структури.

Оскільки вихід моделі при довільних ядрах обчислюється за такою формулою:

$$a(x) = sign\ (\sum_{i=1}^{h} \lambda_i y_i K(x_i, x) - b),$$

то маємо:

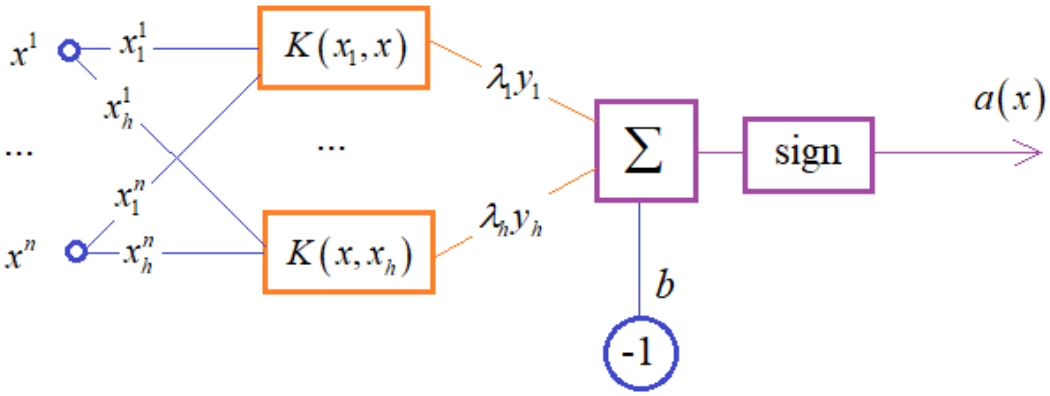


Рис. 4.10

То на прихованому шарі у нас обчислюються згортки вхідного вектора x з опорними векторами $x_1, \ldots, x_h$ з урахуванням обраного ядра $K(x, x')$.

Потім, всі ці значення множаться на вагові коефіцієнти $\lambda_1 y_1, \ldots, \lambda_h x_h$, підсумовуються та пропускаються через знакову функцію активації. Причому *SVM* нам відразу визначив необхідну кількість нейронів прихованого шару і, звичайно ж, значення вагових коефіцієнтів.

Способи синтезу ядер.

На закінчення цього заняття відзначу кілька простих правил синтезу ядер методу опорних векторів.

До основних можна віднести такі підходи:

$K(x, x') = \langle x, x' \rangle$ – скалярний добуток;

$K(x, x') = 1$ - константа;

$K(x, x') = K_1(x, x') \cdot K_2(x, x')$ – добуток ядер (придатних для *SVM*);

$K(x, x') = \langle \psi(x), \psi(x') \rangle, \ \forall \psi : X \to \mathbb{R}$ – застосування функції;

$K(x, x') = \alpha_1 K_1(x, x') + \alpha_2 K_2(x, x'), \alpha_1, \alpha_2 > 0$ – сумма ядер.

Існують деякі інші способи синтезу ядер. Але часто на практиці використовує досить обмежений клас – це лінійні, поліноміальні, радіальні та гіперболічні тангенси. Зазвичай їх цілком вистачає більшості завдань і саме такі ядра вбудовані в пакет *Scikit-Learn* для методу опорних векторів. Причому для кожного типу ядра обчислювальна схема *SVM* дещо змінюється. Тому не можна сформувати універсальний функціонал для довільних ядер (якщо звичайно не використовувати субградієнтні методи).

Переваги та недоліки *SVM*. Отже, якщо нам вдається звести завдання до методу опорних векторів, ми автоматично отримуємо наступні переваги перед субградієнтними методами та нейронними мережами:

- тут вирішується оптимізаційне опукле завдання квадратичного програмування, яке має одну точку мінімуму (єдине рішення).

- дозволяє виділити опорні вектори, у тому числі і викиди, які можуть бути присутніми в навчальній вибірці (іноді *SVM* застосовують для знаходження викидів).

- у порівнянні з нейронними мережами дозволяє визначити необхідну кількість нейронів прихованого шару як число опорних векторів.

Звичайно, універсальних методів у природі не існує і *SVM* притаманні ряд недоліків:

- немає загальних підходів до завдання оптимізації для довільних ядер $K(x, x')$.

- немає вбудованого відбору ознак (аналог $L1$-регуляризатора).

- необхідно підбирати гіперпараметр C для кожного розв'язуваного завдання.

Тим не менш, вважається, що *SVM* є найкращим методом класифікації серед інших лінійних класифікаторів. Завдяки максимізації ширини смуги між класами, він, як правило, призводить до кращих узагальнюючих здібностей на реальних даних. Його відносно легко модифікувати для нелінійного випадку (для різних ядер), а також використовувати $L1$-регуляризатор.

4.4. Ймовірна оцінка якості моделей

На попередніх заняттях ми з вами познайомилися з кількома підходами для вирішення задач бінарної класифікації, зокрема з логістична регресією та методом опорних векторів.

У всіх випадках ми маємо якусь параметричну модель $a(x) = sign(g(x, w))$, яка видає мітки класів: y О {-1; +1} і дозволяє обчислити відступ для об'єктів навчальної чи тестової вибірок: $M_i = y_i \cdot g\langle x_i, w\rangle$, $i = 1, 2, \ldots, l$.

Наприклад, для лінійних моделей відступ визначається звичайним скалярним добутком: $M_i = y_i \cdot \langle w, x_i\rangle$, $i = 1, 2, \ldots, l$.

Маючи ці дані, нам хотілося б вміти оцінювати якість отриманої моделі. Звичайно, ви зараз можете сказати, що немає нічого простішого: давайте, наприклад, подивимося на частку правильної класифікації у відкладеній (тестової) вибірці (метрика *accuracy*):

$$accuracy = \frac{1}{k}\sum_{i=1}^{k}[M_i > 0] = \frac{1}{k}\sum_{i=1}^{k}[a(x_k) = y_k].$$

І вона справді покаже, як часто класифікатор давав вірні відповіді на тестові дані. Наприклад, якщо вся вибірка $k = 1000$, а модель зробила 10 помилок, то accuracy становитиме: $accuracy = \frac{990}{1000} = 0{,}99$.

Але це число нічого не говорить, принаймні, про два важливі аспекти:

- наскільки модель була впевнена у своїх прогнозах;

- наскільки була збалансована вибірка під час обчислення accuracy.

Почнемо із першого аспекту. Як можна визначити впевненість класифікатора у своїй відповіді? Напевно, у найпростішому випадку, можна побудувати розподіл відступів $M_i = y_i \cdot g\langle x_i, w\rangle$, $i = 1, 2, \ldots$ для всіх об'єктів відкладеної вибірки:

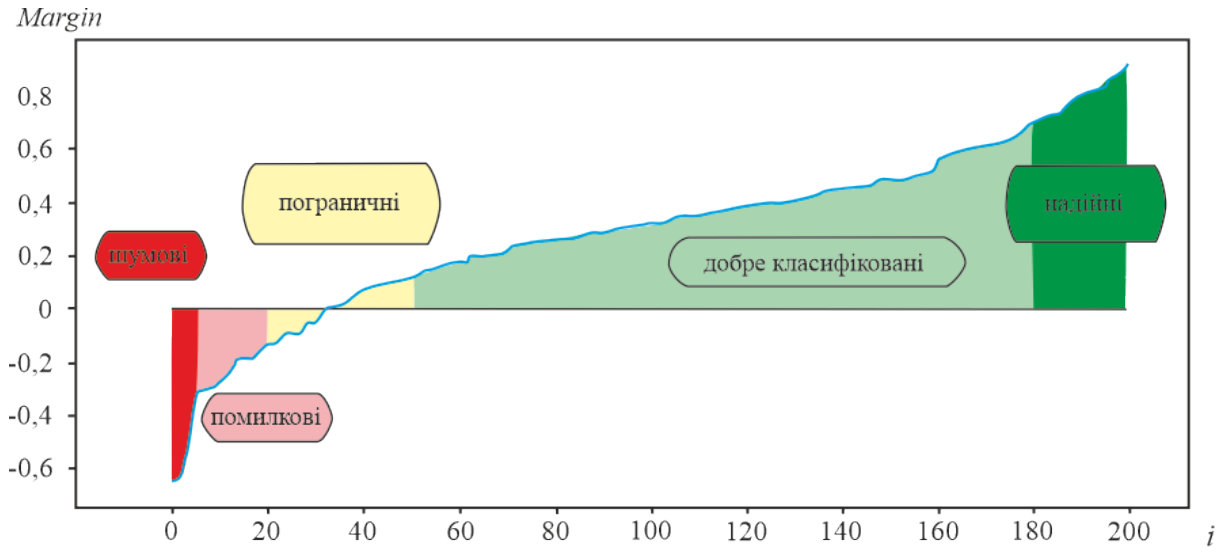


Рис. 4.11

Якщо графік має «швидкий» перехід через нуль із мінімальною зоною прикордонних об'єктів, то, очевидно, такий класифікатор повинен давати надійні оцінки та мати хорошу узагальнюючу спроможність. Якщо жовта область полога і містить багато прикордонних об'єктів, це привід задуматися про надійність моделі і, можливо, спробувати інші алгоритми на вирішення поточного завдання.

Але це суто візуальний метод оцінки надійності моделі, хоч і дуже непоганий. Однак, у ряді завдань нам хотілося б отримувати чисельні характеристики впевненості моделі в правильній класифікації. І, як це зробити, ми з вами вже говорили, коли розглядали ймовірнісний погляд на завдання машинного навчання.

Зокрема, для логістичної регресії за функції втрат:
$L(M) = \log(1 + e^{-M})$ було явно показано, що величину відступу (лінійної моделі): $M_i = y_i \cdot \langle w, x_i \rangle$, $i = 1, 2, \ldots$ можна перевести у значення ймовірності за допомогою сигмоїдної функції:

$$P(y \mid M) = \frac{1}{1 - e^{-M}}.$$

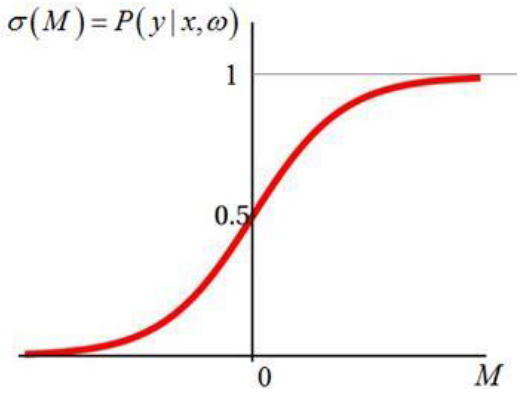


Рис. 4.12

Але наскільки ця можливість відповідатиме дійсності?

Тобто якщо впевненість моделі в прогнозі становить, наприклад, 0,7, то, в принципі, ми очікуємо, що цей прогноз буде збуватися в 70% випадках. Наприклад, модель оцінює позичальників та вирішує, кому дати кредит (мітка класу +1), а кому відмовити (мітка класу -1). Припустимо, вона видає прогноз +1 зі значенням ймовірності 0,9. Отже, ми очікуємо, що 90% таких позичальників будуть віддавати кредит. І тут важливо, щоб це значення якомога точніше збіглося з реальною поведінкою позичальників. Інакше банк зможе коректно розрахувати свої ризики. Тому, це далеко не пусте питання.

То як нам переконатися в коректності імовірнісного значення? Зазвичай, це роблять за відкладеною (тестовою) вибіркою. Весь імовірнісний діапазон [0; 1] розбивають на піддіапазони:

$p_0 = 0 < p_1 < p_2 < \ldots << p_m = 1 + \varepsilon$

Потім перебирають всі k об'єктів тестової вибірки, обчислюючи для кожного ймовірність:

$P(y = y_j \mid M_j), \quad j = 1, 2, \ldots, k$

Після цього об'єкт міститься у відповідний піддіапазон:

$p_t \leq P(y = y_j \mid M_j) < p_{t+1}, \quad t = 0, 1, \ldots, m - 1$

В результаті ми для кожного ймовірнісного інтервалу обчислюємо частку об'єктів, які в нього потрапили:

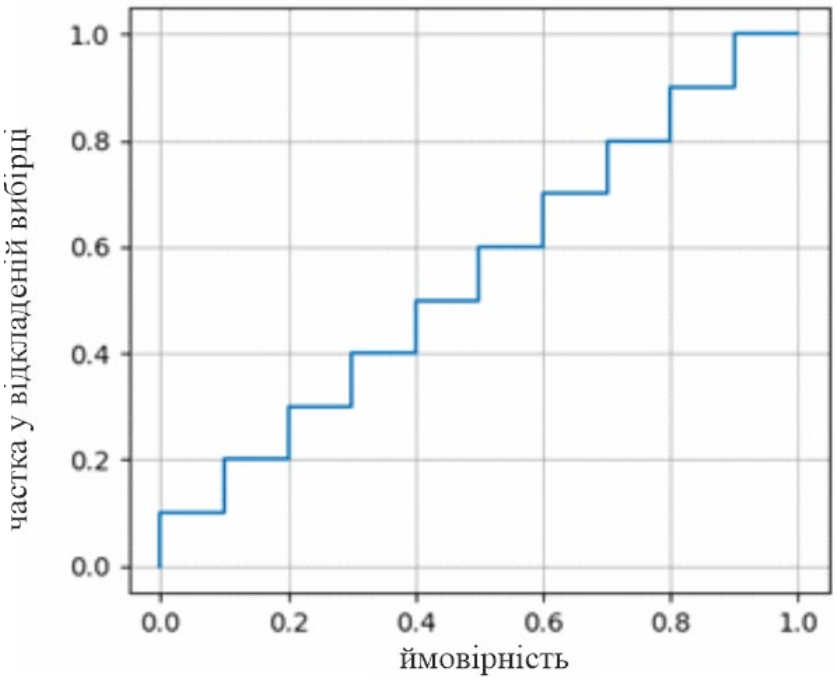


Рис. 4.13

Якщо виходить приблизно діагональний графік, то розрахункові ймовірності відповідають експериментальним показанням і таким значенням можна довіряти. Зокрема, логістична регресія, як правило, дає такий діагональний графік, тому відступи її моделі можна спокійно перераховувати в ймовірності і користуватися цими значеннями надалі.

А от якщо побудувати аналогічний графік для методу опорних векторів, то зазвичай виходить наступна картина:

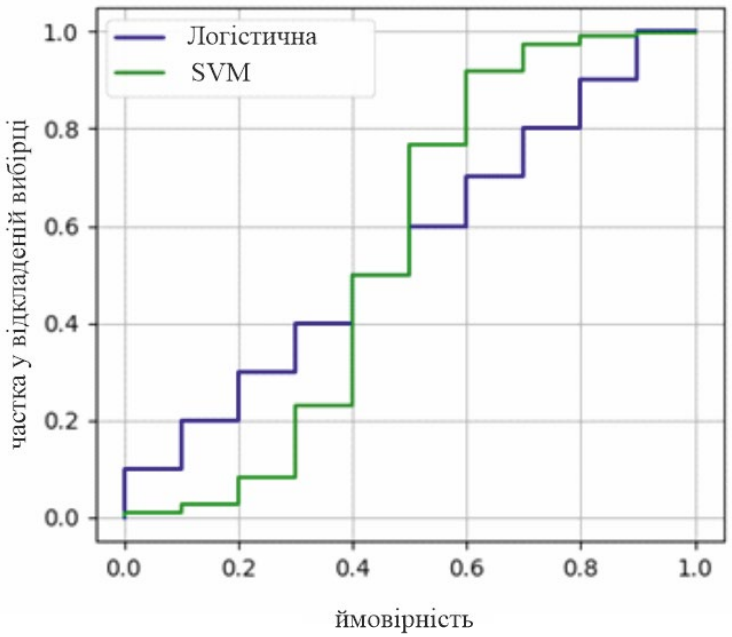


Рис. 4.14

Тут зелений графік дещо перекошений щодо діагоналі (синього графіка). Це означає, що відступи в *SVM* не можуть бути переведені безпосередньо в ймовірності. Виходить, що ми не можемо використовувати ймовірну інтерпретацію виходів моделі у разі методу опорних векторів?

Прямо так, не можемо. Але можна зробити калібрування кривою: приведення розрахункових ймовірностей до експериментальних значень. Наприклад, бачимо, що з діапазону [0,3; 0,4) експериментальна ймовірність становить приблизно 0,2. Значить значення ймовірностей цього діапазону треба замінювати на 0,2. І так всім інтервалів. Такі калібрування справді проводять і це нормальна практика.

Тепер ми з вами знаємо, що ймовірності, які можна розрахувати для моделей, необхідно перевіряти за відкладеною вибіркою. І, за потреби, проводити калібрування цих значень.

На цьому ми завершимо поточне заняття, а на наступному продовжимо цю тему та розглянемо методи оцінки надійності моделей при незбалансованих вибірках.

4.4. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Дайте визначення методу опорних векторів (Support Vector Machine - SVM).
2. Дайте визначення функція втрат (hinge loss).
3. Дайте визначення методу опорних векторів (*SVM*) із нелінійними ядрами.
4. Переваги та недоліки *SVM.*
5. Як оцінюють якість моделей?

4.5. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.
2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA
3. Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale FPGA-based convolutional networks. In R. Bekkerman, M. Bilenko, and J. Langford, editors, Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press.
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

Лекція №5. Метод головних компонентів

Результати навчання, які формуються:

ДРН4: Використовувати методи добування знань, кластеризації та класифікації.

ДРН5: Вміти створювати ефективні алгоритми для обчислювальних задач системного аналізу та систем підтримки прийняття рішень.

ДРН6: Застосовувати знання машинного навчання в умовах слабо структурованих даних різної природи.

5.1. Показники precision та recall. F-міра

На попередньому занятті ми з вами побачили, як коректна оцінювати впевненість моделі віднесення об'єкта до одного із двох класів, використовуючи значення ймовірностей. Зазвичай, це цілком задовільна характеристика, якщо класи у вибірці збалансовані, тобто, є приблизно однакове число об'єктів першого і другого класів. Проте ситуація кардинально змінюється, якщо потужності класів мають сильну диспропорцію.

Тут часто наводять приклад завдання кредитного скорингу, коли модель прогнозує видавати кредиту позичальнику або відмовити. У цьому випадку може виявитися, що серед великої кількості заявок кредити схвалюються лише невеликою частиною.

Давайте уявимо, що з 1000 заявок реально було схвалено лише 100:

Таблиця 5.1

№	Заявка: з/п, стаж, вік, борги	Схвалення
1	30000, 5, 35, 0	+1
2	23450, 10, 44, 0	+1
…	…	…
100	50000, 7, 54, 5000	+1
101	10000, 5, 60, 7000	-1
102	14500, 17, 55, 500	-1
…	…	…
1000	20000, 3, 65, 0	-1

У вибірці є сильний дисбаланс класів: у першому всього 100 об'єктів, а в другому – 900. І якщо оцінювати якість алгоритму, наприклад, за метрикою accuracy: $accuracy = \frac{1}{k}\sum_{i=1}^{k}[a(x_i) = y_i]$, то при константній моделі: $a(x) = -1$, отримаємо значення: $accuracy = \frac{900}{1000} = 0{,}9$.

Якщо не знати про дисбаланс класів, можна було б подумати, що ми отримали відмінний результат, а насправді припинили функціонування кредитної організації, т.к. вона всім охочим взяти кредит видає відмову.

Крім дисбалансу класів (а, можливо, і разом з ним), ціна помилки за невірний прогноз може відрізнятися для різних позичальників.

Наприклад, якщо алгоритм відмовив у кредиті дуже надійному позичальнику, то розумно таку відповідь штрафувати набагато сильніше, ніж при відмові сумнівному позичальнику.

Цей момент метрика accuracy також не враховує.

Щоб точніше оцінювати якість моделі у таких ситуаціях, вводять такі поняття:

Таблиця 5.2

	Відповідь класифікатора	Правильну відповідь
TP, True Positive	$a(x_i) = +1$	$y_i = +1$
TN, True Negative	$a(x_i) = -1$	$y_i = -1$
FP, False Positive	$a(x_i) = +1$	$y_i = -1$
FN, False Negative	$a(x_i) = -1$	$y_i = +1$

Запам'ятати це можна дуже просто. Друге слово (positive чи negative) належить до виходу класифікатора, а перше (true чи false) – вірний вихід класифікатора чи ні.

Ще всі ці комбінації при бінарній класифікації подають у вигляді такої таблиці:

Таблиця 5.3

	$y_i = +1$	$y_i = -1$
$a(x_i) = +1$	*TP*	*FP*
$a(x_i) = -1$	*FN*	*TN*

Що вам простіше, перший чи другий варіант, вибирайте самі.

Отже, на основі минулих емпіричних даних (це може бути і сама навчальна вибірка), ми легко можемо обчислити величини *TP*, *FP*, *FN*, *TN* – це відповідні числа вірних та невірних прогнозів алгоритму обох класів.

Наприклад, ми оцінюємо роботу спам-фільтра будь-якого месенджера та отримали такі дані:

– (клас +1): надійшло 100 нормальних (не спам) повідомлень, з яких модель 90 позначила як спам, а 10 – як спам;

– (клас -1): надійшло 10 спам-повідомлень, з яких модель виділила лише 7.

Отримуємо значення величин:

$TP = 90$; $FN = 10$; $FP = 3$, $TN = 7$

Умовний розподіл величин *TP*, *FP*, *FN*, *TN* можна представити графічно у вигляді наступної діаграми, взятої з сайту Wikipedia:

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

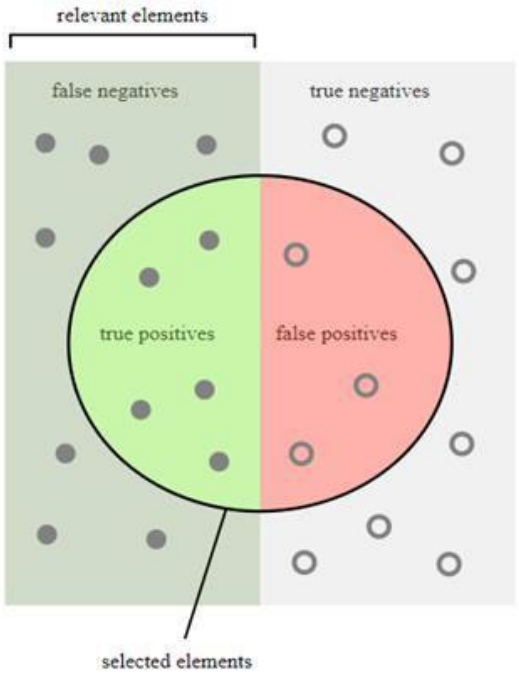


Рис. 5.1

Precision та recall

На основі цих даних ми легко можемо обчислити вже знайому нам метрику accuracy, таким чином:

$$accuracy = \frac{1}{l}\sum_{i=1}^{l}[a(x_i) = y_i] = \frac{TP + TN}{TP + FP + FN + TN} \approx 0{,}88.$$

Тобто, використовуючи величини *TP*, *FP*, *FN*, *TN* ми можемо будувати різні метрики. Спробуємо сконструювати нові показники алгоритму, які б враховували дисбаланс класів. Першою такою величиною буде precision (точність):

$$precision = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{l} [a(x_i)=+1][a(x_i)=y_i]}{\sum_{i=1}^{l} [a(x_i)=+1]}.$$

Вона показує, як можна довіряти моделі, коли вона видає позитивний клас +1. Наприклад, стосовно завдання кредитного скорингу, в знаменнику вказані всі позичальники, яким модель видала кредит, а чисельнику – ті їх, які повернули кредит. Якщо виявляється, що ця величина 0,9 і більше, очевидно, що до рекомендацій такий моделі слід прислухатися, т.к. переважна більшість рекомендацій виправдовується (позичальники повертають кредит).

Але цієї властивості явно недостатньо, т.к. вона не показує наскільки аналізована модель $a(x)$охоплює всіх позичальників позитивного класу. Можливо, вона так завищує вимоги та видає кредити лише дуже заможним організаціям чи громадянам. Тоді, як менш заможні клієнти нею не враховуються, а серед них, вочевидь, чимало тих, хто здатний розплатитися за взятими зобов'язаннями. Тому друга характеристика має показувати охоплення позитивного класу нашою моделлю.

Вона називається recall (повнота) і визначається виразом:

$$recall = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{l} [a(x_i)=+1][a(x_i)=y_i]}{\sum_{i=1}^{l} [y_i=+1]}.$$

Об’єднання precision та recall

В ідеалі хороша модель має показувати високі значення за двома параметрами precision і recall. Однак, на практиці віддають більшу перевагу одному показнику і менше – іншому, тому що одночасно максимізувати обидва (знайти таку модель) часто дуже складно. І тут виникає таке завдання – об’єднання цих двох характеристик в одну, щоб на виході ми мали одне число, яким можна було б судити про якість отриманої моделі.

Лінійна комбінація (не робочий варіант)

Перше, що спадає на думку, просто взяти лінійну комбінацію від цих ознак з деякими вагами:

$$Q = \alpha_1 \cdot precision + \alpha_2 \cdot recall, \quad \alpha_1, \alpha_2 > 0.$$

Але це дуже хороший варіант, т.к. для будь-яких α_1, α_2 можна вигадати контрприклад, де показник Q дасть спірний результат.

Наприклад, при $\alpha_1, \alpha_2 = \frac{1}{2}$.

Отримаємо:

$$Q = \frac{precision + recall}{2}$$

і далі:

$$\{precision = 0{,}2\ recall = 0{,}1 \qquad \text{Ю } Q = \frac{1 + 0{,}2}{2} = 0{,}6$$

$$\{precision = 0{,}6\ recall = 0{,}6 \qquad \text{Ю } Q = \frac{0{,}6 + 0{,}6}{2} = 0{,}6$$

Як бачите, при двох абсолютно різних показниках precision і recall, ми отримуємо те саме значення *Q*. Значить, таке об'єднання не дає об'єктивної картини.

Використання функції min

Інший радикальний варіант – це скористатися функцією min для вибору мінімального значення з двох показників:

$$Q = \min(precision, recall).$$

Але тут, очевидно, виникає інший недолік, ми у значенні *Q* бачитимемо лише одну мінімальну величину, а другу просто відкидатимемо. Це не дуже добре, тому що зазвичай важливі обидві характеристики з різними вагами.

Гармонічне середнє (*F*-міра)

Прийнятний результат можна отримати за допомогою обчислення гармонійного середнього (*F*-заходи) на основі precision та recall:

$$F = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

Чому це добре, я покажу на лініях рівня. Нехай ми маємо графік на площині. По одній осі відкладаємо різні значення precision, а по другій – recall:

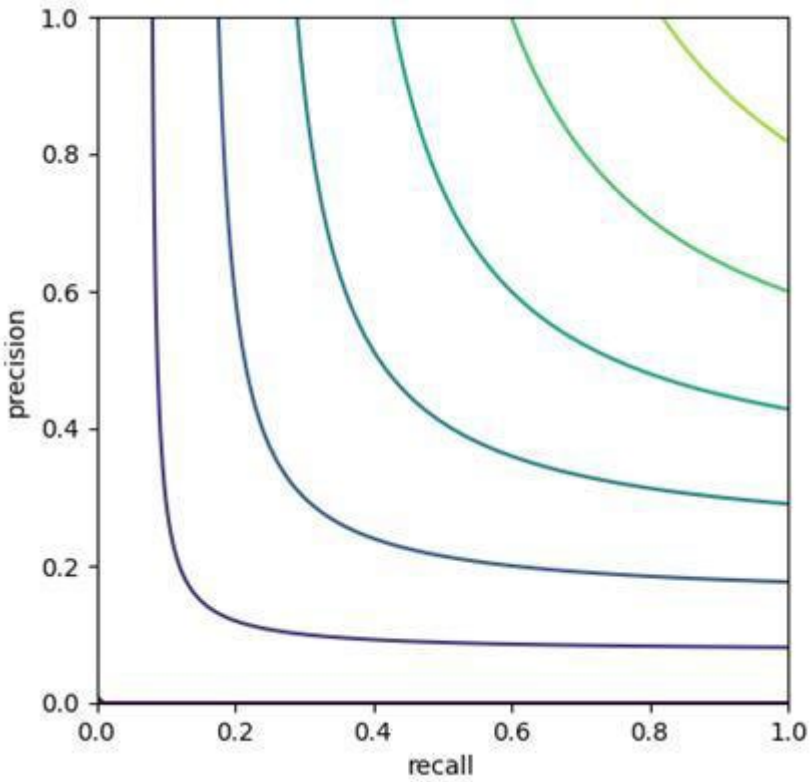


Рис. 5.2

Ми бачимо округлі лінії. Це означає, що з будь-якого фіксованого значення *F*, збільшуючи, наприклад, precision, зменшуватиметься recall. Або,

збільшуючи recall, буде зменшуватися precision. При цьому гармонійне середнє нагадує лінії рівня функції min, тобто, як би «тягнеться» до меншої величини.

Наприклад:

$\{precision = 0{,}2\ recall = 0{,}1$ Ю $F \approx 0{,}33$

$\{precision = 0{,}6\ recall = 0{,}6$ Ю $F = 0{,}6$

F_β-міра

Розглянута *F*-міра вже дає непогані результати об'єднання двох показників precision і recall, але вона враховує їх однаково. А що якщо ми хочемо приділяти більше уваги показнику precision або, навпаки, *recall*?

Для цього було придумано розширення *F*-заходу до *F*-заходу, яка визначається виразом:

$$F_\beta = \frac{(1+\beta^2) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall}$$

Тут β – будь-яке дійсне число:

$$\{\beta < 1 \rightarrow precision\ \beta > 1 \rightarrow recall$$

При $\beta = 1$ отримуємо звичайну *F*-міру.

Геометричне середнє

Дещо рідше використовується геометричне середнє два показники:

$$G = \sqrt{precision \cdot recall}.$$

Ця метрика вважається гіршою, ніж *F*-міра, т.к. «тяжіє» до вищих значень. Наприклад:

$\{precision = 0{,}8\ recall = 0{,}1$ Ю $\{F \approx 0{,}18\ G \approx 0{,}28$

Тобто, гармонійне середнє (*F*-міра) виявляється більш вимогливим до обох метрики precision і recall, а геометричне середнє більшою мірою «перекошене» до вищих значень. Тому, на практиці частіше вибирають саме *F*-міру та її аналоги.

5.2. Метрики якості ранжирування. *ROC*-крива

Продовжуємо знайомитись з метриками якості оцінок моделей при бінарній класифікації. І на цьому занятті ми порушимо питання дуже важливого питання оцінки якості моделей ранжирування. Але спочатку визначимося, що це клас моделей.

Розглянемо класичний приклад утримання клієнтів банку, стільникового оператора чи ще будь-якої організації. Нам поставили завдання з розробки моделі, яка прогнозувала б за деякими ознаками *x*, Що клієнт незабаром залишить установу (для визначеності банк):

$$a(x) = sign\ (\langle w, x \rangle) = \{+1\ \textit{піде};\ -1\ \textit{залишиться}$$

Звісно, модель робить лише припущення. Чи піде клієнт насправді чи ні нам невідомо. Ми можемо це дізнатися тільки через якийсь час. Але про всяк випадок таких «невпевнених» клієнтів було б добре зателефонувати і зробити

їм «приємну» пропозицію, щоб утримати. Однак, ресурси будь-якої організації обмежені та обдзвонювати всіх таких клієнтів може виявитися дуже невигідно. Тому тут краще виділити найневпевненіших, які з найбільшою ймовірністю збираються перейти в інший банк. І виникає питання, як визначити цих «невпевнених»?

Для цього надходять у такий спосіб. Сортують усіх клієнтів за значеннями скалярного добутку:

$\langle w, x_i \rangle$, i =1,2,…

Отримуємо таблицю:

Таблиця 5.4

$\langle w, x_i \rangle$	Реальний результат y_i
100	+1
98	+1
76	+1
75	-1
…	…
50	+1
25	+1
10	-1
6	-1
4	+1
1	-1
-8	+1
-10	-1
-14	-1
…	…

Тут найперші (верхні) клієнти – це найбільш «невпевнені», ті, що з найбільшою ймовірністю покинуть установу в найближчому майбутньому, а чим нижче, тим лояльніші. Причому ми можемо за історичними даними (або через якийсь час) визначити, як реально себе повів той чи інший клієнт (пішов чи залишився) – це цільові значення $\{y_i\}$.

Так ось, при обмежених ресурсах call-центру, було б логічно виділити в цій таблиці кілька перших клієнтів і саме їм робити пропозицію, від якої вони не зможуть відмовитися і таким чином утримати їх.

Математично це можна записати, таким чином: $a(x) = sign\ (\langle w, x\rangle - t)$.

Тут t – це певний поріг (число), яким поділяються клієнти на нелояльних (+1) і лояльних (-1). Встановлюючи, наприклад, цей поріг більше за нуль, ми виділятимемо найменш лояльних клієнтів з точки зору моделі. Причому, граничне значення – це змінна величина і може змінюватися, виділяючи більш менш лояльних клієнтів, наприклад, залежно від навантаження call-центру. У цьому вся ключова особливість роботи алгоритмів ранжирування. Нам доводиться розробляти модель, не знаючи конкретного порогового значення, тобто вона загалом (для будь-яких порогів) повинна давати хороші прогнози поведінки користувачів.

Наприклад, якщо при деякому порозі t, упорядковані за зменшенням значення скалярного твору, відповідатимуть чергуванню цільових значень $\{y_i\}$ на історичних даних, отже, модель робить поганий прогноз та ефективність роботи установи, швидше за все, різко знизиться. Тому, перш ніж застосовувати модель на практиці, її потрібно оцінити і зазвичай робиться це за допомогою розроблених метрик якості ранжування.

ROC-крива

Взагалі ідеальна модель ранжування – це та, у якої після сортування цільові значення $\{y_i\}$ також виявляються відсортованими не зростання (спочатку все +1, потім, все -1). Тоді, для будь-якого порога t ми будемо отримувати якісні результати. Але це в ідеалі. Насправді ж є помилки й у умовах потрібно зрозуміти, як будувати прогнози модель за різних порогів t. Для цього надходять, в такий спосіб. Використовують дві характеристики якості класифікації моделі:

False Positive Rate - частка помилкових позитивних класифікацій:

$$FRR(t) = \frac{FP(t)}{FP(t) + TN(t)} = \frac{\sum_{i=1}^{l}[y_i = -1][a(x_i,t)=+1]}{\sum_{i=1}^{l}[y_i=-1]}$$

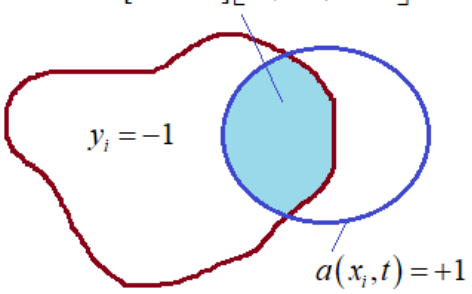


Рис. 5.3

Величину 1 – $FRR(t)$ ще називають специфічністю алгоритму.

True Positive Rate – частка правильних позитивних класифікацій (чутливість алгоритму):

$$TPR(t) = \frac{TP(t)}{TP(t) + FN(t)} = \frac{\sum_{i=1}^{l}[y_i = +1][a(x_i,t)=+1]}{\sum_{i=1}^{l}[y_i=+1]}$$

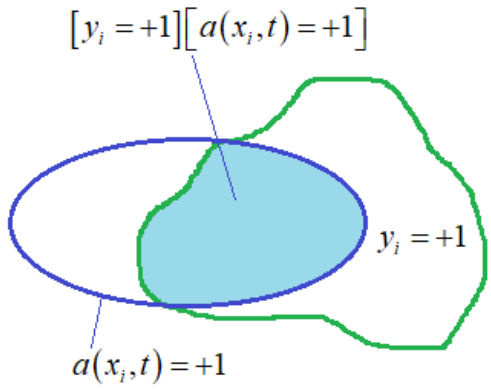


Рис. 5.4

Метрика: *TPR*(*t*) = *recall* те, що ми розглядали на попередньому занятті.

Причому формально величини *FPR*(*t*), *TPR*(*t*) можна обчислювати як для лінійних алгоритмів класифікації, але й будь-яких інших, головне, щоб ми мали можливість ранжування з використанням порога.

Отже, ми маємо дві характеристики *FPR*(*t*), *TPR*(*t*), які залежить від вільного параметра *t*. Як тепер за допомогою оцінити модель для завдання ранжирування?

Для цього будують графік, що історично (з радіотехніки) отримав назву Receiver Operating Characteristic або скорочено *ROC*-крива.

Графік будується в осях *TPR*/*FPR*:

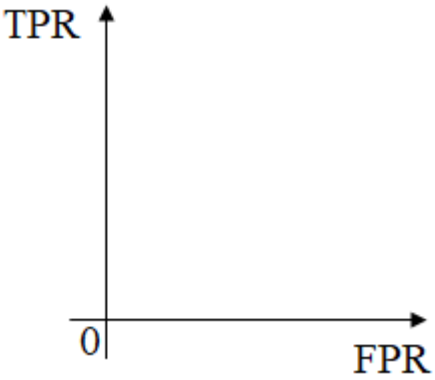


Рис. 5.5

Для цього ми перебиратимемо значення параметра t в діапазоні:

$$\begin{matrix} b(x_1) \geq & b(x_2) \geq & b(x_3) \geq \ldots \geq & b(x_l) & \\ t_1 & t_2 & t_3 & t_l & t_{l+1} \end{matrix}$$

де $t_{l+1} = b(x_l) + \varepsilon$, $\varepsilon > 0$ – нескінченно мала величина.

Спочатку вибираємо поріг t_1. Він більший за всі можливі значення скалярного твору в нашій вибірці.

Отже:

$$TPR(t_1) = \frac{TP(t_1)}{TP(t_1) + FN(t_1)} = \frac{0}{\sum_{i=1}^{l} [y_i = +1]} = 0$$
$$FPR(t_1) = \frac{FP(t_1)}{FP(t_1) + TN(t_1)} = \frac{0}{\sum_{i=1}^{l} [y_i = -1]} = 0$$

Отримуємо, що графік *ROC*-кривий виходить із нуля (0; 0).

Далі припустимо, що ми ідеальна модель, тобто. вона не робить помилок (повністю збігається з цільовими значеннями $\{y_i\}$).

Тоді для наступного порогу t_2 ми отримаємо деяке збільшення $TPR(t_2)$ і нульове значення для $FPR(t_2)$:

$$\{TPR(t_2) = \frac{1}{l_+} \; FPR(t_2) = 0$$

де $l_+ = \sum_{i=1}^{l} [y_i = +1]$ – число об'єктів позитивного класу.

Перебираючи далі пороги, ми, очевидно, постійно збільшуватимемо *TPR*, зберігаючи нульове значення *FPR*, поки не переберемо всі об'єкти позитивного класу. У цьому випадку *TPR* досягне максимального значення 1, а *FPR* поки що дорівнює нулю.

Далі, переходячи до негативних класів, у нас все буде навпаки – *TPR* залишається без змін, рівний 1, а *FPR* поступово збільшуватиметься, доходячи до 1.

У результаті, для ідеальної моделі графік *ROC*-кривий матиме вигляд (синя лінія):

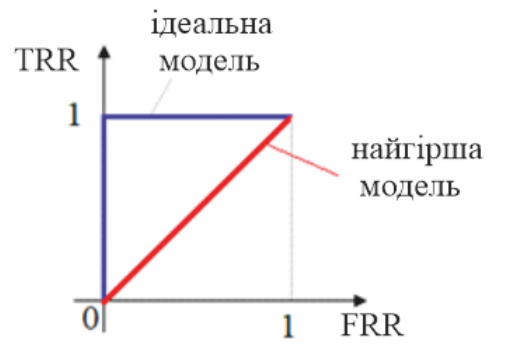


Рис. 5.6

А ось для найгіршої моделі, яка прогнозує значення бінарного класу з ймовірністю 1/2:

$$a(x) = \{A: +1, P(A) = \frac{1}{2} \; B: -1, P(B) = \frac{1}{2}$$

ROC-крива приблизно лежатиме на діагоналі (червона лінія).

Як ви вже здогадалися, реальні моделі з невеликою кількістю помилок прогнозів, лежать між червоною та синьою кривими:

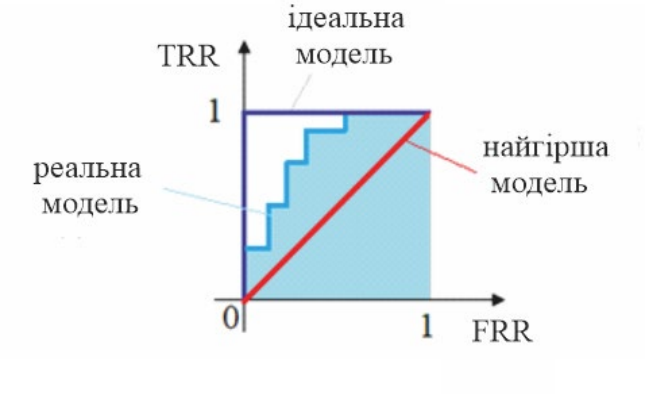


Рис. 5.7

Відмінно, *ROC*-криву ми навчилися будувати, але як тепер по ній отримати числову характеристику якості моделі? Тут є два поширені підходи. У першому випадку просто беруть площу під *ROC*-кривою. Така характеристика називається *AUC – ROC*, і чим вона більша, тим якісніша модель. Для ідеальної моделі площа дорівнює 1, а для найгіршої – 1/2. Хорошими вважаються моделі, які мають площі більше 0,9. Тобто, показник *AUC - ROC* показує, як добре модель сортує (ранжує) об'єкти класу.

У другому випадку обчислюють міру різниці площ між *ROC*-кривими реальної моделі та найгіршою, за такою формулою: $G = 2 \cdot AUC - ROC - 1$

Тобто, це подвійна різниця між площею *ROC*-кривої аналізованої моделі та найгіршим випадком:

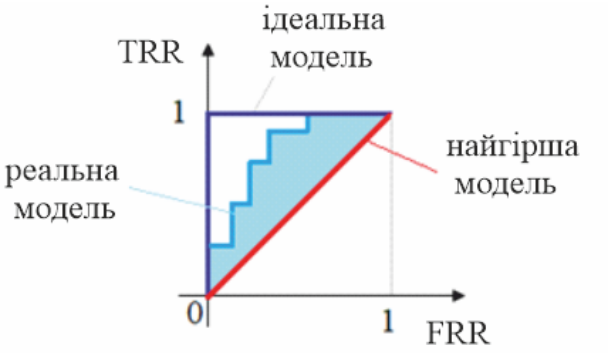


Рис. 5.8

Такий показник називають індексом Джині чи коефіцієнтом Джині. Але це лише дещо інший погляд на ту саму площу під *ROC*-кривою.

Основним недоліком цих показників (*AUC-ROC* та індекс Джині) є поганий облік у їх значеннях незбалансованості класів. Наприклад, якщо до позитивного класу належать 100 об'єктів, а до негативного 1 000 000, то площа під *ROC*-кривою вийде великий (вище 0,9), але при цьому ранжування об'єктів може бути дуже поганим. Ось цей момент слід враховувати, користуючись даними характеристиками якості ранжування.

5.3 Метод головних компонентів (Principal Component Analysis)

Раніше ми з вами говорили, що однією з ключових проблем у машинному навчанні є ефект перенавчання (overfitting). І для боротьби з ним передбачалося або зменшувати простір ознак (спрощувати модель), або вводити регуляризатори, або робити те й те й те. З регуляризаторами загалом усе зрозуміло. А ось як можна скоротити простір ознак, залишивши лише найважливіші характеристики об’єктів, ми докладно познайомимося на цьому і наступних заняттях.

Давайте для простоти припустимо, що у нас є навчальна вибірка, що складається з l об'єктів, представлена у двовимірному просторі ознак, що у загальному випадку описується функціями:

$$F = [f_1(x_1)\ f_2(x_1) \ldots\ldots f_1(x_l)\ f_2(x_l)\ ]_{l\times 2}.$$

(У найпростішому варіанті можна вважати, що $f_1(x_k) = x_{k1}, f_2(x_k) = x_{k2}$).

Нехай, наприклад, безліч об'єктів розподіллено близько нуля і витягнуто вздовж лінії під кутом 45 градусів:

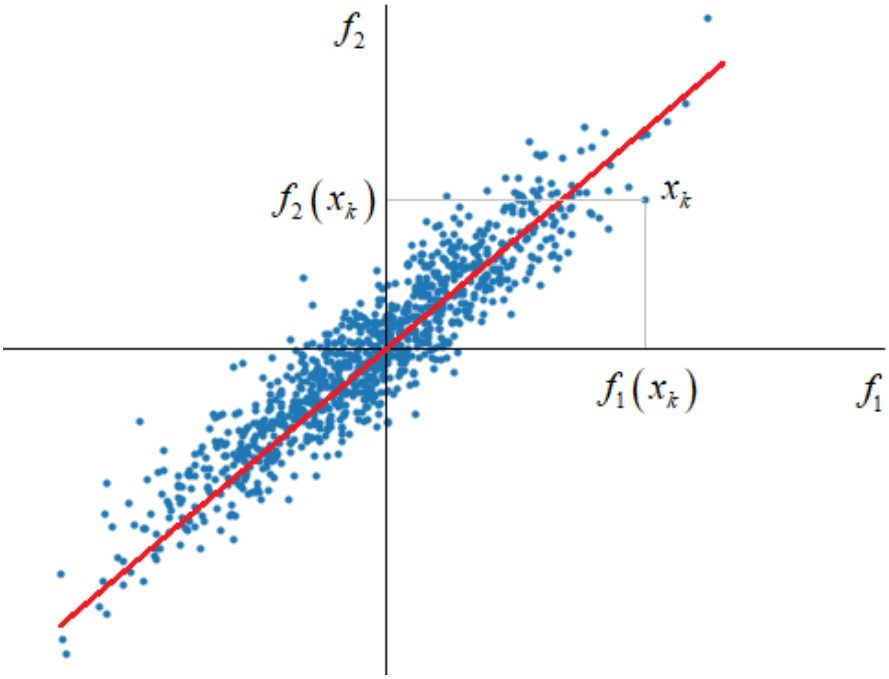


Рис. 5.9

Ми бачимо, що простір ознак $\{f_1, f_2\}$ зображено у вигляді декартової системи координат і кожна точка (об'єкт вибірки) однозначно описується двома координатами:

$$[f_1(x_k), f_2(x_k)]^T,\ k = 1, 2, \ldots, l.$$

З цього малюнка добре видно, що розкид проекцій точок на обидві осі приблизно однаковий з дисперсіями близько 0,25:

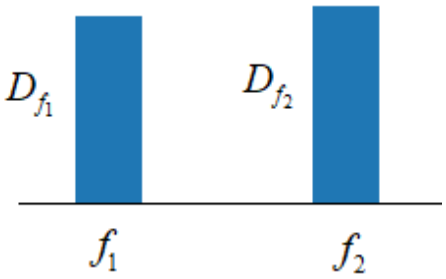


Рис. 5.10

При деяких обмеженнях, величини D_{f1}, D_{f2} можна сприймати як середню кількість інформації, що приходить на першу і другу вісь. Тобто для нашого прикладу розподілу об'єктів $\{x_i\}$ ознаки f_1, f_2 приблизно рівно-інформативні.

Давайте тепер подумаємо, чи можна для нашого конкретного випадку сформувати іншу ортогональну систему координат так, щоб на першу вісь припадали в середньому найбільші проекції векторів $\{x_i\}$, а на другу – частина, що залишилася?

Чому б нам не повернути декартову систему координат на 45 градусів за годинниковою стрілкою:

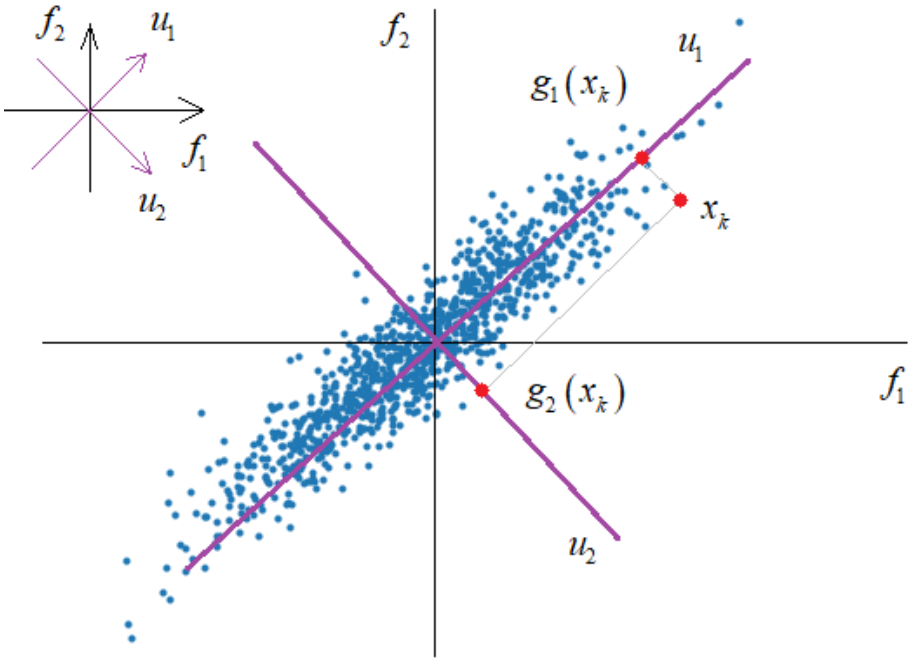


Рис. 5.11

З цього малюнка добре видно, що, в середньому, найбільша проекція припадатиме на вісь u_1, а частина, що залишилася – на вісь u_2. Тобто, у нас інформація про об’єкти більшою мірою концентруватиметься в ознаці u_1 і меншою мірою – в ознаці u_2 .

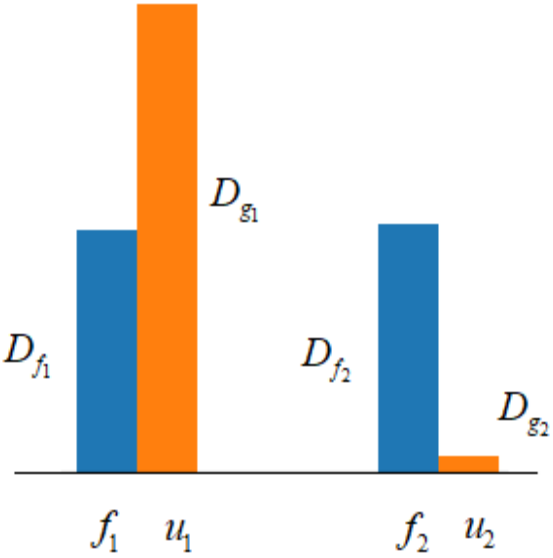


Рис. 5.12

Нову систему координат можна описати за допомогою наступних одиничних векторів:

$$u_1 = \left[\frac{1}{\sqrt{2}}; \frac{1}{\sqrt{2}}\right]^T; \; u_2 = \left[\frac{1}{\sqrt{2}}; -\frac{1}{\sqrt{2}}\right]^T.$$

У цьому випадку квадрат норми кожного вектору дорівнює 1:

$$||u_1||^2 = u_1{}^T \cdot u_1 = \left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2 = 1$$

$$||u_2||^2 = u_2{}^T \cdot u_2 = \left(\frac{1}{\sqrt{2}}\right)^2 + \left(-\frac{1}{\sqrt{2}}\right)^2 = 1$$

отже, вони утворюють ортонормовану систему координат. Тепер будь-який вектор $\{x_k\}$, представлений спочатку у декартових осях f_1, f_2, можна уявити у новому просторі координатних осей u_1, u_2, використовуючи просте векторне-матричне множення:

$$[g_1(x_k)\, g_2(x_k)\,] = \left[\frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} \frac{-1}{\sqrt{2}}\right] \cdot [f_1(x_k)\, f_1(x_k)\,];$$

Або, за очевидних замін, у вигляді:

$$G_k = U \cdot F_k$$

Тобто ми запровадили для нашого конкретного випадку нове функціональне перетворення для ознак $f_1(x_k), f_2(x_k)$:

$$\{g_1(x_k) = \langle u_1, [f_1(x_k), f_2(x_k)]\rangle = \frac{1}{\sqrt{2}} f_1(x) + \frac{1}{\sqrt{2}} f_2(x) g_2(x_k) =$$

$$= \langle u_{21}, [f_1(x_k), f_2(x_k)]\rangle = \frac{1}{\sqrt{2}} f_1(x) - \frac{1}{\sqrt{2}} f_2(x)$$

Причому, оборотне, теоретично, ми можемо повернутися до вихідного простору ознак, використовуючи транспоновану матрицю власних векторів:

$$F_k = U^T \cdot G_k$$

Відмінно, ми з вами навчилися концентрувати інформацію про ознаки одного конкретного випадку. Але як це зробити для довільного розподілу об'єктів $\{x_i\}$?

Та й ще у багатовимірному просторі, коли у нас не дві, а n вихідних ознак:

$$F = [f_1(x_1) \dots f_n(x_1) \dots\dots\dots f_1(x_l) \dots f_n(x_l)]_{l \times n}.$$

Виявляється, зробити це досить просто. Для цього достатньо обчислити так звану матрицю Грама за ознаками:

$$F^* = \frac{1}{l} \cdot F^T \cdot F = \frac{1}{l} \cdot [f_1(x_1) \dots f_1(x_l) \dots\dots\dots f_n(x_1) \dots f_n(x_l)]_{n \times l} \cdot [f_1(x_1) \dots f_n(x_1) \dots\dots\dots f_1(x_l) \dots f_n(x_l)]_{l \times n}$$

$\smile$ матриця $n \times n$

А потім для неї знайти власні вектори і власні числа, використовуючи відоме рівняння з лінійної алгебри $\det(F^* - \lambda I) = 0$.

Мовою Python це можна зробити за допомогою пакета Numpy, командою:

L, *W* = np.linalg.eig(*FF*) # *L* – власні числа; *W* – власні вектори

В результаті отримуємо набір з n власних векторів $\{u_i\}$та, відповідних їм, *n* власних чисел $\{\lambda_i\}$:

$$\begin{array}{ccccccc} \lambda_1 & \geq & \lambda_2 & \geq & \dots & \geq & \lambda_n \\ \downarrow & & \downarrow & & \dots & & \downarrow \\ u_1 & & u_2 & & \dots & & u_n \end{array}$$

Власні вектори утворюють ортонормовану систему координат у *n*-мірному просторі, а власні числа дорівнюють дисперсіям проекцій образів на відповідні координатні осі.

Якщо відсортувати власні вектори щодо спадання власних значень, то на перший вектор будуть, в середньому, припадати найбільші проекції, на другий – найбільша інформація з решти, і так до *n*-го вектору. Таким чином, ми отримуємо систему координат, яка найкраще локалізує інформацію вихідного простору ознак *F*. Такий підхід отримав назву метод головних компонент (*PCA* – principal component analysis), який активно використовується в практиці машинного навчання.

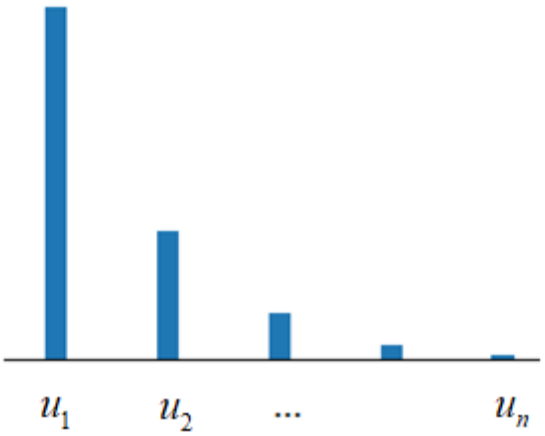


Рис. 5.13

Так вийшло, що цей алгоритм у минулому перевідкривали кілька разів у різних прикладних областях. В результаті, у нього сформувалося безліч різних назв. Наприклад, я вперше про нього дізнався як про перетворення Карунена-Лоева, коли працював над кандидатською дисертацією. Є й інші назви, але все це про те саме.

Щоб не перевантажувати вас інформацією, я розповім на наступному занятті, як за допомогою методу головних компонентів можна скорочувати простір ознак і усувати проблему мультиколінеарності ознак, яка є причиною перенавчання моделей

5.6. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Приведіть показники алгоритму precision (точність).
2. Приведіть показники алгоритму recall (повнота).
3. Дайте визначення гармонійного середнього (*F*-міра).
4. Дайте визначення характеристики якості класифікації моделі False Positive Rate (*FRR*) і True Positive Rate (*TRR*).
5. Дайте визначення Receiver Operating Characteristic (*ROC*-крива).
6. Дайте визначення методу головних компонентів (Principal Component Analysis – PCA).

5.7. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.
2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

Лекція №6. Алгоритми кластеризації

Результати навчання, які формуються:

ДРН4: Використовувати методи добування знань, кластеризації та класифікації.

ДРН7: Використовувати алгоритми побудови асоціативних та класифікуючих правил.

6.1. Завдання кластеризації. Постановка задачі

На попередніх заняттях ми познайомилися з декількома поширеними метричними підходами до завдань класифікації та регресії. Вони нам була дана навчальна вибірка (розмічені дані), якими будувалася модель. Але на практиці зустрічаються завдання, коли вихідний набір об'єктів X не розмічений і потрібно побудувати модель, яка б розносила образи за групами (класами), використовуючи задану (або вибрану) метрику між об'єктами:

$$\rho(x_i, x_j),\ \ i, j = 1, 2, \ldots, l$$

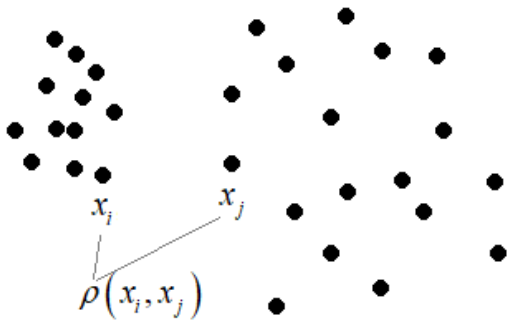


Рис. 6.1

Це постановка завдання кластеризації.

Тут на виході, в загальному випадку, потрібно знайти:

– безліч кластерів Y;

– алгоритм кластеризації $a : X \to Y$.

І, оскільки алгоритм має сам рознести образи за класами, то завдання кластеризації відносять до завдань навчання без вчителя (unsupervised learning). Тобто це приватне завдання навчання без вчителя. Крім кластеризації існують і інші методи, які також ставляться до навчання без вчителя. Тому не слід вважати, що кластеризація охоплює весь клас таких завдань.

Цілі кластеризації. Для чого взагалі може знадобитися кластеризувати дані? Тут можна виділити кілька напрямків:

- складні дані розбиваються за схожими групами, спрощуючи їх уявлення, а значить, і обробку в межах кожної групи (зазвичай використовується в задачах класифікації, регресії);

- скорочення обсягу вибірки за рахунок збереження невеликої кількості характерних представників кожного класу;

- виділення нетипових об’єктів (викидів), які не можна віднести до жодного кластера (завдання класифікації, антифрод та спам системи);

- побудова ієрархії об’єктів (приклад – класифікація тварин і рослин К. Ліннея – завдання таксономії).

Типи кластерних структур. Які на цьому шляху виникають проблеми? Перше. Часто завдання кластеризації немає чітко вираженого єдиного рішення (навіть з позиції людського сприйняття).

Наступний приклад показує, як по-різному можна розділити безліч точок по групах:

Рис. 6.2

Тут результат кластеризації може залежати і від вибору метрики і від алгоритму, що застосовується, і від початкових вихідних даних, які найчастіше визначаються випадковим чином. Тому, для тих самих даних можуть виходити зовсім різні результати кластеризації.

Друге. Самі кластери можуть утворювати різні типи структур. Ось найбільш характерні з них:

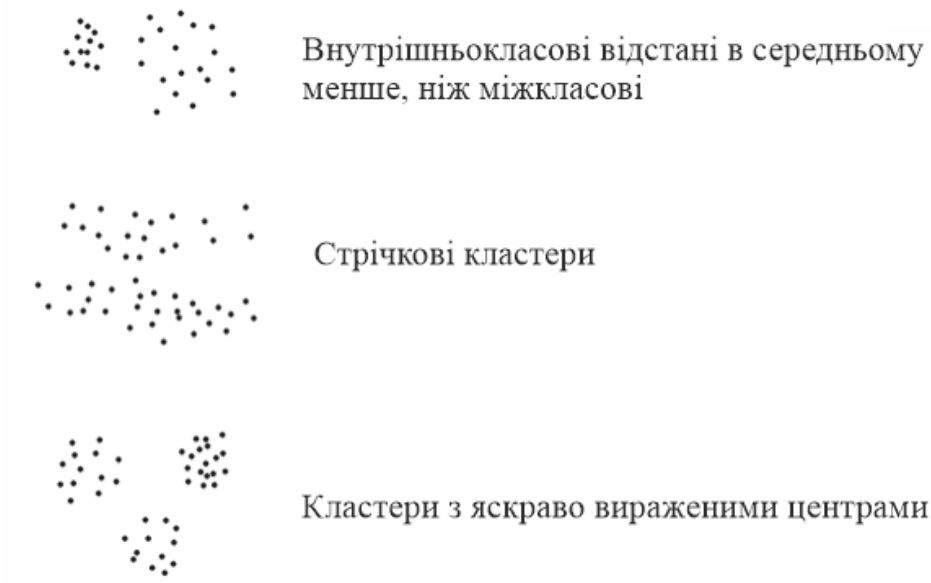


Рис. 6.3

Зверніть увагу, в останніх двох прикладах немає кластерів як таких. Геометричні фігури, як правило, це не той набір даних, який цікавить замовника (попри те, що наш мозок добре їх розрізняє). А нижче загалом представлений один кластер. За такого розподілу об'єктів вибірки завдання кластеризації ставити безглуздо. Тим не менш, на практиці нерідко буває, що замовник ставить завдання кластеризації для даних, які по всіх осях утворюють такий єдиний розподіл у вигляді одного кластера і виділити будь-які окремі структури практично неможливо. У цьому випадку завдання слід скоригувати та вирішувати іншими методами.

Критерії якості кластеризації.

Тепер, коли ми загалом представляємо завдання кластеризації, потрібно сформулювати критерії з метою оцінки якості розбиття образів за групами.

Якщо ми можемо вимірювати лише відстані між парами будь-яких образів: $\rho(x_i, x_j)$, $i, j = 1, 2, \ldots, l$ та маємо модель, яка видає номер кластера:

$a(x) \rightarrow$ номер кластера, то як критерії можна виділити середню внутрішньо кластерну відстань:

$$F_0 = \frac{\sum_{i<j}[a_i=a_j]\cdot\rho(x_i,x_j)}{\sum_{i<j}[a_i=a_j]} \rightarrow \min$$

(воно має прагнути до мінімуму) та середня міжкластерна відстань:

$$F_1 = \frac{\sum_{i<j}[a_i \neq a_j] \cdot \rho(x_i, x_j)}{\sum_{i<j}[a_i \neq a_j]} \to \min$$

(Очевидно, воно має бути якомога більше). Об'єднати ці дві характеристики можна, наприклад, взявши їхнє відношення:

$$\frac{F_0}{F_1} \to \min.$$

Якщо об'єкти описуються координатами в просторі ознак:

$$x_i = [x_i^1, x_i^2, \ldots, x_i^n]^T,\ i = 1, 2, \ldots, l$$

можна додатково ввести поняття центру кластера:

$$\mu_a = \frac{1}{l_a} \sum_{i, a_i = a} x_i,\ a \in X$$

та обчислювати F_0, F_1 щодо них:

$$\Phi_0 = \sum_{a \in Y} \frac{1}{l_a} \sum_{i, a_i = a} \rho(x_i, \mu_a) \to \min$$

$$\Phi_1 = \sum_{a,b \in Y} \rho(\mu_a, \mu_b) \to \max.$$

Об'єднати ці дві характеристики можна також шляхом їхнього ставлення:

$$\frac{\Phi_0}{\Phi_1} \to \min.$$

У вас тут може виникнути питання, а що це за об'єкти, для яких можна обчислити лише відстані між ними, але не центри кластерів? Що означає вони представлені у просторі ознак? Наприклад, це може бути відстані між словами. Саме собою слово – це об'єкт без числових ознак. Проте існують методики, що дозволяють визначати, наскільки одне слово близько (або далеко) від іншого слова.

Те саме можна робити і зі звуками, послідовностями нуклеотидів в ДНК і РНК тощо. Існує чимало завдань, де ми можемо ставити відстані (метрику), але не простір ознак. Або ж, нам окремі ознаки ні до чого і робити додаткову роботу просто не має сенсу. У всіх цих випадках можна скористатися характеристиками F_0, F_1 для оцінки якості кластеризації. Якщо ж у нас для образів явно прописані числові ознаки, то можна перейти до зрозуміліших (на мій погляд) оцінок якості Φ_0, Φ_1 .

Постановка завдання часткового навчання (Semi-Supervised Learning – SSL)

На закінчення цього вступного заняття сформулюю постановку завдання часткового навчання з учителем з урахуванням кластеризації. Припустимо, що у навчальній вибірці є невелика частина (k штук) розмічених об'єктів: $X^k = \{(x_1,y_1), (x_2,y_2),\ldots,(x_k,y_k)\}$ а, інші не розмічені: $U = \{x_{k+1}, x_{k+2},\ldots, x_l\}$ тобто ми не знаємо для них цільових значень y_i, $i = k+1, \ldots, l$.

Очевидно, частина розмічених даних нам може допомогти в кластеризації інших задач U нерозмічених об'єктів. Для цього ми будуємо модель: a: $U \to Y$, яка формує вихідні значення: a_i, $i = k+1,\ldots,l$ для нерозмічених даних. І тут розрізняють два загальні підходи до вирішення цього завдання:

Часткове навчання: будуємо алгоритм a такий, що для будь-якого вхідного образу x (н е обов'язково з навчальної вибірки) прогнозується клас (номер кластера).

Трансдуктивне навчання (transductive learning): будуємо алгоритм a такий, що тільки для об'єктів U формуються прогнози кластерів.

Де використовується такий підхід часткового навчання? Наприклад, при каталогізації текстів. Коли їх потрібно згрупувати за схожістю, то є співвіднести з раніше розміченими текстами (для яких є мітки категорії). Те саме стосується будь-яких типів даних, аж до зображень, звуків, послідовностей ДНК і т.д.

На цьому ми завершимо вступне заняття з кластеризації. На наступних продовжимо цю тему та розглянемо конкретні алгоритми поділу об'єктів за категоріями.

6.2. Алгоритм кластеризації Ллойда (*K*-середніх, *K*-means)

На цьому занятті розглянемо один із найпростіших алгоритмів кластеризації – алгоритм Ллойда (його ще називають методом *K*-середніх). Ідея його дуже проста.

Спочатку, у нас є набір об’єктів $X = \{x_1, x_2,\ldots, x_l\}$ кожен з яких представлений у n-вимірному числовому просторі ознак: $x_i = [x_i^1, x_i^2, \ldots, x_i^n]^T$

Наприклад, якщо кожен об'єкт має дві ознаки ($n = 2$), то безліч об’єктів можна подати у вигляді точок на площині:

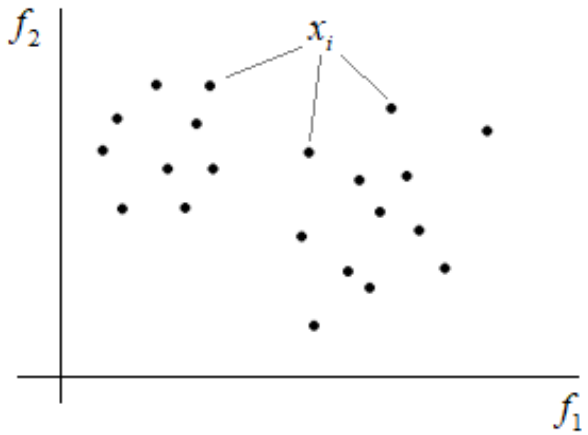


Рис. 6.4

Наше завдання розбити це безліч об’єктів за групами (кластерами). Для цього в алгоритмі Ллойда задається:

$K = |Y|$ – число кластерів; $Y = \{1, 2,\ldots, K\}$ – номери кластерів.

Так, ми спочатку повинні явно вказати алгоритму, на скільки груп слід розбивати об’єкти вибірки. Сам він не визначає кількість кластерів.

Визначається метрика – спосіб обчислення відстаней між довільними двома точками у просторі ознак $\rho(x_i, x_j)$.

Наприклад, часто використовують евклідову відстань:

$$\rho(x_i, x_j) = ||x_i - x_j||^2 = \sum_{k=1}^{n}(x_i^k - x_j^k)^2.$$

Для кожного кластера нам потрібно задати початкові центри. Це можна зробити або випадковим чином, або взяти будь-які K об'єктів і вважати їх центрами: μ_a, a O Y.

Після цього запускається ітераційний процес, у якому виконуються такі дії:

– кожен об’єкт x_i відноситься до найближчого центру μ_a відповідно до обраної метрики $\rho(x_i, \mu_a)$:

$a_i = \rho(x_i, \mu_a)$, $i = 1, 2, \ldots, l$

– робиться перерахунок нових положень центрів кластерів:

$\mu_a = \frac{\sum_{i=1}^{l}[a_i=a]x_i}{\sum_{i=1}^{l}[a_i=a]}$, a O Y

І так доти, доки положення центрів не змінюватимуться (або змінюються дуже незначно).

Алгоритм Ллойда при частково розміченій вибірці

Бувають завдання, у яких частина (k об’єктів) з навчальної вибірки розмічена, а інші $l – k$ – не розмічені:

$$T = \{x_1, x_2, \ldots, x_k\}$$
$$U = \{x_{k+1}, x_{k+2}, \ldots, x_l\}$$

Це варіант (приклад) завдання часткового навчання без учителя (*SSL*). І тут алгоритм Ллойда, загалом, зберігається, лише розбиття на кластери виконується для об'єктів безлічі U.

Спочатку ми також задаємо число кластерів $K = |Y|$ та формуємо центри кластерів: μ_a, a O Y.

Потім, у циклі виконуємо дві дії:

– кожен не розмічений об’єкт x_i O U відносимо до найближчого кластера (найближча відстань до центру): $a_i = arg\ \rho(x_i, \mu_a)$, $i = k+1, \ldots, l$

– перераховуємо центри кластерів по всій вибірці (з урахуванням розмічених об’єктів):

$$\mu_a = \frac{\sum_{i=1}^{l}[a_i=a]x_i}{\sum_{i=1}^{l}[a_i=a]}, \ a \text{ O } Y.$$

Як бачите, алгоритм практично той самий. Як невелике домашнє завдання, спробуйте його самостійно реалізувати.

Недоліки алгоритму Ллойда

Звичайно, основною перевагою алгоритму Ллойда є його простота (швидкість роботи). І щодо нескладних завдань він цілком придатний. Але є й очевидні вади:

- не визначає кількість кластерів за вибіркою (потрібно задавати вручну);

- розбиття на кластери залежить від початкового становища їх центрів (алгоритм може давати різні результати за різних початкових положень центрів);

- повільна збіжність центрів до рівноважного стану при великих обсягах даних (краще використовувати метод *k*-means ++);

- коректне розбиття вдається досягти лише для яскраво виражених гаусівських кластерних структур.

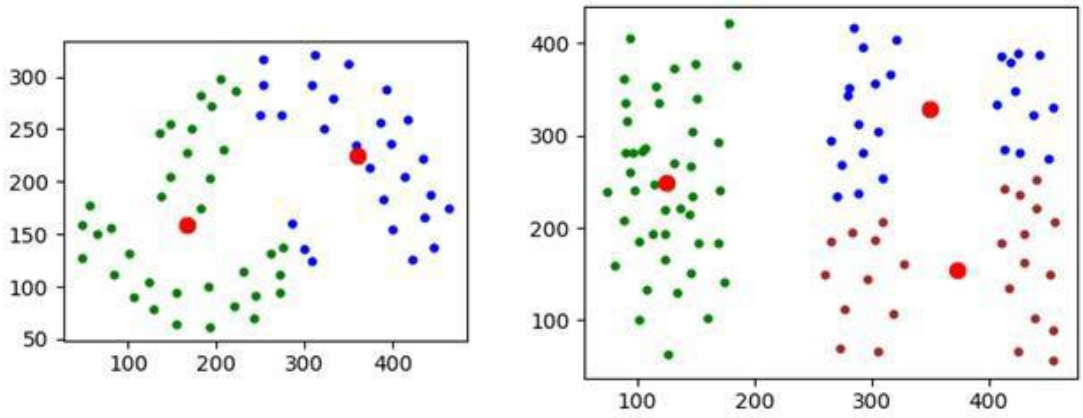


Рис. 6.6

Ось приклади невдалого розбиття об’єктів на кластери. Причина загалом одна – суттєва негаусовість розподілів точок у кожній групі. Тобто алгоритм *K*-середніх (*K*-means) намагається покрити вибірку «кульками» і звідси такий результат.

На наступному занятті ми розглянемо досконаліший алгоритм кластеризації DBSCAN, який позбавлений деяких недоліків алгоритму Ллойда і дає загалом кращі результати.

6.3. Алгоритм кластеризації DBSCAN

Продовжуємо розглядати типові алгоритми кластеризації і на цьому занятті йтиметься про досить популярний алгоритм DBSCAN: (Density-Based Spatial Clustering of Application with Noise).

Його перевагою перед алгоритмом *K*-means (Ллойда), про який ми говорили на попередньому занятті, в тому, що він дозволяє виділяти кластери довільної форми і, крім того, сам визначає кількість кластерів даних.

Незважаючи на те, що вперше DBSCAN був запропонований далекого 1996 року, він широко застосовується і понині. Це суто алгоритмічний підхід, ніяк не пов'язаний з теорією ймовірностей і щільністю розподілів даних. Він

виходить з досить розумних евристиках, придуманих і запропонованих авторами цього алгоритму.

Все починається з поняття епсилон-околиці об’єкта. Для будь-якого вектору x в метричному просторі ознак епсилон-околиця визначається як безліч точок, що віддаляються від x не більше ніж на ε:

$$U_z(x) = \{u \in U: \rho(x, u) \leq \varepsilon\},$$

де $\rho(x, u)$ – вибрана метрика простору ознак (наприклад, евклідова відстань).

Зрозуміло, параметр $\varepsilon > 0$ ми ставимо самі як параметр роботи алгоритму DBSCAN. Далі, в алгоритмі на основі епсилон-околиці об'єкти поділяються на три типи:

- кореневий: що містить не менше m об’єктів у своїй епсилон-околиці, $|U_z(x)| \geq m$;

- граничний: не кореневий, але в околиці кореневого;

- шумовий: (викид): не кореневий і не граничний.

Все це може здатися якимось дивним та заплутаним. Які типи об’єктів? Чому саме такі? Навіщо це взагалі потрібно?

Ось тому даний алгоритм і заснований на евристиках, а не математики, де все суворіше та чіткіше. Типи об’єктів – це одна з евристики. Зараз ви зрозумієте, як вони використовуються в алгоритмі та навіщо потрібні.

Припустимо, ми маємо деякий набір даних у двовимірному просторі ознак. Спочатку ми випадково вибираємо об'єкт x_i із цього набору. Якщо в околиці цього об’єкта менше m інших об'єктів, то він позначається як можливий шумовий. Потім, ми знову вибираємо випадковим чином об’єкт (виключаючи раніше розглянуті) і знову перевіряємо повноту його околиці. Якщо в ній знаходиться не менше m інших образів, то вектор позначається як кореневий і для всіх образів (крапок), що входять в околицю, процедура рекурентно повторюється. Причому, якщо об'єкт не містить достатньої кількості сусідів у своїй околиці, він позначається граничним, інакше – кореневим. В результаті ми перебираємо всі об'єкти, які захоплюються заданою ε-околицею. Таким чином формується кластер.

Далі, процес повторюється від початку, виключаючи раніше оброблені образи. Також випадково відбирається об'єкт і формується ще один кластер або шумові образи. У результаті, після проходження по всіх об’єктах вибірки, на виході ми отримуємо розбиття даних на кластери та шумові образи, які не увійшли до жодного кластера. Причому число кластерів було визначено автоматично, виходячи із заданих параметрів ε і m.

Псевдокод цього алгоритму можна записати наступним чином:

Вхід: вибірка $X^l = \{x_1,\ldots,x_l\}$, параметри ε та m

Вихід: розбиття вибірки на кластери та шумові викиди N

1: $U = X^l$ - непомічені; $N = Ж$, $a = 0$

2: поки у вибірці є непомічені точки, $U \neq Ж$:

3: вибирається випадковий образ $x \in U$

4: якщо $|U_z(x)| < m$ то

5: образ x позначається як можливий шумовий
6: інакше
7: створюється новий кластер $K = U_z(x)$; $a = a + 1$
8: для всіх $x' \in K$, не помічених або шумових
9: якщо $|U_z(x')| \geq m$ то $K = K \cup U_z(x')$
10: інакше помітити x' як граничний кластер K
11: $a_i = a$ для всіх $x_i \in K$
12: $U = U \setminus K$.

Переваги алгоритму DBSCAN швидка (за обсягом обчислень) реалізація для великих вибірок:

– $O(l \ln l)$ - для ефективних реалізацій алгоритму;
– виділяє кластери довільної форми;
– поділяє об’єкти на кореневі, граничні та шумові.

Такий поділ відіграє важливу роль, коли нам за набором даних потрібно визначити не тільки кластери, а й об'єкти з нетиповою поведінкою (шумові, викиди). Наприклад, вони можуть відповідати СПАМ-розсилкам, або неприродною поведінкою користувачів в онлайн-сервісах (відстеження ботів та зловмисників), визначення неприродності текстів, наприклад, згенерованих комп'ютером тощо. У всіх цих задачах алгоритм кластеризації нам може дуже допомогти як з групуванням даних за деякими ознаками, так і виділення дивних, нетипових об’єктів.

Щоб усе це було зрозуміліше, наведу ще один приклад із реального життя. У 90-х новосибірські соціологи вивчали причини масової міграції сільського населення міста. Під час опитування місцевих жителів Алтайського краю вони ставили близько сотні питань. У результаті утворився масив даних із 7000 анкет та у кожній по 100 питань. Перед вченими постало питання, як аналізувати отримані дані? І, звичайно ж, єдиним розумним рішенням було виконати угруповання (кластеризацію) спостережень за допомогою комп’ютера. Всі дані були переведені в цифровий вигляд, подані на вхід алгоритму кластеризації і на виході отримали дуже цікаві та закономірні результати у вигляді семи основних груп (таксонів).

Наприклад, було виділено групу жінок віком понад 60 років, діти яких живуть у містах. Зрозуміло, що ці жінки (під умовним найменуванням «бабусі») виїжджали до міста, щоб няньчити онуків. Були групи "демобілізовані солдати", "нареченої" та деякі інші. Ось приклад того, як алгоритм кластеризації дозволяє суттєво спростити аналіз даних, які виконати вручну було б дуже непростим заняттям

6.4. Агломеративна ієрархічна кластеризація. Дендрограмма.

Продовжуємо розглядати типові алгоритми кластеризації та на черзі алгоритм ієрархічної кластеризації. У чому суть такого підходу?

Припустимо, у нас є набір даних у числовому просторі ознак із заданою метрикою:

$$X^l = \{x_1, x_2, \ldots, x_l\}$$
$$\rho(x_i, x_j), \quad i, j = 1, \ldots, l$$

Для зручності я зображу їх як точки у двовимірному просторі (хоча, у випадку, простір має n розмірностей – за кількістю ознак). Далі, ми ієрархічно їх або об’єднуємо, або розбиваємо більш дрібні кластери. При об'єднанні спочатку кожен кластер містить по одному об’єкту, а потім, на кожній ітерації відбувається об’єднання двох найближчих кластерів, утворюючи все більші групи. Насправді, переважно використовують ідею послідовних об’єднань (агломерації) об’єктів. Звідси й походить назва агломеративна ієрархічна кластеризація.

Одним із перших учених, який зробив такий підхід до даних, був Карл Лінней, який намагався за своєю об'ємною картотекою рослин і тварин об'єднати їх за родами та видами:

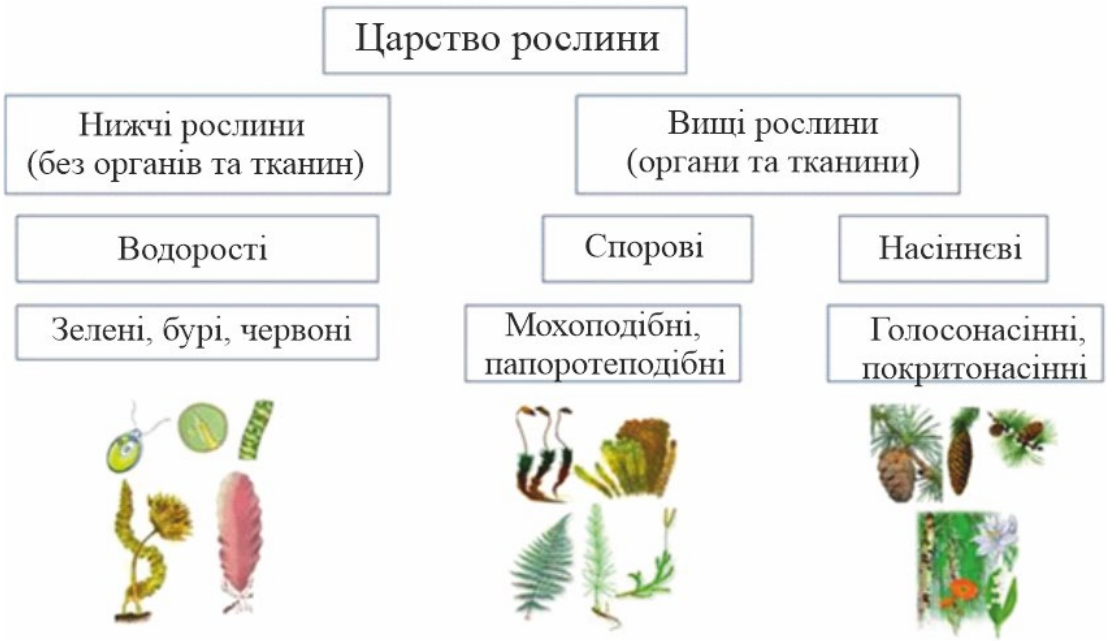


Рис. 6.7

Отримані групи об’єктів, близьких за певними ознаками, називають таксонами. Фактично, таксон і кластер – це те саме і я використовуватиму ці слова як синоніми.

Давайте докладніше розглянемо ідею агломеративної кластеризації, як найчастіше використовувану практично. Отже, спочатку маємо набір даних і кожен об’єкт – це незалежний кластер. Потім, відповідно до заданої метрики між об’єктами: $\rho(x_i, x_j)$, $i, j = 1, \ldots, l$, вибираємо два найближчих таксони U, V і об’єднуємо їх в один W: $W = U$ И V.

Цей крок ми легко виконали, оскільки вміємо обчислювати відстань між окремими об’єктами даних. Але тепер у нас з’явилася група з двох об’єктів. І

нам потрібно вміти обчислювати відстані від неї до решти об’єктів вибірки. Як це зробити?

Тут є безліч евристик:

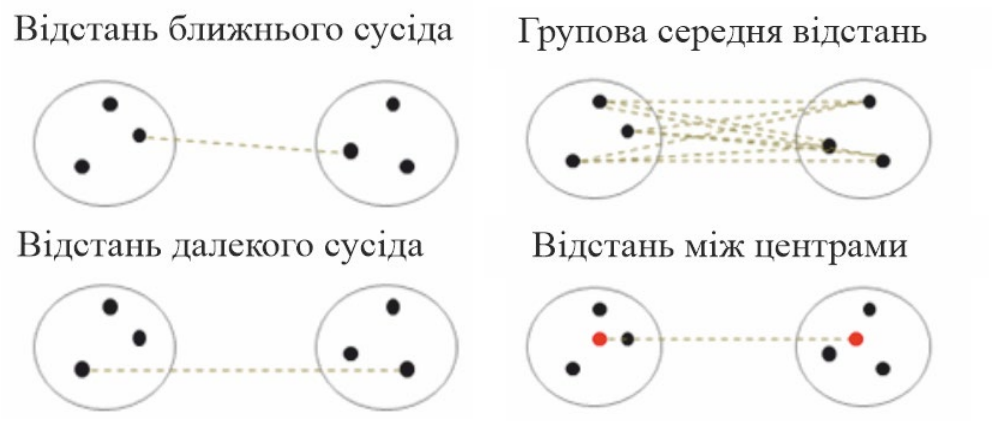


Рис. 6.8

Так от, виявляється, всі ці варіанти можна описати однією математичною формулою:

$$R_{WS} = \alpha_U R_{US} + \alpha_V R_{VS} + \beta R_{UV} + \gamma |R_{US} - R_{VS}|,$$

де α_U, α_V, β, γ – деякі параметри, що визначають вид метрики між кластерами.

Цей вираз називається формулою Ланса-Уільямса і для зазначених способів обчислень відстаней між таксонами, коефіцієнти набувають вигляду:

Відстань ближнього сусіда	$\alpha_U = \alpha_V = \frac{1}{2},\ \beta = 0,\ \ \gamma = -\frac{1}{2}$
Відстань далекого сусіда	$\alpha_U = \alpha_V = \frac{1}{2},\ \beta = 0,\ \ \gamma = \frac{1}{2}$
Групова середня відстань	$\alpha_U = \frac{\lvert U\rvert}{\lvert W\rvert},\ \alpha_V = \frac{\lvert V\rvert}{\lvert W\rvert},\ \beta = \gamma = 0$
Відстань між центрами	$\alpha_U = \frac{\lvert U\rvert}{\lvert W\rvert},\ \alpha_V = \frac{\lvert V\rvert}{\lvert W\rvert},\ \beta = -\alpha_U \cdot \alpha_V,\ \ \gamma = 0$
Відстань Уорд	$\alpha_U = \frac{\lvert S\rvert + \lvert U\rvert}{\lvert S\rvert + \lvert W\rvert},\ \alpha_V = \frac{\lvert S\rvert + \lvert V\rvert}{\lvert S\rvert + \lvert W\rvert},\ \beta = \frac{-\lvert S\rvert}{\lvert S\rvert + \lvert W\rvert},\ \ \gamma = 0$

Найчастіше практично використовують відстань далекого сусіда та відстань Уорда, вони призводять, як правило, до найкращих результатів кластеризації.

Давайте тепер детальніше розберемося, як працює формула Ланса Вільямса. Найпростіше показати її принцип на одному об’єднаному таксоні W з двома об’єктами та другого таксону S – з одним об’єктом:

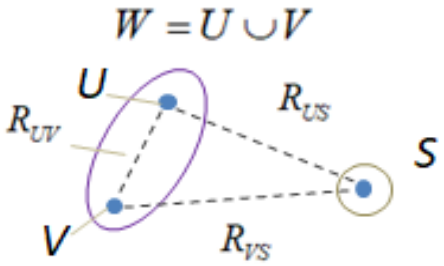


Рис. 6.9

Тут відзначено всі складові формули. Припустимо, що ми вибрали відстань ближнього сусіда, для якого:

$$\alpha_U = \alpha_V = \frac{1}{2},\ \beta = 0,\ \ \gamma = -\frac{1}{2}$$

та обчислення відстані між кластерами W та S відбувається, наступним чином:

$$R_{WS} = \frac{1}{2} R_{US} + \frac{1}{2} R_{VS} - \frac{1}{2} | R_{US} - R_{VS}|.$$

Дивіться, якщо тут розкрити модуль, то отримаємо:

$$R_{WS} = \{\frac{1}{2}R_{US} + \frac{1}{2}R_{VS} - \frac{1}{2}R_{US} + \frac{1}{2}R_{VS} = R_{VS},\ \ R_{VS} < R_{US}\ \ \frac{1}{2}R_{US} + \frac{1}{2}R_{VS} - \frac{1}{2}R_{US} - \frac{1}{2}R_{VS} = R_{US},\ \ R_{US} < R_{VS}$$

А це не що інше, як вибір мінімальної відстані між точками таксонів. В результаті ми знайшли відстань між таксоном W і таксоном S. Потім, коли нам доведеться таксон з двох об’єктів об’єднувати з іншим таксоном, то при обчисленні відстані об’єднаного кластера W ми будемо використовувати раніше обчислену відстань R_{VS} для отримання нового R_{WS} вже для кластера із трьох об’єктів:

$$W = U \text{ И } V$$

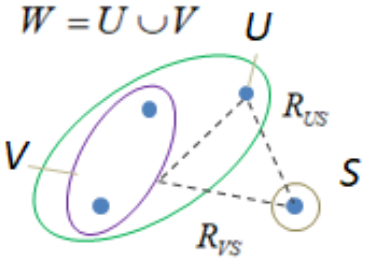


Рис. 6.10

Тобто U, V і S тут можуть бути не лише окремими точками, але, в міру об'єднання (роботи алгоритму), і кластерами з довільними числом точок. Завдяки рекурентному обчисленню відстаней між таксонами, що об'єднуються, і всіма іншими, ми, на кожній ітерації алгоритму, маємо повну інформацію про відстані між будь-якими парами сформованих кластерів.

Це ключовий момент роботи алгоритму агломеративної ієрархічної кластеризації. За аналогією працюють і всі інші метрики з іншими значеннями коефіцієнтів α_U, α_V, β, γ.

В результаті отримуємо наступний алгоритм ієрархічної кластеризації:

1: $C_1 = \{\{x_1\},\ldots,\{x_l\}\}$, – безліч одноелементних кластерів

$R_{\{x_i\}\cdot\{x_l\}} = \rho(x_i, x_j)$ – метрика (відстань) між окремими об'єктами

2: для всіх $t = 2, \ldots, l$ (t – номер ітерації)

3: знайти в C_{t-1} пару кластерів U, V з мінімальною відстанню R_{UV}

4: об'єднати їх в один кластер:

$$W = U \text{ И } V$$

$$C_t = C_{t-1} \text{ И } \{W\} \setminus \{U, V\}$$

5: для всіх S O C_t

6: обчислити R_{WS} за формулою Ланса-Уїльямса:

$$R_{WS} = \alpha_U R_{US} + \alpha_V R_{VS} + \beta R_{UV} + \gamma |R_{US} - R_{VS}|.$$

Дендрограма. Для візуалізації процесу ієрархічної кластеризації та оцінки її якості часто будують графік спеціального виду, який отримав назву дендрограма.

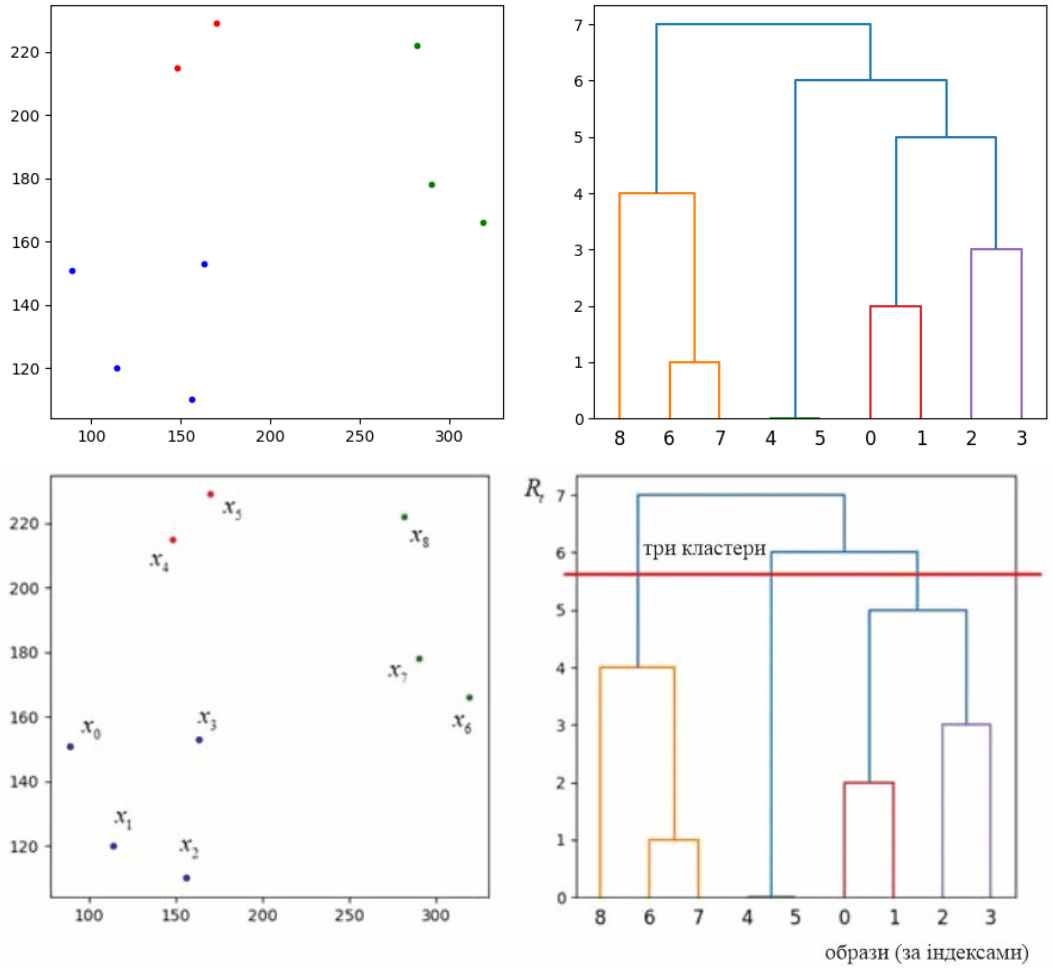


Рис. 9.11

По вертикалі відкладається мінімальна відстань R_t між кластерами, а горизонталлю – вихідні дані (об'єкти). Дивлячись на цей графік, ми відразу бачимо, в якому порядку відбувалося об'єднання даних у групи і наскільки кластери відокремлені один від одного по мінімальній відстані.

Хороший алгоритм ієрархічної кластеризації повинен давати дендограму без внутрішніх перетинів і з яскраво вираженими мінімальними відступами між кластерами, що формуються. Дендограма малюнку відповідає цим критеріям. Звичайно, при виборі інших метрик та способів обчислення відстаней між кластерами, дендограма змінюватиметься.

Крім оцінки якості, цей графік показує нам, де можна провести рівень відсікання для отримання певної кількості вихідних кластерів. Це дуже зручно, особливо якщо ми наперед не знаємо, на скільки таксонів слід розбити вхідні дані. Дендограма може допомогти побачити явні різницю між різними групами різних рівнях ієрархії.

Отже, загальний висновок можна зробити, таке:

- дендограма повинна мати монотонний характер (не мати внутрішніх перетинів), тобто відстані між кластерами, що об'єднуються, повинні постійно збільшуватися $R_2 \leq R_3 \leq \ldots \leq R_i$. Існує теорема (Мілліган, 1979), яка стверджує, що дендограма буде монотонною, якщо коефіцієнти у формулі Ланса-Уільямса підкоряються, таким умовам:

$$\alpha_U \geq 0,\ \alpha_V \geq 0,\ \alpha_U + \alpha_V + \beta \geq 1,\ \min\{\alpha_U, \alpha_V\} + \gamma \geqslant 0.$$

- при стискаючій відстані: $R_t \leq \rho(\mu_U, \mu_V),\ \forall t$ (коли на кожній наступній ітерації відстані між центрами кластерів зменшуються), ми отримуватимемо внизу дендограми розріджені кластери, а вище до кореня – густіші;

- при розтягуючій відстані $R_t \geq \rho(\mu_U, \mu_V),\ \forall t$.

Навпаки, внизу будуть утворюватися густі кластери, а вище до кореня – більш розріджені (зазвичай прагнуть до відстані, що розтягує, тому що це добре узгоджується з поданням правильної роботи алгоритмів кластеризації).

Рекомендації щодо агломеративної ієрархічної кластеризації

1. При виборі метрики відстаней між кластерами краще почати з відстані Уорда, а потім пробувати інші, наприклад, часто наступним беруть відстань далекого сусіда.

2. Будують кілька варіантів кластеризації (з різними метриками) і по дендограмі вибирають найкращий варіант (чисто візуально).

3. Результуюче число кластерів визначають за рівнем, де максимально зміна мінімальної відстані R_t.

6.4. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Назвіть і перелічіть цілі кластеризації.
2. Назвіть і перелічіть типи кластерних структур.
3. Визначте критерії якості кластеризації.
4. Дайте визначення алгоритму кластеризації Ллойда (*K*-means).

5. Переваги і недоліки алгоритму Ллойда.

6. Дайте визначення алгоритму кластеризації.

7. Дайте визначення алгоритму Density-Based Spatial Clustering of Application with Noise (DBSCAN).

8. Переваги і недоліки алгоритму DBSCAN.

9. Дайте визначення агломеративної ієрархічній кластеризації.

10. Дайте визначення сутності дендрограми.

6.5. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.

2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA

3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

Лекція №7. Методи класифікації

Результати навчання, які формуються:

ДРН4: Використовувати методи добування знань, кластеризації та класифікації.

7.1. Багатокласова класифікація. Методи one-vs-all та all-vs-all

Досі ми з вами розглядали лише двокласову (бінарну) класифікацію. Однак, на практиці чимало завдань, коли потрібно зарахувати об'єкт до одного з M класів. Наприклад, визначення марок машин за їхніми зображеннями, або виду тварин за їх характеристиками тощо. Про способи вирішення таких завдань ми з вами поговоримо на цьому занятті.

Найочевидніший підхід – це взяти будь-який алгоритм бінарної класифікації та розширити його на M класів. Тут існує безліч стратегій, але найпоширеніші дві, відомі під назвами:

– one-vs-all (один проти всіх) – у ScLearn one-vs-rest;

– all-vs-all (все проти всіх).

У першому випадку (one-vs-all) ми будуємо M алгоритмів, які відокремлюють один певний клас від решти:

$$\{a_1(x_i, w_1) = +1 \to x_i \in C_1 \dots a_k(x_i, w_k) = +1 \to x_i \in C_k \dots a_M(x_i, w_M) = +1 \to x_i \in C_M$$

Наприклад, якщо намалювати в двовимірному просторі ознак об'єкти трьох класів ($M = 3$), то кожен алгоритм визначатиме свою роздільну гіперплощину, для виділення одного класу від решти:

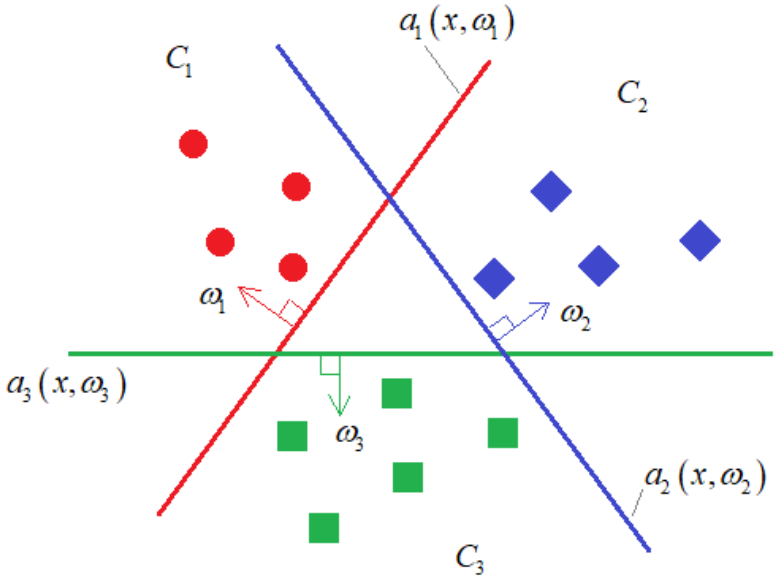


Рис. 7.1

Причому вектори ваг w_1, w_2, w_3 підбираються так, щоб позитивне значення класифікатора відповідало належності об'єкта до певного класу:

$$a_k(x, w_k) = sign(g_k(x, w_k, w_{0k})) = \{+1 \to C_k \; -1 \to \textit{все інше}$$

Зокрема, для лінійних моделей:

$$g_k(x, w_k, w_{0k})) = \langle x, w_k \rangle - w_{0k}.$$

Після того, як моделі навчені, вибір робиться на користь класу з найбільшим позитивним відступом від межі:

$$a(x) = arg\ g_k(x, w_k, w_{0k}) .$$

І, зверніть увагу, щоб цей багатокласовий висновок працював коректно, довжини векторів $\{w_i\}$ повинні збігатися (цю умову можна записати також через квадрат норми):

$$||w_1||^2 = ||w_2||^2 = \ldots = ||w_M||^2.$$

Ось принцип роботи стратегії один проти всіх (one-vs-all). Головним його недоліком є те, що ми навчаємо кожну модель $\{a_k(x, w_k)\}$ незалежно від інших моделей. Це дещо погіршує кінцевий результат багатокласової класифікації.

Стратегія all-vs-all (все проти всіх). Щоб якось подолати нестачу попередньої стратегії, була запропонована інша, в якій моделі виконують бінарну класифікацію лише для кількох класів:

	1	2	...	M
1	0	$a_{12}(x, \omega_{12})$	...	$a_{1M}(x, \omega_{1M})$
2	$a_{21}(x, \omega_{21})$	0	...	$a_{2M}(x, \omega_{2M})$
...	...	...	...	...
M	$a_{M1}(x, \omega_{M1})$	$a_{M2}(x, \omega_{M1})$	...	0

(Тут M – число класів).

Тобто модель, що розділяє два класи C_i, C_j, формує наступні вихідні значення:

$$a_{ij}(x,w_{ij}) = \{+1,\ x \in C_i \ -1,\ \ x \in C_j$$

Причому при правильному виборі моделей, очевидно, має виконуватися умова:

$$a_{ij}(x,w_{ij}) = -a_{ji}(x,w_{ji}), \quad i,j = 1, \ldots, M,$$

оскільки це, власне, та сама, лише «перевернута» поділяюча гіперплощина.

З урахуванням цієї рівності ми отримуємо $\frac{M \cdot (M-1)}{2}$ унікальні моделі.

Потім підсумковий результат класифікації для M класів визначається шляхом голосування (за мажоритарним принципом):

$$a(x) = arg\ \sum_{i=1}^{M} \sum_{j=1\ j \neq i}^{M} [a_{ij}(x) = k].$$

Багатокласова логістична регресія. Для всіх цих алгоритмів багатокласової класифікації можна застосувати формулу SoftMax:

$$P(a(x) = y_k) = \frac{exp(g_k(x,w_k))}{\sum_{j=1}^{M} exp(g_j(x,w_j))},$$

для оцінки ймовірності приналежності прогнозу моделі *k* класу. Однак тут слід бути акуратними з інтерпретацією отриманих значень ймовірностей.

Як ми з вами вже говорили (коли розглядали метод опорних векторів), ці ймовірності загалом відповідають реальним даним для логістичної регресії (за логарифмічної функції втрат). Якщо модель реалізує будь-який інший алгоритм бінарної класифікації, то значення функції SoftMax можуть сильно розходитися з їх імовірнісною інтерпретацією.

Тому, якщо вам дійсно потрібно отримати ймовірнісні характеристики, слід використовувати багатокласову логістичну регресію, яку можна отримати шляхом мінімізації наступного функціоналу:

$$Q(a, X^l) = \sum_{i=1}^{l} \sum_{k=1}^{M} [y_i = k] \log\Big(1 - y_i \cdot (\langle w_k, x_i \rangle - w_{0k})_{\text{відступ } M_i}\Big) \to \min_{w_k, w_{0k}}$$

Забігаючи наперед, зазначу, що завдання поділу образів на кілька класів досить добре вирішується за допомогою нейронних мереж. Проте, нейронні мережі має сенс застосовувати лише тих завдань, де вони справді ефективні проти іншими алгоритмами.

Насамперед, це завдання з вторинними ознаками, що важко формалізуються, наприклад, класифікація зображень, звуків, аналіз тексту та інше. Якщо ознаки чітко визначають рішення поставленого завдання, то класичні методи, зазвичай, простіше (з обчислювальної погляду) і призводять до подібним результатам, як і нейронні мережі.

Метрики якості

На закінчення цього заняття скажу кілька слів про те, як оцінювати якість моделей за багатокласової класифікації.

Найпоширеніша метрика accuracy (частка вірних класифікацій) визначається абсолютно за тією ж формулою, що і для бінарного випадку:

$$accuracy = \frac{1}{l} \sum_{i=1}^{l} [a(x_i) = y_i].$$

Але, як ми знаємо, вона дає повну характеристику моделі.

Для повнішої картини можна сформувати так звану матрицю помилок класифікації:

Таблиця 7.1

	$y = 1$	…	$y = M$
$a(x) = 1$	q_{11}	…	q_{1M}
…	…	…	…
$a(x) = M$	q_{M1}	…	q_{MM}

Тут q_{ij} – число відповідей моделі класу i, коли насправді має місце клас j: $q_{ij} = \sum_{k=1}^{l}[a(x_k) = i]\,[y_k = j]$.

По цій матриці можна побачити, як класифікатор поводиться на різних класах і скільки робить помилок.

Узагальнення precision, recall, *AUC-ROC* за кількох класів.

Зрештою, ми можемо застосувати відомі нам метрики precision, recall та *AUC-ROC* для багатокласового класифікатора $a(x)$.

Для цього спочатку приводять загальну модель до бінарного виду для аналізу класифікації k-го класу:

$$\{y_i^k = 2 \cdot [y_i = k] - 1 \; a^k(x) = [a(x) = k] \qquad k = 1,2,\ldots, M$$

А потім для кожного класу обчислюють характеристики:

$$TP_k, FP_k, TN_k, FN_k, \; k = 1,2,\ldots, M.$$

Після цього роблять мікро-усереднення цих показників:

$$\underline{TP} = \frac{1}{M}\sum_{k=1}^{M} TP_k;\; \underline{FP} = \frac{1}{M}\sum_{k=1}^{M} FP_k;\; \underline{TN} = \frac{1}{M}\sum_{k=1}^{M} TN_k;\; \underline{FN} = \frac{1}{M}\sum_{k=1}^{M} FN_k;$$

з наступним обчисленням *precision* та *recall*:

$$precision = \frac{\underline{TP}}{\underline{TP} + \underline{FP}};\; recall = \frac{\underline{TP}}{\underline{TP} + \underline{FN}};$$

Або виконують макро-усереднення:

$$precision_k = \frac{TP_k}{TP_k + FP_k};\; recall_k = \frac{TP_k}{TP_k + FN_k};$$

а вже потім обчислюють середні показники:

$$\underline{precision} = \frac{1}{M}\sum_{k=1}^{M} precision_k$$

$$\underline{recall} = \frac{1}{M}\sum_{k=1}^{M} recall_k.$$

Відмінності в мікро-і макро-усереднення досить прості. З формул добре видно, якщо класи об'єктів у навчальній вибірці не збалансовані, то при мікро-усереднення вони будуть «губитися». Тоді як при макро-усередненні вони матимуть приблизно ту саму вагу, що й великі (за кількістю об'єктів) класи. Тобто якщо нам важливі, при оцінці моделі, і великі та малі класи, то краще використовувати макро-усереднення. Якщо ж, головним чином, аналізуємо великі класи, то мікросереднення.

Якщо ж вибірка збалансована, то відмінностей між цими особливими підходами немає.

7.2 Метричні методи класифікації. Метод *k* найближчих сусідів

На цьому занятті почнемо розглядати метричні методи класифікації. Спочатку невеликий, короткий вступ. В 1936 Рональд Фішер, фахівець з математичної статистики, представив роботу з аналізу даних квітів ірисів.

Дані були зібрані ботаніком Едгаром Андерсоном, а Фішер знайшов спосіб за цими даними класифікувати три види ірисів:

- ірис щетинистий (iris bristly);
- ірис віргінський (iris virginica);
- ірис різнокольоровий (iris versicolor).

Іриси Фішера, як їх стали пізніше називати, складаються з даних про 150 екземплярів та результатів, наступних вимірів:

- довжина зовнішньої частки оцвітини (sepal length);
- ширина зовнішньої частки оцвітини (sepal width);
- довжина внутрішньої частки оцвітини (petal length);
- ширина внутрішньої частки оцвітини (petal width).

Ці дані можна як чотиривимірний простір ознак, у якому представлені об’єкти трьох класів (трьох видів ірисів). Для їхньої візуалізації використовується так звана діаграма розсіювання ірисів Фішера.

Це відображення всіх попарних наборів ознак:

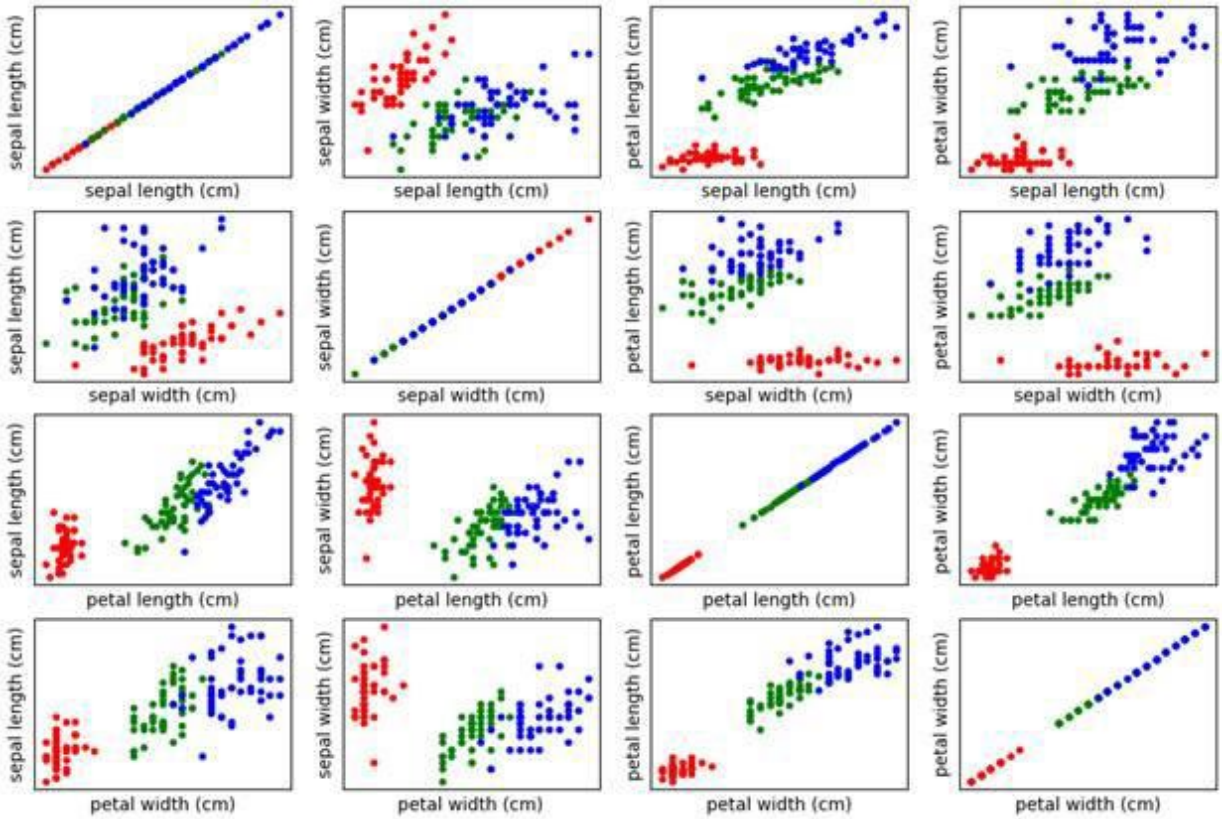


Рис. 7.2

З цих малюнків добре видно, що об’єкти (результати вимірювань) одного й того класу лежать, як правило, близько один до одного. І це досить часто ситуація. Надалі було висунуто гіпотезу компактності, яка говорить:

Координати образів одного й того класу в просторі ознак концентруються в геометрично близькі точки, утворюючи «компактні» згустки.

Це саме гіпотеза. Не всім об’єктів вона виконується. Ми це можемо лише припускати. Але якщо аналіз простору ознак показує, що гіпотеза

компактності, загалом, виконується, то завдань класифікації можна застосовувати, звані, метричні методи.

Метричні методи – це методи, що ґрунтуються на відстанях (заходи близькості) конкретного образу x до образів інших класів у просторі ознак. Наприклад, у нас є результат виміру (ромб). Тоді було б логічно віднести його до того виду (групи точок), до якого він найближчий:

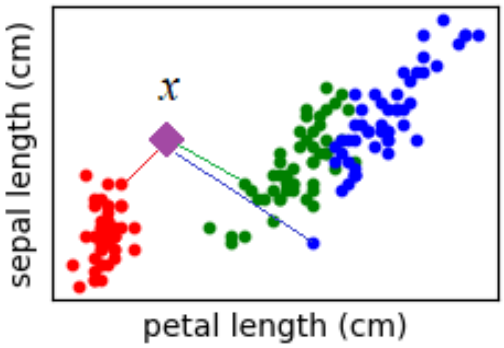


Рис. 7.3

Але тут одразу виникають дві проблеми:

- як вимірювати відстані між об’єктами у просторі ознак;
- як визначати на основі обчислених відстаней належність об’єкта до тієї чи іншої групи точок.

На ці питання немає єдиної, загальної відповіді. Тому існує безліч різних метричних методів класифікації.

Метрики у просторі ознак.

Почнемо з першого питання – способу визначення відстані між двома будь-якими векторами в n-мірному просторі ознак $\mathbb{R}^n$. Припустимо, ми маємо вектори об’єктів розмічених даних: $x_i = [x_i^1, x_i^2, \ldots, x_i^n]^T$, $i = 1, 2, \ldots, l$

та нові результати вимірювань, які слід класифікувати: $x_i = [x^1, x^2, \ldots, x^n]^T$

Тоді, перше, що спадає на думку, взяти евклідову відстань між двома точками:

$$\rho(x, x_i) = \sqrt{\sum_{j=1}^{n} \left(x^j - x_i^j\right)^2} = \left(\sum_{j=1}^{n} \left|x^j - x_i^j\right|^2\right)^{\frac{1}{2}}.$$

І така метрика справді застосовується на практиці. Але чи єдина вона можлива? Звичайно, ні.

У найпростішому випадку ми можемо узагальнити евклідову метрику, таким чином:

$$\rho(x, x_i) = \left(\sum_{j=1}^{n} w_j \cdot \left|x^j - x_i^j\right|^p\right)^{\frac{1}{p}},$$

де $p > 0$ – параметр метрики; $w_1, w_2, \ldots, w_n$ – ваги ознак.

Такий загальний спосіб обчислення відстаней називається метрикою Мінковського.

Навіщо потрібні ваги у цій формулі? Справа в тому, що ознаки можуть мати різний масштаб. Наприклад, вага золота в грамах і його вартість у рублях – це два абсолютно різні діапазони значень. Щоб їх однаково враховувати щодо близькості векторів, слід нормувати через підбір вагових коефіцієнтів.

Інший параметр p задає поведінку метрики у різних напрямках. Це найпростіше показати на малюнках еквідистантних кривих:

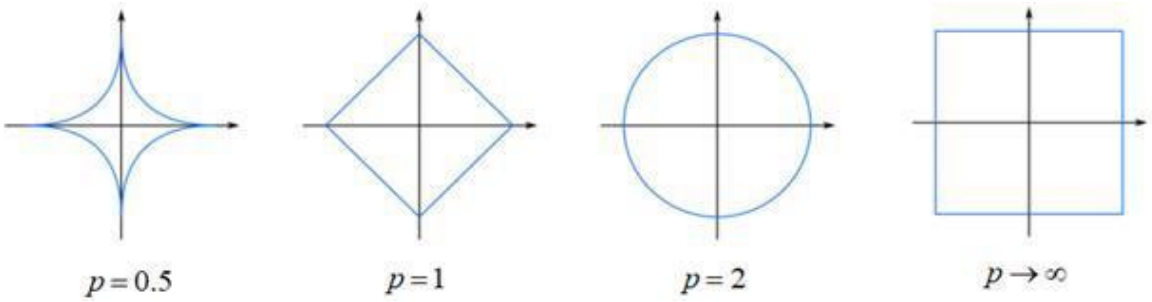


Рис. 7.4

Як бачимо, при p = 1 маємо аналог суми різниць модулів координат, при $p = 2$ – класична евклідова відстань, а якщо $p \to \infty$, то коло вироджується у квадрат. Найчастіше практично можна побачити метрику з модулями і квадратами (при $p = 1$ і 2).

Узагальнений метричний класифікатор.

Після того, як введена метрика, ми можемо будувати алгоритм класифікації на основі критерію близькості деякого об'єкта x O X до розмічених образів $\{x_i\}$ (її можна сприймати як навчальну вибірку). Для цього зручно усі об'єкти вибірки впорядкувати за зростанням відстані щодо x спільно з цільовими значеннями:

$$\rho(x, x^{(1)}) \le \rho(x, x^{(2)}) \le \cdots \le \rho(x, x^{(l)})$$
$$y^{(1)}, y^{(2)}, \ldots, y^{(l)}$$

(Верхній індекс – новий порядок спостережень у вибірці). Тоді математично класифікатор у найзагальнішому вигляді можна записати, таким чином:

$$a(x, X^l) = \sum_{i=1}^{l} [y^{(i)} = y] \cdot w(i, x) ,$$

де $w(i, x)$ – вага (ступінь важливості) i-го об'єкта по відношенню до x, пов'язаний з метрикою $\rho(x_i, x)$. Величина $w(i, x)$ невід'ємна та не зростає при збільшенні i.

Фактично, тут, ми обчислюємо загальну вагу $\sum_i w(i, x)$ для кожного класу та вибираємо клас із найбільшою сумою.

Метод *k* найближчих сусідів (*k* nearest neighbors, *kNN*)

Для кращого розуміння конкретизуємо цю формулу для найпростішого випадку, коли ми вважаємо: $w(i, x) = [\, i \le k]$.

(Пам’ятаємо, що образи впорядковані за зростанням відстаней щодо вектору x). Тобто, у нас i-й вага набуває одиничного значення для найближчої точки, і нульові – для всіх інших $w(i, x) = \{1, \rho(x_i, x) \quad 0, \text{інаке}$

У цьому випадку модель можна записати у вигляді: $a(x, X^l) = y^{(1)}$.

Тобто ми відносимо об’єкт x до класу об’єкту x_i з мінімальною відстанню відповідно до обраної метрики $\rho(x_i, x)$:

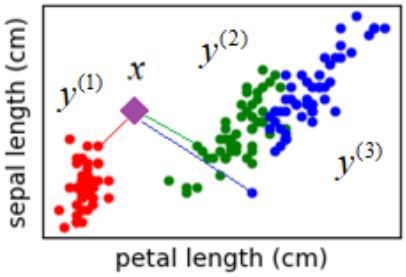


Рис. 7.5

Цей метод відомий під назвою метод найближчого сусіда. Очевидно, він має істотний недолік. Якщо серед образів одного класу випадково виявиться представник іншого класу, до якого відстань мінімальна, то класифікатор зробить неправильний висновок:

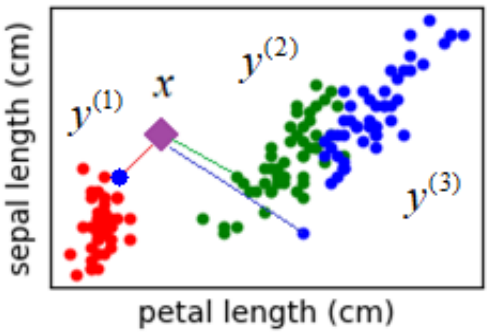


Рис. 7.6

Подолати такі поодинокі викиди відносно просто, якщо враховувати не одного, а найближчих сусідів. В цьому випадку вектор ваг можна визначити за правилом: $w(i, x) = [i \leq k]$.

Тобто тепер ваги $w(i, x)$ прийматимуть одиничні значення для перших k найближчих векторів, а решта залишатиметься нульовими. В результаті ми приходимо до методу k найближчих сусідів із наступним алгоритмом класифікації: $a(x, X^l) = \sum_{i=1}^{k} [y^{(i)} = y]$.

За простим, ми віддаємо перевагу тому класу, для якого число з k найближчих сусідів найбільше:

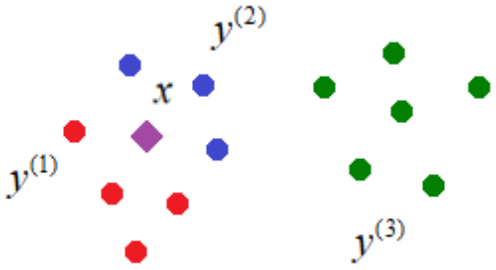


Рис. 7.7

Наприклад, якщо в околиці об’єкта x сім образів, причому 4 відносяться до класу $y^{(1)}$, а 3 – до класу $y^{(2)}$, то робимо висновок, що об’єкт x належить першому класу, т.к. його представників більше **Додаток 2**.

Недоліки методів найближчих сусідів

Звичайно, тут одразу видно нестачу такого підходу: що робити, якщо представників обох класів буде порівну? Якому класу тоді віддати перевагу?

Можна спробувати взяти непарне число k тоді для двох класів ця проблема буде вирішена. Але ж на околиці може виявитися і три і чотири різних класів. Тому ця загальна проблема існує і не має однозначного рішення.

Іншим важливим недоліком є одиничні значення ваги для найближчих k образів:

$$w(i, x) = 1, i \leq k.$$

Тобто ми не враховуємо відстані до сусідніх об’єктів, лише їх наявність чи відсутність. А, напевно, було б логічно оперувати, зокрема, й відстанями.

Далі ми з вами познайомимося з іншими метричними алгоритмами класифікації, які вирішують ці проблеми, хоча породжують деякі інші. Зрештою, універсальних рішень немає.

Переваги методу *k* найближчих сусідів

Незважаючи на очевидні недоліки, метод k найближчих сусідів набув широкого поширення і в ряді завдань показує хороші результати. Головною його перевагою є простота реалізації. Нам достатньо виділити найближчих об'єктів і виділити клас з максимальним числом представників. Обчислювально, це дуже просто. Тут навіть немає як такого алгоритму навчання. Все, що нам потрібно – це мати масив розмічених даних (представників класів) і за ними, потім відносити нові об'єкти до того чи іншого класу. Це називається lazy learning (ледаче навчання).

Завдяки простоті реалізації, ми можемо на навчальній вибірці застосувати техніку ковзного контролю leave-one-out (*LOO*) для знаходження найкращого параметра k. Я нагадаю, що при *LOO* ми послідовно прибираємо з вибірки по одному образу і оцінюємо якість його прогнозування обраною моделлю (алгоритму класифікації) по вибірці, що залишилася. Оскільки наша модель методу k найближчих сусідів залежить від параметра k: $a(k, x, X^l)$ то його можна підбирати за методом *LOO* так, щоб алгоритм робив якнайменше помилок: $LOO\ (k, X^l) = \sum_{i=1}^{l} [a(k, x_i, X^l \backslash \{x_i\} \neq y_i] \rightarrow min_k$

Навіть для великих вибірок у сотні тисяч і мільйон спостережень можна перебрати цілі значення k, скажімо, в діапазоні від 1 до 100 і вибрати той, який дасть мінімум функції *LOO*.

Я, сподіваюся, з цього заняття ви зрозуміли, що собою в цілому представляють метричні методи класифікації і як працює алгоритм k найближчих сусідів.

7.3. Методи парзенівського вікна та потенційних функцій

Ми з вами розглянули найпростіший алгоритм метричної класифікації найближчих сусідів. І відзначили суттєву нестачу його вагових коефіцієнтів $w(i, x)$: $a(x, X^l) = \sum_{i=1}^{l} [y^{(i)} = y] \cdot w(i, x)$ приймають лише два значення:

$$w(i, x) = [i \le k] = \{1, i \le k\ 0, i > k\ .$$

Я нагадаю, що індекс i означає i-й образ у впорядкованій за зростанням відстаней послідовності щодо вектору, що класифікується x:

$$\rho(x, x^{(1)}) \le \rho(x, x^{(2)}) \le \cdots \le \rho(x, x^{(l)})$$
$$y^{(1)}, y^{(2)}, \ldots, y^{(l)}$$

Тобто формула $w(i, x) = [i \le k]$ просто виділяє k найближчих сусідів для вектору x. Але, мабуть, було б логічно враховувати сусідні об'єкти з урахуванням відстані $\rho(x, x^{(i)})$ до них: чим вона більша, тим менша вага повинна мати дане спостереження.

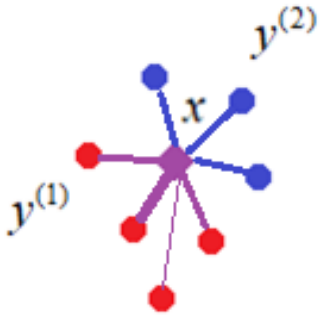


Рис. 7.8

(Близькі сусіди несуть більший внесок, ніж далекі).

Для реалізації цієї ідеї ваги $w(i, x)$ пропонується вибирати з використанням деякої функції, що не зростає $K(r)$ (її ще називають ядром):

$$w(i, x) = K\left(\frac{\rho(x, x^{(i)})}{h}\right).$$

Причому часто її вибирають обмеженою (фінітною) на інтервалі [0; 1] та симетричною щодо нуля (парної):

$$K(r) \ne 0, \quad r \text{ O } [0;1].$$

Такі ядра отримали назву **вікно Парзена чи парзенівські вікна**.

Для вікон параметр h у формулі ваги визначає область охоплення об’єктів щодо вектору x (з центром у точці x). Наприклад, якщо $h = 1$, то максимальна ненульова відстань дорівнюватиме одиниці. Якщо h взяти рівним 4 то відстань (охоплення) також збільшиться в 4 рази.

Залишається питання, яку функцію ядра взяти? Часто на практиці використовують такі варіанти:

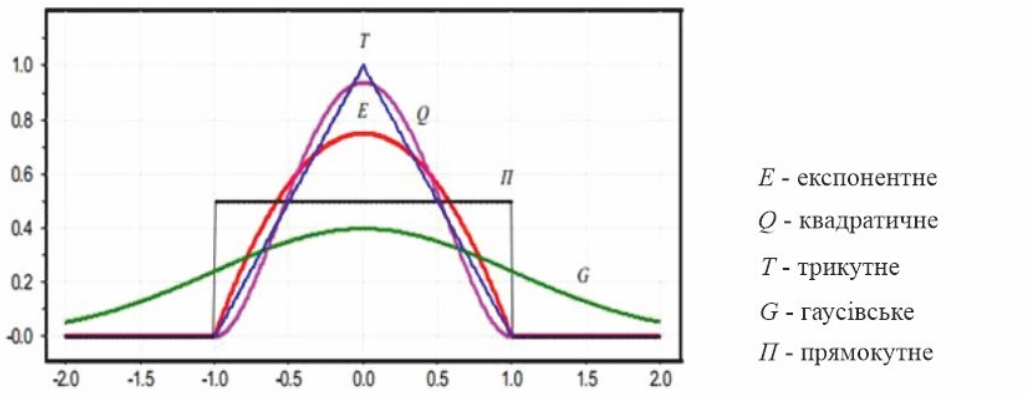


Рис. 7.9

Таблиця 7.2

Ядро $K(r)$	**Формула**
експонентне	$E(r) = (3/4)\ (1 - r^2) \cdot [\lvert r \rvert \le 1]$
квадратичне	$Q(r) = (15/16)\ (1 - r^2)^2 \cdot [\lvert r \rvert \le 1]$
трикутне	$T(r) = (1 - \lvert r \rvert) \cdot [\lvert r \rvert \le 1]$
гаусівське	$G(r) = \frac{1}{\sqrt{2\pi}} exp\left(-\frac{r^2}{2}\right)$
прямокутне	$П(r) = (1/2) \cdot [\lvert r \rvert \le 1]$

Тут $[\lvert r \rvert \le 1]$ – індикатор того, що $r \le 1$ тобто повертається 1, якщо це так і 0 – в іншому випадку.

В результаті ми отримуємо наступний алгоритм класифікації за допомогою парзенівського вікна:

$$a(x^-, X^l, h, K) = \sum_{i=1}^{l} [y_i = y] \cdot K(\frac{\rho(x,x_i)}{h}).$$

Тут ми можемо не сортувати спостереження за відстанями, т.к. функція відстані використовується у самому ядрі.

Фактично алгоритм працює за схожим принципом, що і метод k найближчих сусідів, тільки розглядає всіх сусідів на відстані h і з урахуванням відстані до них.

Наприклад, ось так виглядає карта класифікації для квадратичного парзенівського вікна з параметром $h = 0{,}4$ у просторі двох ознак для кольорів ірисів:

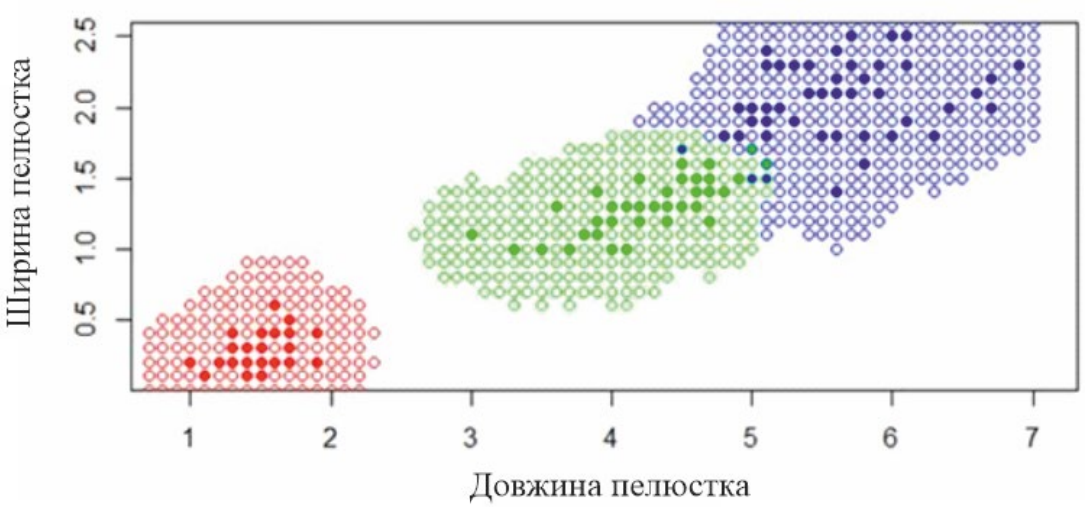


Рис. 7.10

У порівнянні та сама карта класифікації для методу k найближчих сусідів при $k = 6$, виглядає наступним чином:

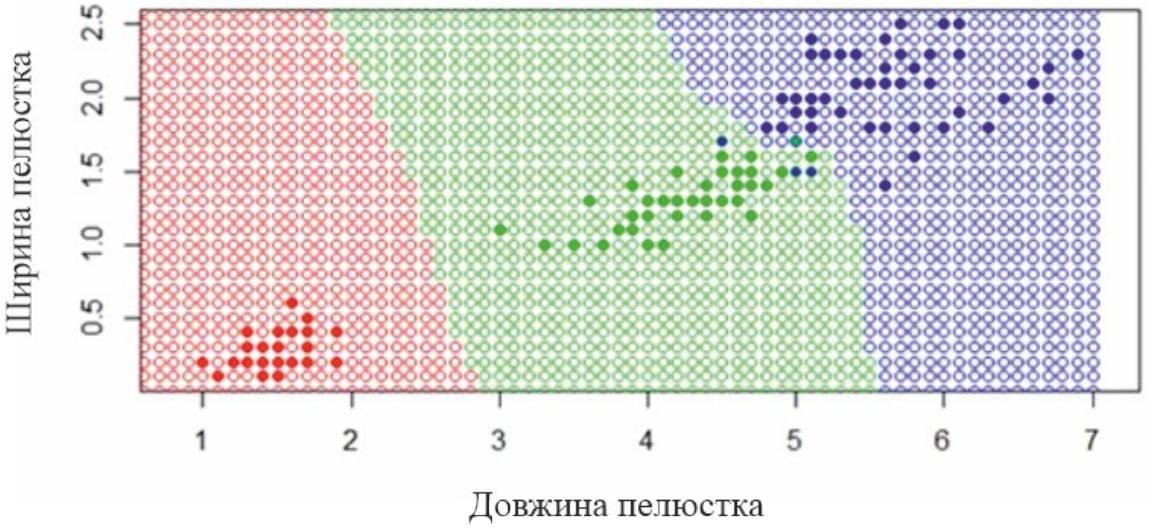


Рис. 7.11

Бачите, як помітно відрізняються результати моделей за цих двох підходів. І, напевно, більш логічні результати ми отримуємо саме при використанні парзенівських вікон, оскільки області прийняття рішень стають обмеженими, як і повинно бути в більшості завдань класифікації.

Однак, у розглянутого підходу є одна істотна вада. Нам потрібно якось вибирати параметр h для кожної конкретної задачі. Зробити це можна двома способами. У першому вчинити також, як ми робили в задачі найближчих сусідів, використовуючи метод leave-one-out (LOO):

$$LOO\ (h, X^l) = \sum_{i=1}^{l} [a(h, x_i; X^l \backslash \{x_i\} \neq y_i] \rightarrow min_h.$$

Тобто, по черзі відкидаємо по одному спостереженню з навчальної вибірки і оцінюємо якість класифікації за даними, що залишилися при заданому значенні параметра h. Потім міняємо h і повторюємо цю процедуру.

За отриманими результатами вибираємо h, який призводить до меншої кількості помилок на навчальній вибірці.

Другий варіант – це «схрестити» метод найближчих сусідів з парзенівським вікном. Якщо вважати, що ядро $K(r)$ відмінно від нуля лише на інтервалі [0; 1], то ширину вікна h можна вибрати таку, щоб воно охоплювало k найближчих сусідів щодо вектору, що оцінюється x становить $h = \rho(x, x^{(k+1)})$ та алгоритм класифікації набуває вигляду:

$$a(x^{-}, X^{l}, k, K) = \sum_{i=1}^{l} [y_i = y] \cdot K\left(\frac{\rho(x,x_i)}{\rho(x,x^{(k+1)})}\right).$$

Щоправда, тут нам знову доведеться повернутися до ідеї впорядкування спостережень щодо зростання відстаней, щоб визначити ширину вікна для кожного конкретного вектору x.

Однак цей другий підхід, як правило, дає кращі результати, ніж вікна з фіксованою шириною. Особливо актуально це тим завдань, у яких щільність (кучність) об’єктів різних класів різна. Тоді підібрати адекватно фіксовану ширину не завжди вдається, а адаптивний вибір ширини $h = \rho(x, x^{(k+1)})$ працює набагато краще. Тому на практиці частіше використовують саме такий другий підхід.

А загальним недоліком цих методів є необхідність вибору ядра $K(r)$. Зазвичай, це робиться з якогось «здорового глузду» в кожному конкретному завданні і, як правило, вибором одного з найчастіше використовуваних ядер, наведених у таблиці. Також можна перебрати кілька варіантів ядер та за допомогою методу *LOO* відібрати найкраще.

Метод потенційних функцій

Дещо інший погляд на алгоритми метричної класифікації дає метод потенційних функцій. Тут міркування йдуть не щодо вектору, що класифікується x, а щодо об’єктів $\{x_i\}$ навчальної вибірки:

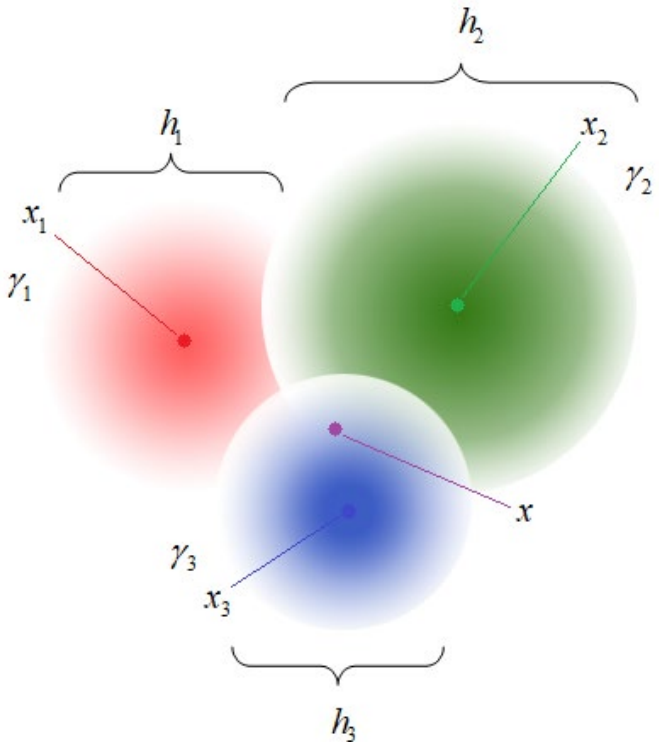


Рис. 7.11

Кожен об’єкт вибірки $\{x_i\}$ – це точка в n-мірному просторі ознак, які ніби «випускають» поля (потенціали) розмірами $\{h_i\}$ та інтенсивностями $\{\gamma_i\}$. Все, що нам залишається, це в точці x визначити, який потенціал найбільший i, відповідно, до того класу i віднести вектор x.

Математично цей алгоритм можна записати, таким чином:

$$a(x, X^l) = \sum_{i=1}^{l} [y_i = y] \cdot \gamma_i \cdot K\left(\frac{\rho(x,x_i)}{h_i}\right),$$

де $\gamma_i \geq 0$ – ваги i-х об’єктів (інтенсивності); $h_i > 0$ – міра ширини розподілу значень ядра $K(r)$ (ядро може бути необмеженим по r).

Недоліком і в той же час перевагою такого підходу є велика кількість параметрів, що настроюються $\{h_i\}$, $\{\gamma_i\}$. У найпростішому випадку їх вважатимуться рівними і вибрати деякі значення h і γ. Або, якщо у нас є деякі знання про властивості об'єктів вибірки, то завжди можна тонко налаштувати ці параметри для кожної групи об'єктів.

Метод потенційних функцій при бінарній класифікації

Цікаву інтерпретацію методу потенційних функцій можна зробити для бінарної класифікації, коли у нас лише два класи з мітками:

$$y \in Y = \{-1, +1\}.$$

У цьому випадку, у нас у точці x найбільшим може виявитися сумарний потенціал полів або 1-го чи 2-го класу:

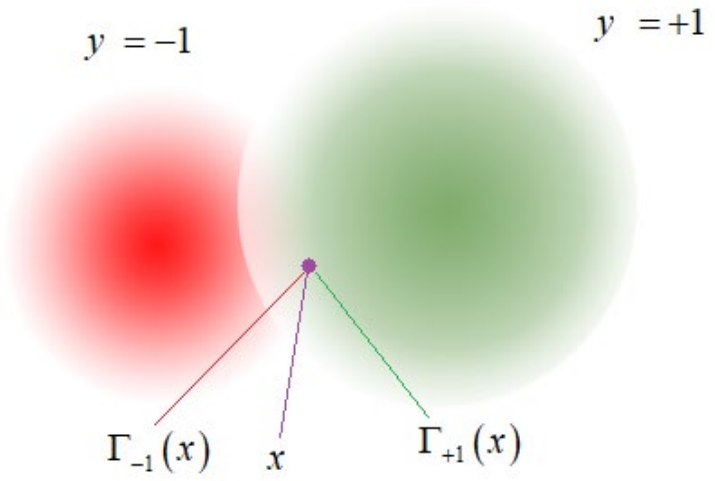


Рис. 8.11

Тут $\Gamma_{-1}(x)$ – сумарний потенціал від об'єктів класу y= -1, а $\Gamma_{+1}(x)$ – сумарний потенціал від об'єктів класу y= +1.

Тоді алгоритм ухвалення рішення можна записати через знакову функцію:

$$a(x^-, X^l) = sign\,(\Gamma_{+1}(x) - \Gamma_{-1}(x)),$$

або так:

$$a(x^-, X^l) = sign\left(\sum_{i=1}^{l} \gamma_i \cdot y_i \cdot K\left(\frac{\rho(x,x_i)}{h_i}\right)\right).$$

В останній формулі значення різниці сумарних потенціалів досягається за рахунок міток класів {+1; -1}. Для всіх об'єктів класу +1 отримуємо звичайну суму, а об'єктів класу – 1 – суму зі знаком мінус. У результаті маємо сумарну різницю потенціалів об'єктів обох класів.

Якщо тепер виділити вираз:

$$f_j(x) = y_j \cdot K\left(\frac{\rho(x,x_i)}{h_i}\right), j=1, 2, \ldots, l$$

то його можна сприймати як формування нової ознаки для j-го об'єкта навчальної вибірки.

Причому, ця ознака формується, як функція від відстані між вектором, що класифікується. x та j-м об'єктом x_j. Далі автоматично отримуємо наступний алгоритм класифікації:

$$a\,(x) = sign\left(\sum_{j=1}^{l} \gamma_j \cdot f_j(x)\right).$$

Що нагадує ця формула? Так, це загальний запис лінійного алгоритму бінарної класифікації образів. А $\{\gamma_j\}$ – набір вагових коефіцієнтів, що визначають орієнтацію гіперплощини у просторі ознак. У разі число ознак виходить рівним розміру всієї навчальної вибірки. Але ми можемо їх скоротити, якщо, наприклад, підбирати коефіцієнти $\{\gamma_j\}$ градієнтним (або псевдограліентним) алгоритмом з $L1$-регуляризацією. Тоді незначні об'єкти

навчальної вибірки просто зануляться і залишиться кілька ненульових $\{\gamma_j\}$, які й визначатимуть розмірність простору ознак.

Цей приклад показує, що ми можемо створювати нові ознаки для об'єктів $\{x_i\}$ навчальної вибірки, використовуючи відстані до них. Для низки завдань, де ми можемо використовувати метричний простір, це дуже корисний прийом розширення простору ознак.

Отже, на цьому занятті ми з вами розглянули два очевидні узагальнення найпростішого метричного алгоритму *k* найближчих сусідів:

- метод парзенівського вікна;
- метод потенційних функцій.

Переваги всіх метричних алгоритмів у простоті реалізації. Правда, при цьому потрібно зберігати всю вибірку розмічених даних (навчальну вибірку), за якими здійснюється подальша класифікація

7.4. Метричні регресійні методи. Формула Надарая-Ватсона

На попередніх заняттях ми побачили, як можна використовувати метричний підхід для побудови класифікаторів. Але його можна застосовувати і в завданнях регресії, коли виходом моделі є не номер класу, а деяке речове значення:

$$a\,(x) = y \text{ O } \mathbb{R} \text{ або } \mathbb{R}^n$$

Давайте припустимо, що ми маємо деякий набір розмічених даних $\{x_i, y_i\}_{i=1}^{l}$, пов’язаних деякою залежністю (не обов’язково строго функціональною):

$$y_i = f\,(x_i),\ i = 1, 2,\ldots,l$$

У таких завданнях модель $a(x)$ має вловити природу взаємозв'язку між входами x_i та виходами y_i. Раніше ми з вами вирішували це завдання шляхом параметричної оптимізації. Тобто передбачали залежності у вигляді деякої функції:

$$a\,(x,\ \theta) = g(x,\ \theta)$$

та знаходили вектор параметрів θ так, щоб забезпечити мінімум функціоналу якості.

Якщо функціонал має вигляд:

$$Q(a, X^l) = \sum_{i=1}^{l} w_i \cdot (y_i - a(x_i, \theta))^2 \to min_\theta,$$

тоді автоматично приходимо методу найменших квадратів (*МНК*).

Тут $\{w_i\}$ – ваги доданків (ступінь їх важливості). У найпростішому випадку їх можна прирівняти одиниці.

Істотним недоліком такого підходу є необхідність вибору параметричної функції $g(x,\ \theta)$. Вона має коректно відображати залежність $y = f(x)$ даних. І ця залежність не завжди очевидна. У цьому випадку на вирішення цього завдання можна подивитися з іншого боку.

Якщо нам не важко задати метрику $\rho(x, x_i)$, $i = 1, 2, \ldots, l$ між довільним відліком x та значеннями $\{x_i\}$, то можна застосувати метричні методи апроксимації вихідних значень $\{y_i\}$.

Давайте, я покажу принцип такого підходу на простому прикладі лінійної апроксимації даних із двох точок:

$$\{(x_1, y_1),(x_2, y_2)\}$$

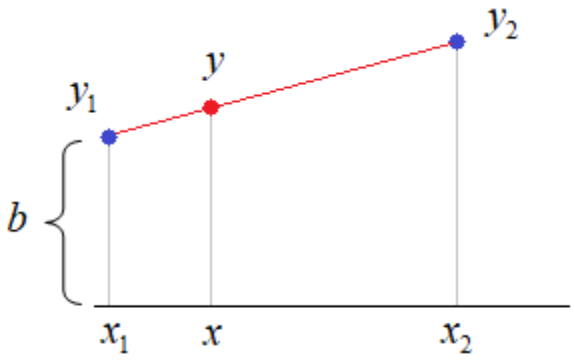


Рис. 8.12

Тут усі проміжні значення можна обчислити за формулою прямої:

$$y = kx + b, \quad x \text{ O } [x_1; x_2].$$

Для простоти припустимо, що:

$$x_1 = 0; x_2 = 1$$

Тоді:

$$k = y_2 - y_1; b = y_1$$

Цю ж формулу можна переписати у вигляді:

$$y = y_2 \cdot x - y_1 \cdot x + y_1;$$
$$y = y_1 \cdot (1 - x) + y_2 \cdot x;$$

Дивіться тут

$$\{1 - x = |x_2 - x| \, x = |x_1 - x|$$

відстані до сусідніх точок $x_1; x_2$ тобто фактично метрика:

$$\rho(x, x_i) = | x - x_i|, \quad i = 1, 2$$

У цих позначеннях величину y можна визначити, таким чином:

$$y = y_1 \cdot \rho(x, x_2) + y_2 \cdot \rho(x, x_1);$$
$$y = y_1 \cdot (1 - \rho(x, x_1)) + y_2 \cdot (1 - \rho(x, x_1))$$

А узагальнення на довільний інтервал $h = x_2 - x_1$ дасть вираз

$$y = y_1 \cdot \left(1 - \frac{\rho(x,x_1)}{h}\right) + y_2 \cdot \left(1 - \frac{\rho(x,x_2)}{h}\right).$$

Вам це нічого не нагадує? Дивіться, тут ваги перед y_1, y_2 можна виразити через трикутне ядро:

$$K(r) = |1 - r\,| \cdot [r \le 1]$$

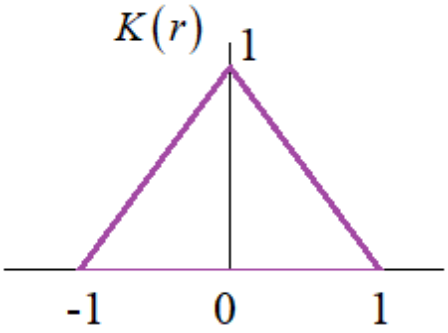


Рис. 8.13

Отримаємо:

$$y = y_1 \cdot K\left(\frac{\rho(x,x_1)}{h}\right) + y_2 \cdot K\left(\frac{\rho(x,x_2)}{h}\right).$$

Або, узагальнюючи на довільну кількість відліків, маємо:

$$y = a(x, X^l) = \frac{\sum_{i=1}^{l} y_i \cdot K\left(\frac{\rho(x,x_i)}{h}\right)}{\sum_{i=1}^{l} K\left(\frac{\rho(x,x_i)}{h}\right)} = \frac{\sum_{i=1}^{l} y_i \cdot w_i(x)}{\sum_{i=1}^{l} w_i(x)}.$$

В останній рівності введено позначення вагових коефіцієнтів:

$$w_i(x) = K\left(\frac{\rho(x,x_i)}{h}\right).$$

Також зверніть увагу, ми тут у формулу додали нормування за вагами, тому що при більшій кількості відліків і різних функцій ядра, нам необхідно зберегти масштаб вихідних значень.

Отримана формула відома під назвою формули **ядерного згладжування Надарая-Ватсона**.

Її можна вивести суворішими математичними методами, наприклад, з критерію мінімуму квадрата помилки, при константній параметричній функції:

$$a(x, X^l) = \sum_{i=1}^{l} w_i(x) \cdot (\alpha - y_i)^2 \to \min_{\alpha}$$

Диференціюючи цей вираз по α і прирівнюючи результат нулю, отримаємо:

$$\frac{\partial Q(\alpha; X^l)}{\partial \alpha} = 2 \cdot \sum_{i=1}^{l} w_i(x) \cdot (\alpha - y_i) = 0$$

Звідки

$$\alpha = y = a(x) = \frac{\sum_{i=1}^{l} y_i \cdot w_i(x)}{\sum_{i=1}^{l} w_i(x)}$$

Цей простий висновок показує, який критерій якості мінімізується формулою Надар-Ватсона.

Загальний висновок тут такий. Ми отримуємо можливість апроксимувати експериментальні залежності $\{(x_i, y_i)\}$ через визначення метрики $\rho(x, x_i)$ та функції ядра $K(r)$. В результаті ми пішли від явного визначення параметричної функції $g(x, \theta)$ для всього обсягу вибірки, а вказуємо лише спосіб апроксимації локальних областей.

Як функція ядра $K(r)$ можна використовувати будь-які парзенівські вікна, а метрика вибирається виходячи з знань про експериментальних даних.

Це може бути і модуль, як у розглянутій задачі, і евклідова відстань та багато інших відомих метрик.

Експериментальні дані тут генеруються за такою формулою:

$$y_i = \sin(x_i) + \varepsilon_i, \quad i = 1, 2, \ldots,$$

де ε_i –гаусівські випадкові величини з нульовим середнім та дисперсією 0,5.

Самі ж величини x_i О [0; 10] із кроком 0,1.

Потім ми визначимо метрику як модуль відстані:

$\rho(x, x_i) = | x - x_i |, i = 1, 2, \ldots$

А як ядро візьмемо гаусівську криву: $K(r) = \exp(-2r^2)$

Отримаємо наступні результати за різної ширини вікна h:

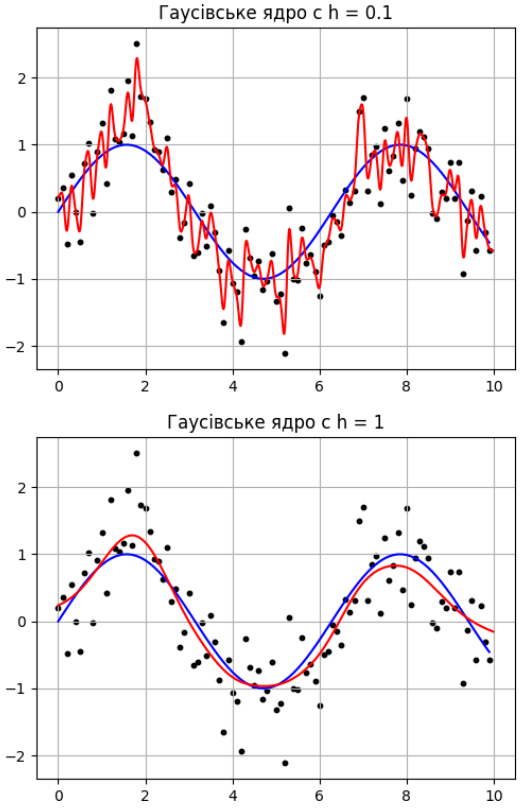


Рис. 7.14

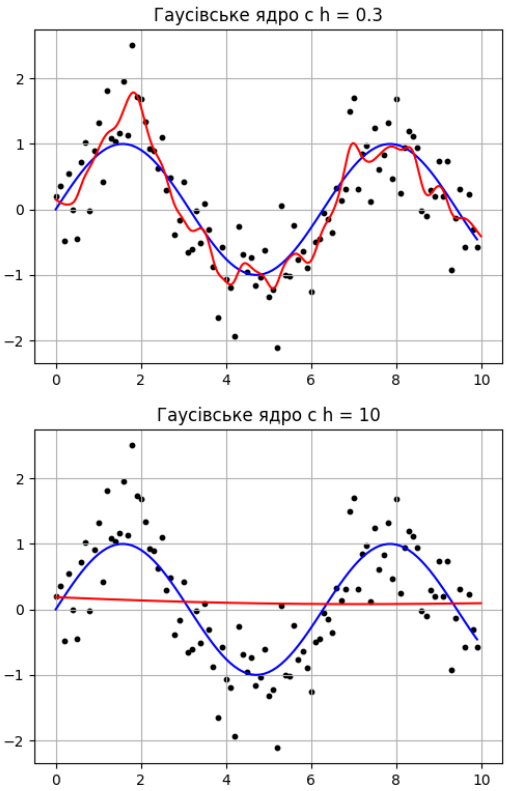


Рис. 7.15

Як бачимо, найкращі результати виходять при ширині вікна $h = 1$, причому зі зростанням h у нас дані, фактично, усереднюються.

Давайте тепер виберемо прямокутне вікно і подивимося, як виглядатиме апроксимація даних:

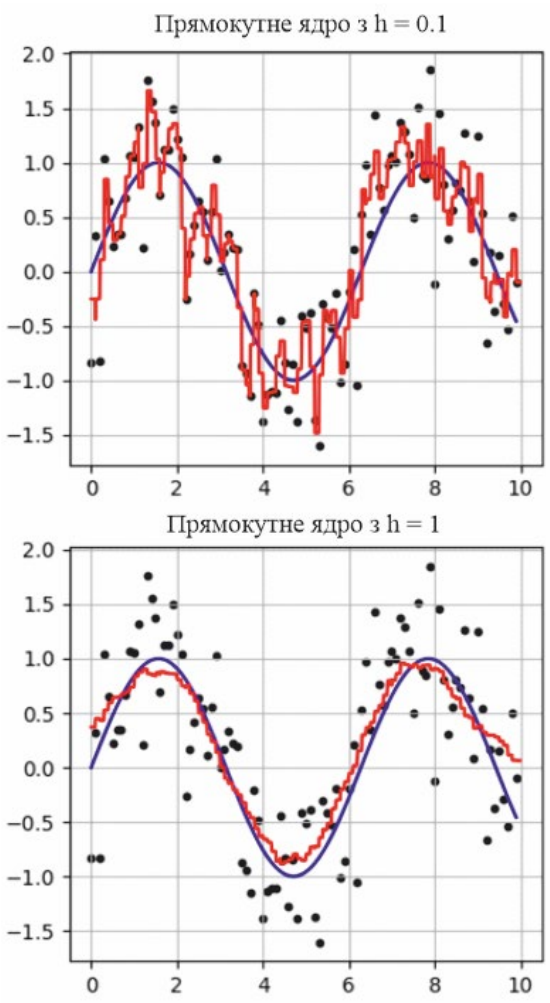


Рис. 7.16

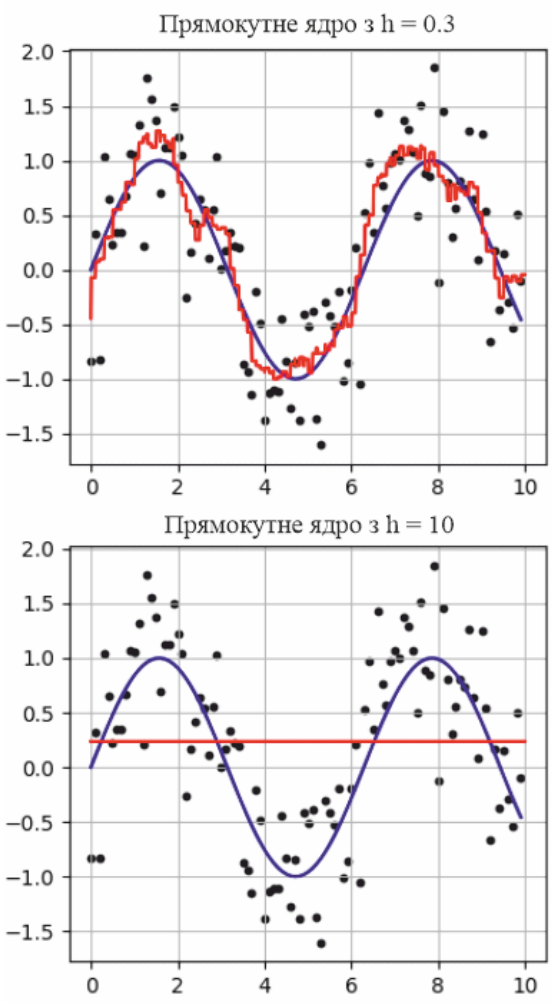


Рис. 7.17

Тут червона крива виглядає не такою гладкою, т.к. використовується не гладка функція ядра. В усьому іншому принцип зберігається: найкраще значення при $h = 1$, а при $h = 10$ маємо середнє арифметичне за всіма вихідними (цільовими) значеннями вибірки.

В обох випадках мале значення h призводить до свого роду перенавчання моделі, т.к. ми надто сильно підлаштовуємося під вихідні значення. Така модель навряд чи даватиме хороші прогнози. З іншого боку, велике значення h дуже сильно огрубляє вихідні значення і призводить до значного розходження між істинною залежністю (синій графік) та відновленими значеннями (червоний графік).

На практиці ширину вікна h часто вибирають за результатами експериментів, наприклад методом ковзного контролю leave-one-out (*LOO*):

$$LOO\,(h, X^l) = \sum_{i=1}^{l} (a_h(x_i; X^l \backslash \{x_i\}) - y_i)^2 \to min_h.$$

Тут ми послідовно з вибірки прибираємо по одному спостереженню і будуємо його прогноз щодо всіх інших. Те вікно h, яке призводить до кращих прогнозів, вибираємо як параметр результуючої моделі.

Ось, загалом, принцип вирішення регресійних завдань метричними методами, що призводять до формули ядерного згладжування Надарая-Ватсона.

7.5 Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Дайте визначення стратегії one-vs-all (один проти всіх).
2. Дайте визначення стратегії all-vs-all (все проти всіх).
3. Яким методом слід використовувати багатокласову логістичну регресію?
4. Дайте визначення метриці accuracy.
5. Дайте визначення метричним методам.
6. Дайте визначення метриці Мінковського.
7. Дайте визначення методу *k* найближчих сусідів.
8. Надайте переваги і недоліки методів найближчих сусідів.

7.6 Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.
2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.
4. Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale FPGA-based convolutional networks. In R. Bekkerman, M. Bilenko, and J. Langford, editors, Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press.

Лекція №8. Вирішальні дерева

Результати навчання, які формуються:

ДРН6: Застосовувати знання машинного навчання в умовах слабо структурованих даних різної природи.

ДРН7: Використовувати алгоритми побудови асоціативних та класифікуючих правил.

8.1. Логічні методи класифікації

На цьому занятті ми торкнемося нового класу алгоритмів у машинному навчанні під назвою логічні методи класифікації.

Досить часто практично зустрічаються завдання, де потрібно робити висновок по невеликому набору вихідних даних (ознак), вибудовуючи деякий ланцюжок міркувань. Наприклад, наступна схема дозволяє зробити деякі висновки залежно від кількох відповідей на прості запитання:

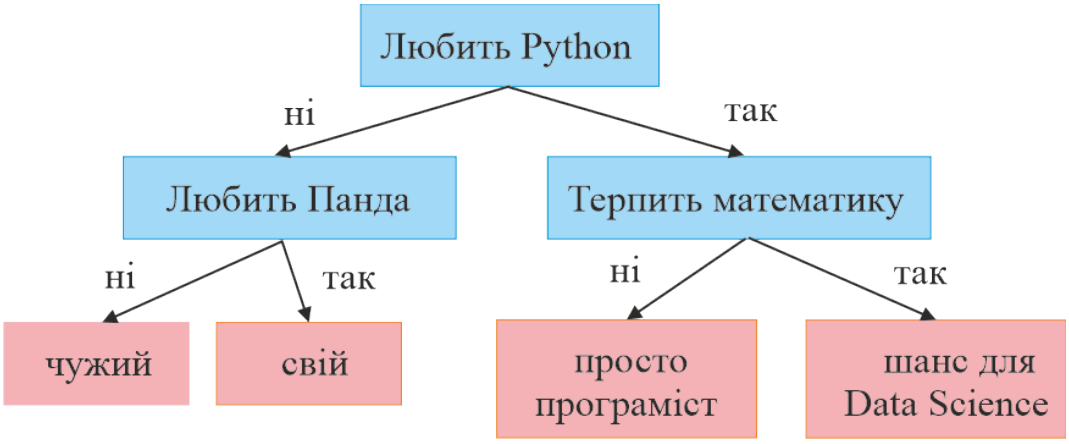


Рис. 8.1

Тут умовно можна виділити три бінарні ознаки:

$f_1(x) = \{0 - \textit{не любить "Python"}\ 1 - \textit{любить "Python"}$

$f_2(x) = \{0 - \textit{не любить "Панда"}\ 1 - \textit{любить "Панда"}$

$f_3(x) = \{0 - \textit{не терпить математику}\ 1 - \textit{терпить математику}$

І сформувати на їх основі логічні правила виведення, наприклад:

$$R(x) = f_1(x)\ \text{Щ}\ \underline{f_3(x)} \rightarrow \texttt{просто програміст}$$

$$R(x) = \underline{f_1(x)}\ \text{Щ}\ f_2(x) \rightarrow \texttt{свій}$$

Тобто, у кожному правилі ми маємо послідовність тверджень, які всі мають бути виконані, щоб зробити той чи інший висновок. Це найпростіші приклади логічних висновків з урахуванням серії кон'юнкцій бінарних ознак. Причому в логічних правилах використовується досить обмежена кількість ознак для виведення (рекомендується 2-4 і трохи більше 7).

Це з людським сприйняттям такого висновку, коли експерт намагається зрозуміти, чому алгоритм рекомендує те чи інше рішення. Усвідомити дві-три ознаки (предикатів) щодо просто. Наприклад, система відхиляє кредит позичальнику, зазначаючи, що він низький прибуток і великий вік. А якщо цих показників буде більше, наприклад, наявність квартири, дітей, машини, закордонного паспорта тощо, то експерту вже значно складніше оцінити причини відмови або, навпаки, рекомендації видати кредит. І так у різних областях. Тому рекомендаційні системи та загалом логічні методи, як правило, оперують невеликою кількістю ознак.

Добре, я думаю, цей момент зрозумілий. Давайте тепер дещо розширимо, узагальним правила логічних висновків, покладемо ознаки речовими, числовими: $f_j(x)$ O $\mathbb{R}$, $j = 1, 2, \ldots, n$.

Тут n – загальна кількість ознак, обчислених за вектором x.

Як у такому разі робити логічні висновки? Очевидно, тут кожну ознаку потрібно порівнювати з деяким числовим порогом та видавати відповіді 0 або 1 (аналог «так» і «ні»).

Математично це зручно записати з використанням нотації Айзерсона:

$$[f_j(x) \le a_j], \quad j = 1, 2, \ldots, n$$

або в ще більш загальній формі:

$$[a_j \le f_j(x) \le b_j], \quad j = 1, 2, \ldots, n.$$

Якщо умова виконується, то дужка повертає 1, інакше – 0. У результаті ми перетворюємо речову ознаку знову на бінарну і можемо сформувати логічні правила за раніше наведеним принципом:

$R(x) = [a_j \le f_j(x) \le b_j]$

Тут J – це безліч ознак, які ми відбираємо на формування висновку.

Щоб ця математика була більш зрозумілою, наведу приклад із класифікацією кольорів ірису Фішера. На малюнку трьома різними кольорами показано розподіл трьох видів квітів ірисів за двома ознаками: ширина та довжина пелюсток. Наша мета – розділити їх за класами, використовуючи правила логічного висновку на основі цих двох речових ознак.

Очевидно, перший клас C_1 (червоні точки) дуже добре відокремлюється від інших за другою ознакою з відсіканням по порозі a_2:

$R(x) = [f_2(x) \le a_2] \rightarrow \{1 - \text{клас } C_1 \quad 0 - \text{другі класи}$

Для класифікації двох інших додатково потрібно порівнювати першу ознаку з порогом a_1:

$R(x) = \underline{[f_2(x) \le a_2]} \wedge [f_1(x) \le a_1] \rightarrow \{1 - \text{клас } C_2 \; 0 - \text{другі класи}$

Тут верхня риса означає заперечення предикату (якщо було 0 стає 1, якщо була 1, стає 0). Або цей же логічний висновок можна записати без заперечення, змінивши тільки знак у дужці на більше:

$R(x) = \underline{[f_2(x) > a_2]}$ Щ $[f_1(x) \le a_1] \rightarrow \{1 - \text{клас } C_2 \quad 0 - \text{другі класи}$

Очевидно, отримаємо те саме правило.

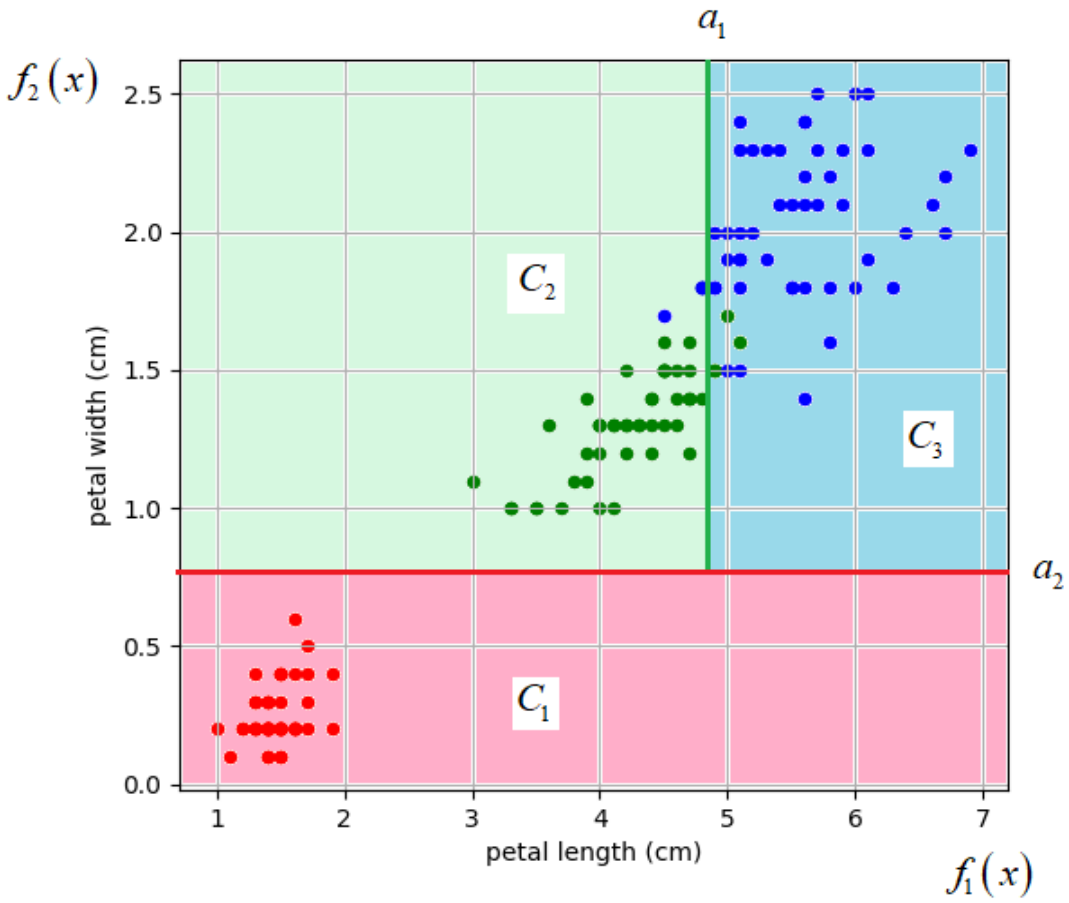


Рис. 8.2

Зрештою, останній третій клас можна виділити за правилом:

$R(x) = [f_2(x) \le a_2]$ Щ $[f_1(x) \le a_1] \rightarrow \{1 - клас\ C_3\ \ 0 - другі\ класи$

Ось приклад із речовими ознаками та набором правил логічного висновку для визначення видів кольорів ірису.

Всі ці правила можна об'єднати та побудувати наступне вирішальне дерево:

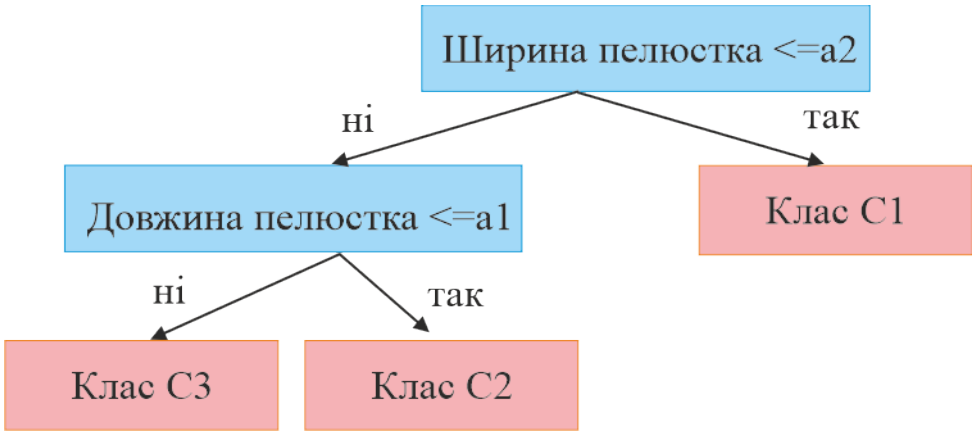


Рис. 8.3

Ми не випадково назвали цей спрямований ациклічний граф вирішальним деревом. Це один із найпоширеніших підходів серед логічних

методів класифікації. Тобто якщо на основі вихідних розмічених даних (навчальної вибірки) вдається побудувати таку деревоподібну структуру, то у нас автоматично формується набір правил для класифікації даних навчальної вибірки. А якщо пощастить, то і для довільних даних тієї ж природи, що і вибірка, що навчає.

Але перш, ніж з головою поринути у теорію вирішальних дерев, слід зазначити, що логічні методи не обмежуються лише ними. Є ще кілька загальних підходів на формування правил логічного висновку.

З першим загальним підходом ми з вами щойно познайомилися – це набір предикатів, об'єднаних кон'юнкціями: $R(x) = [a_j \le f_j(x) \le b_j]$ і, як наслідок, одержуємо з них вирішальні дерева.

Інший підхід під назвою синдром був підглянутий у лікарській практиці (звідси та його назва). Коли лікар ставить діагноз, то спочатку вивчає набір симптомів у пацієнта, а потім робить висновок на користь захворювання, якому більшою мірою відповідають симптоми, що спостерігаються.

Математично це можна формалізувати так:

$$R(x) = [\sum_{j \in J} [a_j \le f_j(x) \le b_j] \ge d]$$

Тобто тут ми маємо певний набір ознак (вимірювань) J. Наприклад, температура тіла, почервоніння, першіння в горлі тощо. А потім підраховуємо, скільки з цих ознак не відповідають нормі, тобто, виділяємо набір симптомів. Причому у різних пацієнтів набір цих симптомів може бути різним. Але якщо набирається певна кількість d симптомів характерних для тієї чи іншої хвороби, робиться висновок (ставляться діагноз), що у хворого саме це захворювання.

Зрештою, є ще пара відомих, але менш поширених схем побудови логічних висновків:

- напівплощина (лінійна гранична функція):

$$R(x) = [\sum_{j \in J} w_j f_j(x) \ge w_0]$$

- куля (порогова функція близькості):

$$R(x) = [\rho(x, x_0) \le w_0]$$

Тут $\rho(x, x_0)$ – деяка метрика (відстань між векторами). В останньому випадку ми дивимося, на скільки близький вектор x до певного характерного представника x_0 певного класу. Якщо x потрапляє в околицю x_0 щодо сукупності певних ознак, то ми відносимо його до того ж класу, що й x_0.

Подібних правил дуже багато, але інші дуже рідко застосовуються практично. Здебільшого найпростіші, що ми відзначили на початку нашого заняття.

У вас тут може виникнути питання, а які правила коли використовувати? Насправді це часто випливає з постановки завдання (технічного завдання – ТЗ). Деякі експерти добре сприймають правила як набору кон'юнкцій. Інші (зокрема медики) – правила на основі синдромів. Десь явно слід виділяти характерні об'єкти та вимірювати близькість до них. І так далі. Тобто, все це,

як правило, випливає з вирішуваного завдання. Але найчастіше, все ж таки, використовуються правила у вигляді набору кон'юнкцій, які добре виражаються у вигляді вирішальних дерев. Саме про них ми докладніше поговоримо на наступному параграфі.

8.2. Критерії якості для побудови вирішальних дерев

В попередньому параграфі ми з вами познайомилися із загальною ідеєю логічних методів. Тут же продовжимо цю тему та докладніше поговоримо про бінарні вирішальні дерева див. рис. 8.1.

Фактично це зв'язковий ациклічний граф, у якому є корінь, внутрішні вершини та листи (вершини без нащадків). Причому кожна внутрішня вершина має рівно два відгалуження, що відповідають стрілкам «так» і «ні». Саме тому дерево з такою структурою називають бінарним.

Якщо у нас є вже побудоване дерево за багатьма даними, то логічний висновок робиться дуже просто.

Припустимо, ми класифікуємо вектор: $x = [x_1, x_2, \ldots, x_n]^T$ із n ознаками.

Тоді, починаючи з кореневої вершини v_0, перевіряємо предикат: $B_v(x) = [f_j(x) \le a_j]$. Тобто ми дивимося на j-у ознаку для вектору x і порівнюємо його з порогом a_j. Якщо умова виконується, йдемо за стрілкою з позначкою 1, інакше – за стрілкою з позначкою 0. У наступній проміжній вершині повторюємо цей процес і за допомогою предикату $B_v(x)$ перевіряємо значення іншої j-ї ознаки (або тієї самої). Залежно від відповіді, переходимо або за стрілкою 1, або за стрілкою 0.

Як тільки дійшли до аркуша, відносимо вектор x до того класу, що прописаний у поточному аркуші. Процес логічного висновку у цьому завершується.

Саме тому листові вершини ще називають термінальними.

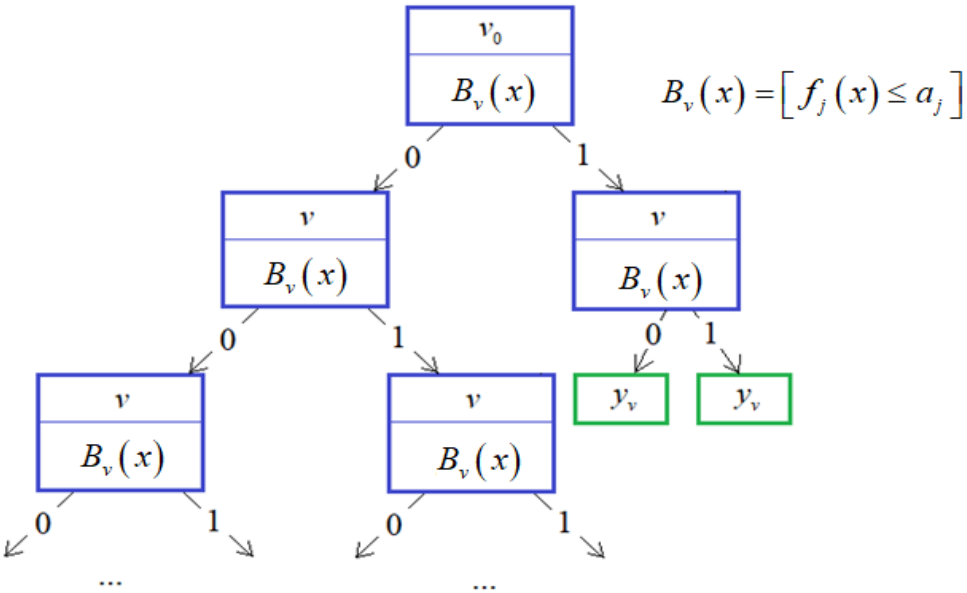


Рис. 8.6

Однак, щоб зробити такий висновок, ми з вами повинні знати структуру дерева, набір предикатів B_v для кожної проміжної вершини та мати правильні значення у листових вершинах. Листи дерева, як правило, зберігають певні числа. Вони можуть означати чи номер (мітку) класу, чи константне значення функції під час вирішення завдань регресії. Але поки що будемо вважати, що в них записані позначки класу.

Отже, головне питання, як побудувати бінарне вирішальне дерево за багатьма даними навчальної вибірки $X^l = \left\{(x_i, y_i)_{i=1}^{l}\right\}$?

Щоб краще розуміти цей процес, давайте візьмемо дуже простий приклад розбиття послідовності червоних та синіх куль, власне, на червоні та сині:

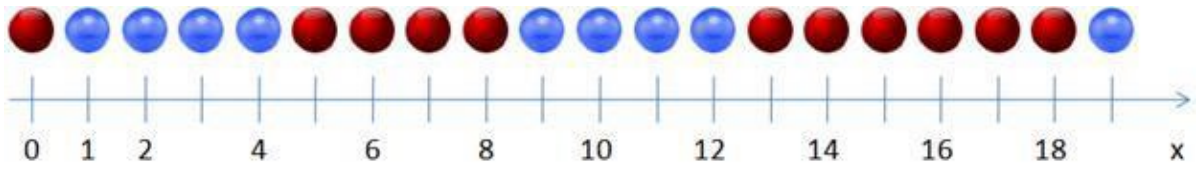


Рис. 8.7

Тут вектор $x = [x_1]$ складається з однієї компоненти і є числом – номер позиції кулі. Це єдиний ознака, яким можна виконувати розбиття даної послідовності. У кожній проміжній вершині бінарного вирішального дерева буде використано предикат:

$$B_{j,t}(x) = [x_j \leq t].$$

І, оскільки ознака всього одна, то $j = 1$. Залишається знайти лише потрібні пороги t.

Якби розбиття робив ми самі сам, то запропонував би таке вирішальне дерево:

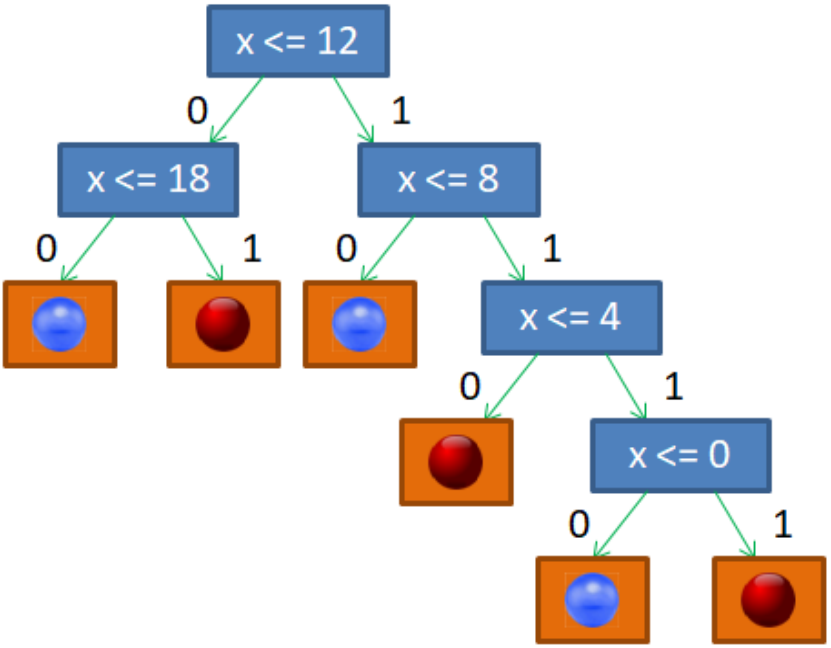


Рис. 8.8

Чому так? Насправді можна було б придумати й інші варіанти. Я керувався простим принципом: у кожній вершині поріг *t* слід вибирати так, щоб в одній частині було якнайбільше представників тільки одного будь-якого класу (куль одного кольору), а в іншій – всі, що залишилися. Тобто це певний критерій здорового глузду, природна евристика, яка скорочує глибину дерева. Але ви можете сказати, навіщо нам мінімізувати глибину вирішального дерева? Нехай воно виходить таким, яким зробить його алгоритм.

Наприклад, почергово перебирати пороги *t* від 0 до 19 і рано чи пізно гарантовано розіб'ємо послідовність куль на червоні та сині. Так, дерево буде значно глибше, та й що? Однак, тут завжди слід пам'ятати, що дана послідовність лише навчальна вибірка. Ми маємо на увазі використовувати це дерево і для інших подібних послідовностей з дещо іншим розподілом синіх та червоних куль. Інакше це було б не завдання машинного навчання, а просто розбиття конкретної вибірки. Так от, можна помітити, що чим менше глибина вирішального дерева, тим менше використовується порогів розбиття і тим вище, в середньому узагальнююча здатність такого дерева. Саме тому намагаються будувати дерева мінімальної глибини.

Але як це зробити? У найпростішому варіанті можна виконати повний перебір всіх можливих розбиття та вибрати дерево мінімальної глибини. Для нашого завдання це якось можна було б реалізувати. Але, як ви розумієте, загалом це займе так багато часу, що не вистачить і життя всього людства. Тому тут треба шукати інших підходів.

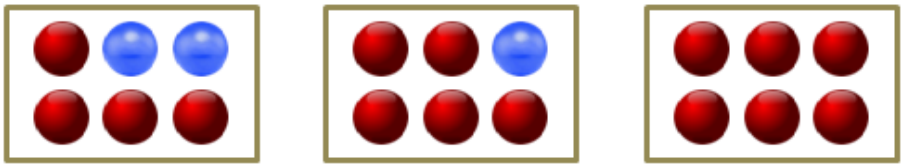
Якщо повернутися до мого варіанту бінарного дерева, то, як я зазначав, у кожній проміжній вершині намагався поріг *t* вибирати так, щоб в одній частині було якнайбільше представників одного класу (кулі одного кольору), а в іншій як вийде, тобто. всі, що залишилися. Це якась евристика. Наша мета,

наділити алгоритм приблизно такою самою евристикою. Як це зробити? Математики звернули свої погляди в ранні напрацювання різних формул і представили світу кілька критеріїв якості предикатів.

Одним із популярних стала ентропія $S = -\sum_{i=1}^{N} p_i p_i$, де p_i – ймовірність i-го класу; N – загальна кількість класів (у прикладі – кольорів куль).

Відомо, що ця величина є певною характеристикою хаотичності системи. Чим більше різноманітності, тим вища ентропія, і навпаки, що більше порядку, то вона менша. Її можна використовувати для оцінки характеристики поточного розбиття кожної вершини.

Наприклад:



$S \approx 0,92$ $S \approx 0,65$ $S = 0$

Рис. 8.9

Таку характеристику у вирішальних деревах називають impurity (інформативністю). Чим вона менша, тим краще (якісніше) набір даних у вершині. І, навпаки, чим вона більша, тим різноманітніші (хаотичніші) дані у вершині.

Повернемося, тепер, наприклад, розбиття послідовності червоних і синіх куль. Послідовність складається з $N = 2$ класів із ймовірностями появи кожного з них:

$$p_1 = \frac{11}{20},\ p_2 = \frac{9}{20}.$$

Отже, непридатність (в даному випадку ентропія) кореневої вершини дерева дорівнює:

$$S_0 = -p_1 \cdot \log_2(p_1) - p_2 \cdot \log_2(p_2) \approx 0,993$$

Далі, ми вибираємо предикат із порогом $t = 12$:

$$B_{j=1,t=12}(x) = [x \le 12]$$

і отримуємо дві підвибірки з impurity (ентропіями):

$$S_1 = -\frac{5}{13} \cdot \log_2\left(\frac{5}{13}\right) - \frac{8}{13} \cdot \log_2\left(\frac{8}{13}\right) \approx 0,96.$$

$$S_2 = -\frac{6}{7} \cdot \log_2\left(\frac{6}{7}\right) - \frac{1}{7} \cdot \log_2\left(\frac{1}{7}\right) \approx 0,96.$$

Величина 0,6 значно менше початкового значення 0,993, отже, у другій послідовності більше порядку, ніж у вихідній. І це дуже добре. Саме цього ми й досягаємо, вибираючи поріг t.

Тепер нам потрібно на основі величин S_0, S_1, S_2 сформувати єдиний критерій (одне число), яким ми судили про якість розбиття вибірки на дві підвиборки. Це також можна виконати у різний спосіб, але часто

використовується наступна формула під назвою інформаційний виграш (information gain):

$$IG\,(Q) = S_0 - \sum_{i=1}^{q} \frac{N_i}{N} S_i$$

де q – число груп (підвибірок) після розбиття; N_i – число елементів в i-й групі після розбиття; Q – критерій розбиття (предикат).

Для нашого прикладу оцінки якості розбиття кореневої вершини з порогом $t = 12$ маємо:

$$IG\,(\boldsymbol{x} \leq 12) = S_0 - \frac{13}{20} S_1 - \frac{7}{20} S_2 \approx 0{,}16.$$

Якщо порахувати цей критерій інших порогів t, то виявиться, що з $t = 12$ маємо найбільше значення. Отже, найкраще розбиття з погляду обраних критеріїв: ентропії та інформаційного виграша.

Математично пошук найкращого предикату в поточній вершині дерева можна записати так:

$$t_+ = argIG(t)$$

Або, загалом, враховуючи, що ми підбираємо не лише пороги, а й ознаки j, за якими діляться об'єкти на дві підвиборки, маємо:

$$j_+, t_+ = argIG(j, t)\,.$$

А як impurity, крім ентропії також використовують:

– **критерій Джині** (Gini impurity): $G(j,\ t) = 1 - \sum_k (p_k)^2$, тут p_k - імовірність (частота) появи об'єктів k-го класу у вершині дерева;

– **помилка класифікації** (Misclassification error): $E(j, t) = 1 - p_k$.

(Критерій Джині не слід плутати з індексом Джині, про який ми раніше говорили – це дві різні характеристики.)

На практиці критерій Джині працює майже так само, як і ентропія.

Це добре видно з графіків даних критеріїв за бінарної класифікації:

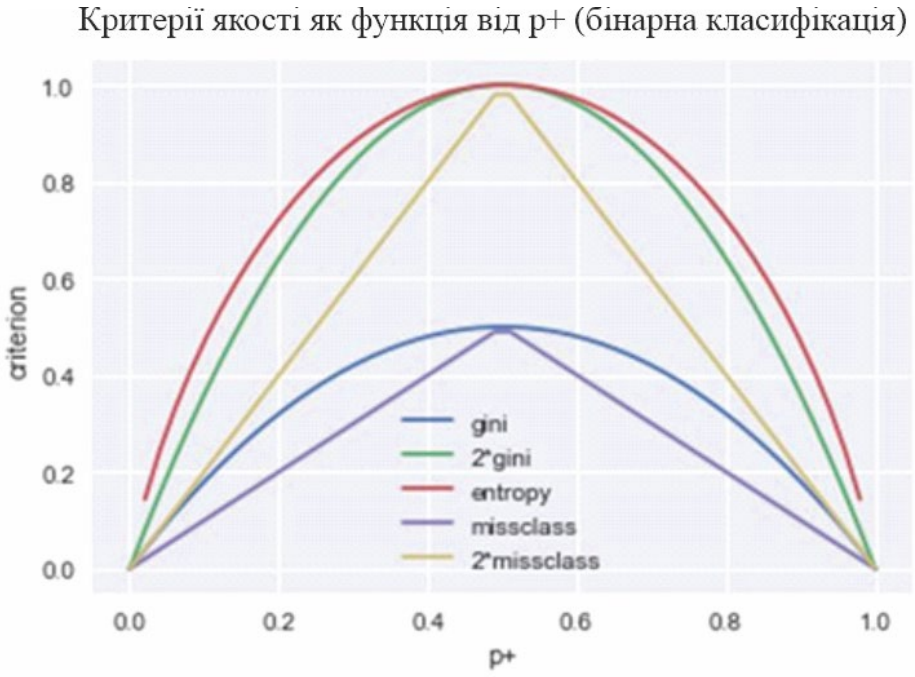


Рис. 8.10

Тут по осі абсцис (горизонтальної осі) відкладено можливість p_+ позитивного класу, а, по осі ординат – значення критеріїв. Як бачите, графіки критерію Джині та ентропії практично збігаються. Тому вони і призводять до подібних конструкцій вирішальних дерев.

На наступному занятті ми розглянемо два алгоритми побудови вирішальних дерев, використовуючи розглянуті критерії якості розбиття

8.3. Побудова вирішальних дерев жадібним алгоритмом ID3

На попередньому занятті ми з вами познайомилися з критеріями вибору предикатів для розбиття безлічі даних на вершинах вирішального дерева.

Насамперед – це impurity (інформативність, міра невизначеності), позначимо її через $H(R_m) \to \min$, де R_m – вибірка, що потрапила в поточну вершину. І показник якості розбиття, що часто обчислюється як інформаційний виграш:

$$Q(R_m, j, t) = H(R_m) - \frac{|R_r|}{|R_m|} H(R_l) - \frac{|R_r|}{|R_m|} H(R_r) \to \max$$

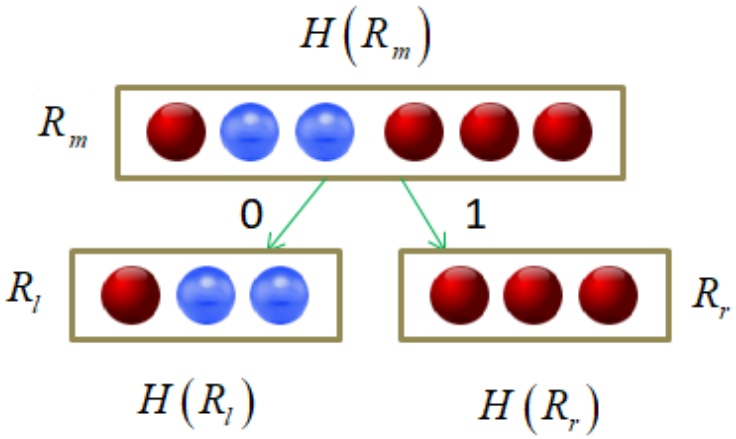


Рис. 8.11

Давайте тепер скористаємося цими показниками для побудови вирішального бінарного дерева.

Як навчальну вибірку візьмемо 150 об’єктів трьох класів ірисів (50 у кожному класі), розподілених у двовимірному просторі ознак: довжина та ширина пелюсток:

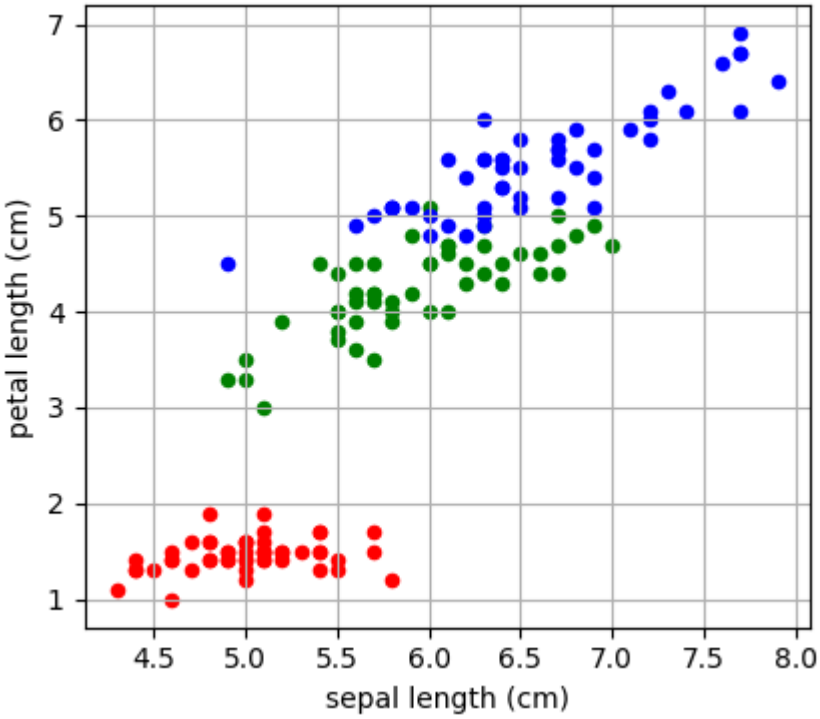


Рис. 8.12

Тобто, вхідні вектори матимуть дві числові ознаки $x = [width, length]^T$.

Існує кілька поширених алгоритмів побудови вирішальних дерев. Спочатку я розгляну найпростіший, який називається ID3 (Iterative Dichotomiser 3), розроблений Россом Куінланом в 1986 для завдання класифікації. Цей алгоритм використовує ентропію як інформативність і реалізує жадібну стратегію, тобто, у кожному вузлі дерева, починаючи з кореневого, знаходить таку ознаку j і такий поріг t, які дають найбільше значення показника $Q(R_m, j, t)$:

$$j_+, t_+ = argQ(R_m, j, t)$$

Потім, побудова рекурсивно повторюється кожної нової вершини.

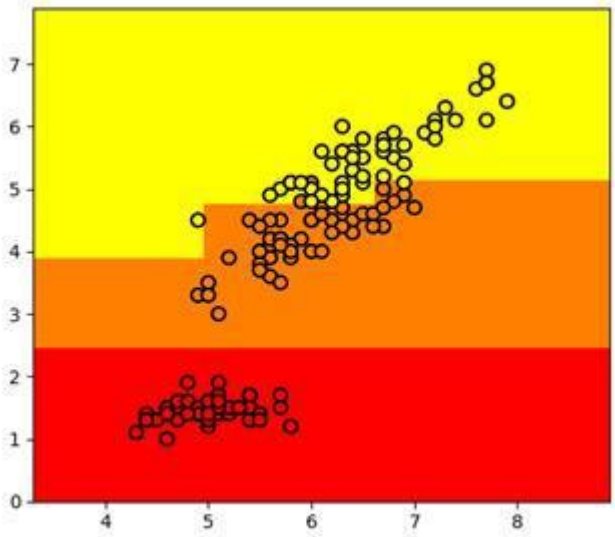


Рис. 8.13

Ось приклад розбиття навчальної множини деревом глибиною 4. Саме дерево має вигляд:

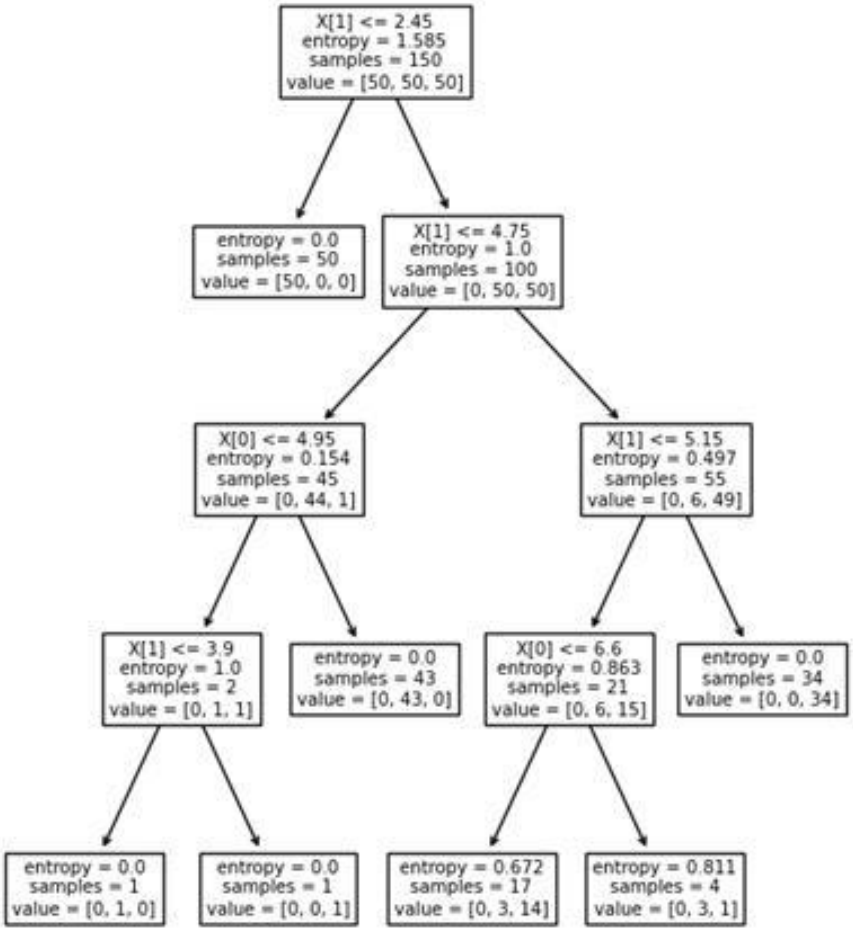


Рис. 8.14

Звідси добре видно, як відбувалося розбиття. Спочатку були відокремлені всі червоні об'єкти вибірки за другою ознакою (довжина пелюстки) з порогом $t = 2{,}45$ і предикатом:

$$B_{j,t}(x) = [x_1 \leq 2{,}45].$$

Якщо предикат виконується, то потрапляємо до листової вершини з нульовою ентропією та 50-ма представниками класу «червоних» об’єктів. До другої вершини потрапляє 100 об’єктів двох інших класів з ентропією, що дорівнює 1. Далі, ми ділимо об’єкти другої вершини дерева також за другою ознакою та рівнем 4,75. Формуються дві вершини. І так далі, поки або не буде досягнуто нульової ентропії і сформовано лист дерева, або глибина дерева досягне рівня 4. (Цей максимальний рівень був заданий при генерації даного вирішального дерева.) У кожній листовій вершині ми зберігаємо мітку класу з найбільшою кількістю представників.

Ось приклад роботи жадібного алгоритму ID3. Здавалося б, все прекрасно і ніщо не заважає нам використовувати його в різних завданнях. Однак практика показала, що жадібна стратегія побудови дерев часто не найкраща.

Простий контрприклад – завдання XOR, коли об’єкти двох класів зосереджені у різних кутах квадрата:

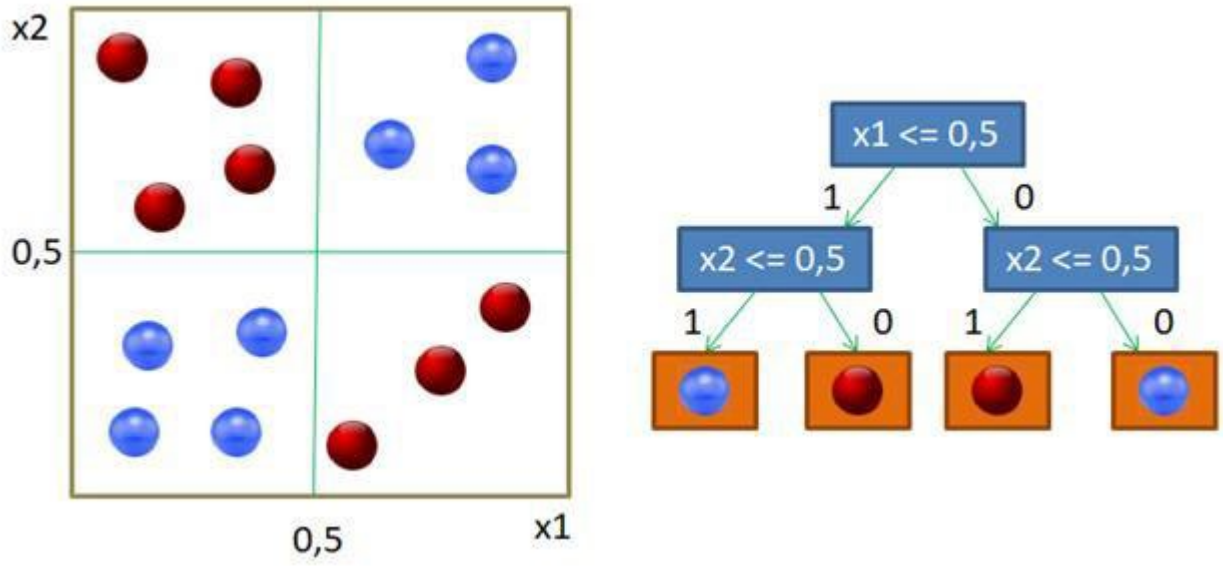


Рис. 8.15

Жадібна стратегія дає наступне вирішальне дерево:

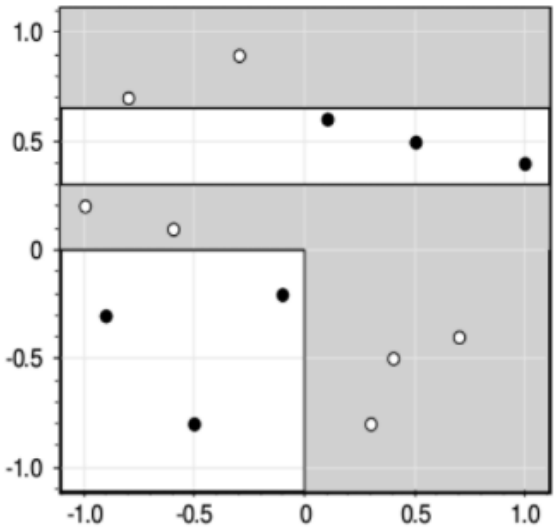


Рис. 8.16

Проте алгоритм ID3 відносно простий у реалізації, а тому є основою для більш складних підходів до побудови вирішальних дерев, про які ми поговоримо на наступних заняттях.

Критерії зупинки (методи регуляризації)

Коли виконується побудова вирішальних дерев за безліччю даних (навчальної вибірки), важливо визначити алгоритму критерії формування листових вершин. Один із них очевидний. Коли impurity вершини дорівнює 0, то в ній присутні представники лише одного класу, а тому подальший поділ виконувати не потрібно. Однак на практиці застосовують і інші критерії зупинки. До основних можна віднести такі:

– impurity дорівнює нулю чи менше деякого заданого значення;

– число об’єктів у вершині менше за задану величину;

– ймовірність правильної класифікації об’єкта більша за задану величину;

– досягнуто максимальної кількості листя в дереві;

– досягнуто максимальної заданої глибини дерева.

Звичайно, можуть бути використані й інші критерії, які лише часто застосовуються. Причому в процесі побудови дерева можна використовувати відразу кілька критеріїв і, як правило, так і роблять. Їх вибирають із розрахунку отримання хорошої узагальнюючої здатності навченого (результуючого) дерева, тобто критерії формування листових вершин можна розглядати як евристику регуляризації вирішального дерева.

Застосовувати критерії зупинки можна також по-різному. Або одночасно у процесі побудови дерева, для формування кожної поточної вершини. Такий спосіб називається pre-pruning або early stopping. Або побудувати дерево за жадібним алгоритмом без сильних обмежень, а потім провести його стрижку (pruning), тобто видаляти окремі вершини так, щоб якість його роботи залишалася приблизно на тому ж рівні.

Причому якість такого дерева слід перевіряти за відкладеною (тестовою) вибіркою (а не навчальною).

Переваги та недоліки жадібної стратегії

Отже, я думаю, загалом ідея побудови вирішальних дерев за алгоритмом ID3 вам зрозуміла. На закінчення відзначу переваги та недоліки такого підходу.

Переваги:

– інтерпретованість та простота класифікації (легко пояснити результат класифікації експерту).

– допустимі різнотипні дані та пропуски в даних.

– немає відмов від класифікації.

Недоліки:

– жадібна стратегія в більшості завдань надмірно ускладнює структуру дерева, тобто, призводить до його перенавчання.

– що далі від кореня дерева, то менше об’єктів у листових вершинах, а отже, нижча статистична надійність різних показників, наприклад, ймовірності появи того чи іншого класу.

– висока чутливість до шуму в об’єктах вибірки та критерію інформативності (impurity).

Деякі з наведених переваг та недоліків притаманні взагалі всім вирішальним деревам. Наприклад, простота інтерпретації результату, обробка перепусток даних (про це йтиметься далі), висока чутливість до шумів (викидів). Усе це наслідок самої структури логічних висновків з допомогою вирішальних дерев. На наступному занятті ми розглянемо методику усічення (pruning) перенавчених дерев для підвищення їх узагальнюючих здібностей

8.4. Усічення (prunning) дерева, обробка перепусток та категоріальних ознак

На попередньому занятті ми з вами розглянули алгоритм класифікації ID3, який реалізує жадібну стратегію під час побудови вирішальних дерев та використовує ентропію для обчислення інформативності (impurity) поточних вершин.

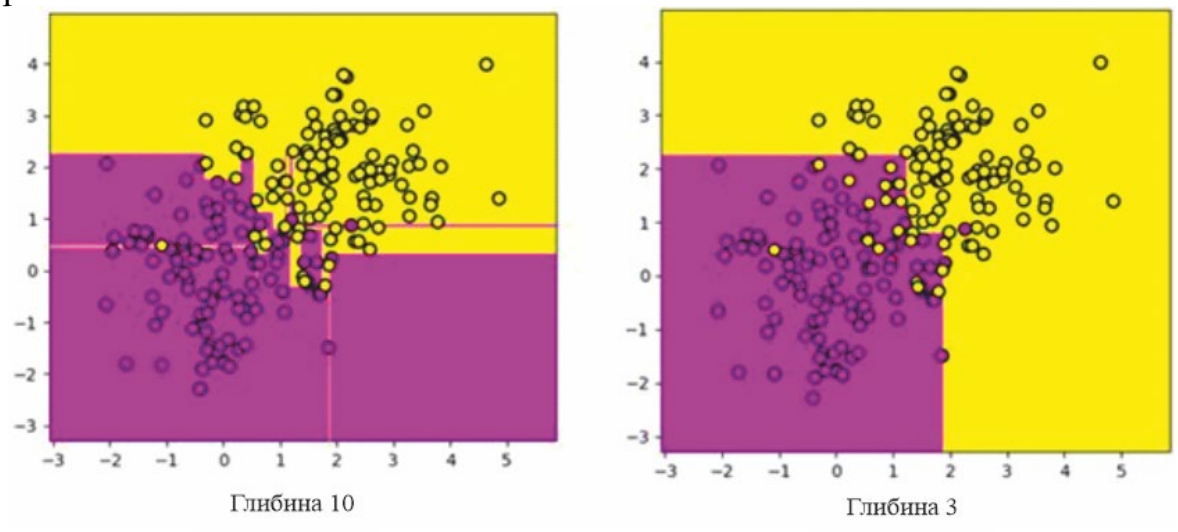


Рис. 10.17

Однак, часто дерево в результаті виходить сильно перенавченим, тобто занадто добре підганяється під дані навчальної вибірки, що різко знижує якість прогнозування відкладеної (тестової) вибірки. Розумним рішенням тут здається обмежити глибину вирішального дерева і таким чином підвищити його узагальнюючі здібності.

Як я вже зазначав, процедура усічення дерева після його побудови за навчальною вибіркою називається pruning. На відміну контролю глибини безпосередньо за його побудові. Цей процес вже зветься pre-pruning або early stopping. Зараз йтиметься саме про pruning – усічення дерева після його навчання (побудови).

Усічення дерева (pruning). Оскільки ми маємо справу з перенавчанням (overfitting), нам знадобиться контрольна вибірка. Саме на ній слід визначати якість прогнозу вирішального дерева. Контрольна вибірка, зазвичай, формується з 30% об'єктів вихідних розмічених даних, а 70% становить, власне, навчальна вибірка.

Ми перебираємо всі внутрішні вершини дерева: v O $V_{внутр}$

Опустимо деталі, в якому порядку це робиться: вершини в одних алгоритмах перебираються знизу нагору, проходячи все вище і вище до кореня дерева, а в інших, навпаки, від кореня – до листових.

Для кожної поточної внутрішньої вершини v ми визначаємо безліч об'єктів X_v^q контрольної вибірки, що дійшли до неї. Якщо виявляється, що це безліч порожнє (таке цілком може бути, тому що використовується відкладена

вибірка, а не навчальна, по якій будувалося дерево), то цю вершину перетворюємо на листову, так як висока ймовірність, що піддерево цієї вершини описує якісь деталі (можливо, викиди) навчальної вибірки та її слід відкинути (обрізати). У листову вершину прописуємо мітку класу, що відповідає найбільшій кількості представників об'єктів з навчальної вибірки.

Якщо ж безліч об’єктів X_v^q не порожнє, то обчислюємо помилки прогнозів для чотирьох можливих ситуацій усічення піддерева:

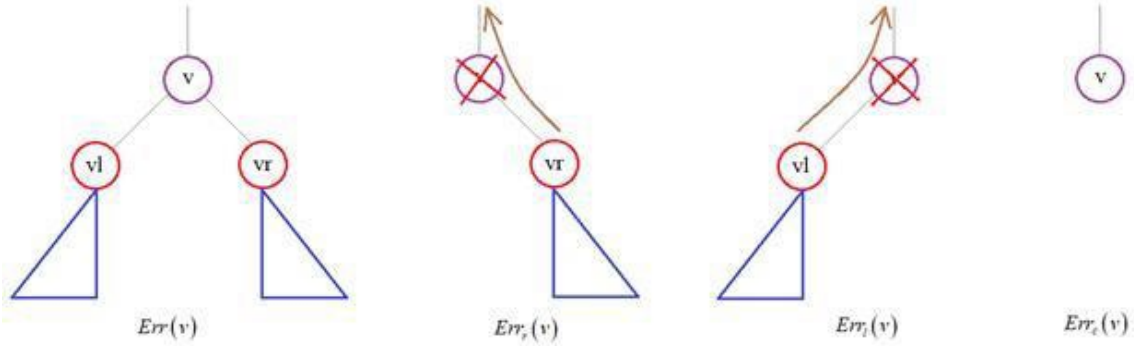


Рис. 8.18

Вибираємо варіант із найменшою помилкою, яка обчислюється за контрольною вибіркою. В результаті, ми або залишаємо все як є, або усікаємо праве або ліве піддерево, або перетворюємо вершину на листову.

Ось загалом ідея обрізки вирішального дерева. І така процедура рекомендується, коли дерево будується відповідно до жадібного алгоритму, т.к. висока ймовірність, що воно виявиться перенавченим.

Обробка перепусток у даних. На даному етапі ви повинні добре уявляти ідею побудови вирішальних дерев за жадібним алгоритмом та їх усічення для підвищення узагальнюючих здібностей. Але одне важливе питання у нас залишилося осторонь. Як виконувати обробку реальних даних (вхідних векторів *x*), якщо вони відсутні потрібні висновку ознаки?

Про що тут йдеться? Уявімо, що у нас є наступний фрагмент вирішального дерева. Щоб обробити вектор *x* потрібні ознаки, які, умовно, я позначив *j*, *k*, *m*. Припустимо, що об'єкти навчальної вибірки мали ці ознаки. Але в процесі експлуатації або на етапі тестування з'ясовується, що деякі об'єкти (рідкісні) не мають цих ознак, тобто виникають перепустки в даних. Така ситуація не є чимось винятковим і зустрічається на практиці. І потрібно вироблення деякої стратегії (підходу) для обробки рідкісних перепусток даних.

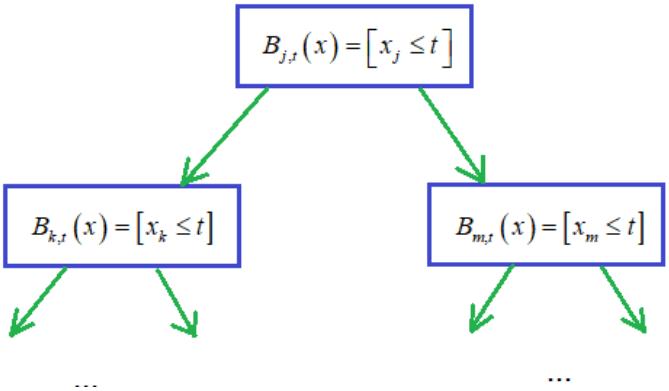


Рис. 8.19

Тут є кілька підходів розв'язання цього завдання. Я наведу математичний алгоритм, як прихильник суворої математики. Отже, у нас є навчальна вибірка X^l за якою було збудовано вирішальне дерево. І нехай у деяку вершину v потрапляє U об'єктів вибірки X^l. Також ми знаємо, підмножини U_1, U_2,…, U_k об'єктів, які перейшли з кожної гілки вершини v відповідно до заданого предикату. Я тут розглядаю загальну структуру вирішального дерева з багатьма нащадками.

Для бінарних дерев достатньо покласти $k = 2$. На основі цих даних легко обчислити частотну ймовірність того, що довільний вектор x перейде по одній із гілок:

$$q_{vj} = -\frac{|U_j|}{|U|}, \quad j = 1,\ldots,k$$

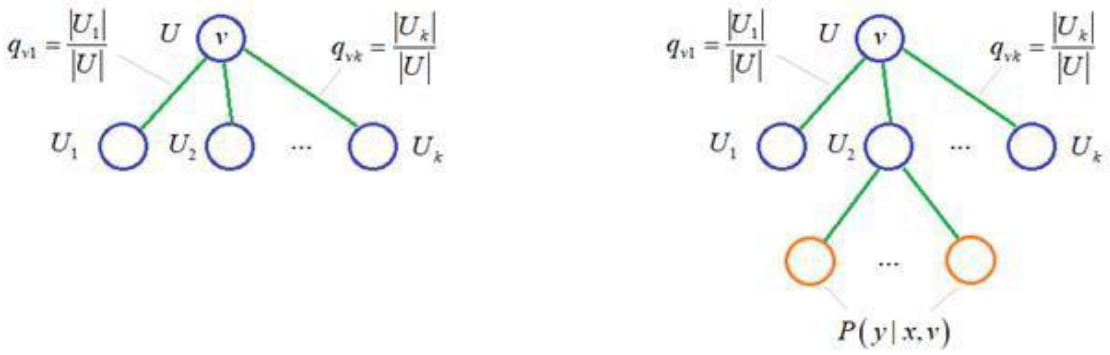


Рис. 8.20

Якщо вершина v є листовою, то ймовірність того, що вектор x належить класу y.

Можна визначити за формулою: $P(y \mid x,v) = \frac{1}{|U|}\sum_{x_i \in U} [y_i = y]$, $\forall v O V_{лист}$

Тобто ми в листовій вершині підраховуємо представників класу y і ділимо на загальну кількість об'єктів U, які потрапляють до неї.

Знаючи ймовірність $P(y \mid x,v)$, $\forall v O V_{лист}$ і q_{vj}, $j = 1, \ldots, k$ ми можемо рекурентно їх перераховувати для будь-якої проміжної вершини за формулою $P(y \mid x,v) = \sum_{j \in J} q_{vj} P(y \mid x,v_j)$.

Тобто, проходимо гілками вершини v і підсумовуємо ймовірності листових вершин $P(y \mid x,v)$. Якщо наступна вершина не листова, то для неї застосовується та сама формула з рекурсії. В результаті ми отримуємо ймовірність того, що деякий вектор x належить класу y.

А далі все просто. Якщо у будь-якій проміжній вершині v для вектора x не існує необхідної для предикату ознаки j, то він відноситься до класу, у якого найбільша ймовірність:

$$a(x) = argP\,(y \mid x,v).$$

На наш погляд, це досить стрункий та логічний алгоритм обробки пропущених ознак для вхідних векторів x.

Обробка категоріальних ознак

Досі ми вважали, що вектор $x = [x_1, x_2,\ldots, x_n]^T$ представлений числовими ознаками і тому, їм можна було визначати предикати виду: $B_{j,t}(x) = [x_j \leq t]$.

Але не рідко зустрічаються завдання, коли будь-яка ознака (а може бути і кілька) ставляться до категоріальних, тобто приймають одне із заданої множини значень. Наприклад, це може бути стать людини, або місто, в якому він живе, і тобто:

$$x_j \text{ O } \{\text{'чоловічий', 'жіночий'}\}$$
$$x_j \text{ O } \{\text{'Дніпро', 'Рівно', 'Одеса', 'Харків',}\ldots\}$$

Як правильно їх обробляти за допомогою дерев? Давайте спочатку узагальним запис для категоріальних ознак, таким чином:

$$C = [c_1, c_2,\ldots, c_M], \quad x_j \text{ O } C.$$

Тут C – це безліч з M можливих категорій. І j-а ознака вектору x набуває одного з M значень.

Тоді, перше що спадає на думку, це визначити вершину вирішального дерева з M нащадками для обробки такої ознаки:

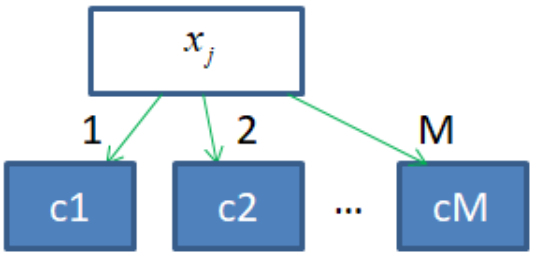


Рис. 8.21

Якщо ми тепер обчислюватимемо показник якості розбиття для такої вершини так, як це робили для звичайних числових ознак, то отримаємо вираз виду:

$$Q(R_m) = H(R_m) - \frac{|R_1|}{|R_m|} H(R_1) - \frac{|R_2|}{|R_m|} H(R_2) - \ldots - \frac{|R_M|}{|R_m|} H(R_M) \rightarrow \max.$$

Нагадуємо, що тут $H(R_m) \rightarrow \min$, це є impurity (інформативність) вершини дерева.

Так ось, вираз $Q(R_m)$ при категоріальних ознаках підступно тим, що майже завжди вигравatиме (приймати більше значення) проти числовими ознаками. Тому що інформативність при великій кількості дочірніх вершин швидко зменшуватиметься. Крім того, таке розгалужене дерево утворює набагато складнішу ієрархію взаємозв'язків і, як наслідок, дуже швидко перенавчається – підлаштовується під навчальну вибірку.

Існують методики обходу цих проблем, наприклад, через додавання регуляризаційного множника у формулі $Q(R_m)$. Але це часто призводить до інших додаткових проблем. Тому тут краще спробувати звести завдання обробки категоріальних ознак до завдання бінарної класифікації, коли вершина, як і раніше, матиме лише два нащадки.

Знову ж таки, в самому простому варіанті, ми можемо просто розбити все безліч категорій на дві безлічі, що не перетинаються:

$$C = C_l \cup C_r.$$

А потім визначити предикат, наприклад, виду:

$$B_j(x) = [x_j \leq C_l] \text{ чи навпаки, } B_j(x) = [x_j \leq C_r].$$

Тобто тут нам необхідно визначити (підібрати) вміст множин C_l, C_r для формування предикату $B_j(x)$.

Якщо вирішувати це завдання «в лоб», то отримаємо $N = 2^{M-1} - 1$ варіантів розбиття.

Як ви розумієте, навіть за невеликої кількості M (від 10 і вище) ми отримуємо величезну кількість варіантів. Вибрати їх краще простим перебором виявляється обчислювально дуже складно.

Ви можете сказати, а чому б нам не сприймати номери категорій як числову ознаку і не використовувати його поряд з іншими? І формувати предикат вже знайомого нам виду:

$$B_{j,t}(x) = [x_j \leq t].$$

Це цілком розумна ідея лише тут є один нюанс. Уявімо, що наше вирішальне дерево пов’язане з визначенням (прогнозом) рівня оплати праці для деякого індивіда, представленого ознаками вхідного вектору x. І для j-ї категоріальної ознаки маємо наступне співвідношення між містами та рівнем оплати:

Таблиця 8.1

Номер	1	2	3	4	5
Категорія	Рівне	Дніпро	Львів	Харків	Київ
Середня з/п	25 000	40 000	35 000	75 000	45 000

Як бачите, тут рівень заробітної плати то зростає, то зменшується з номером категорії. Тому збільшення порога t тут не матиме прямий взаємозв'язок з розміром заробітку.

Наприклад, якщо взяти предикат: $B_{j,t}(x) = [x_j \leq 4]$, то остання 5-та категорія не буде відповідати максимальній оплаті. А числові ознаки повинні інтерпретуватися саме так: що більше номер категорії, то вища оплата праці.

За цим прикладом, я думаю, ви вже здогадалися, що достатньо впорядкувати категорії за цільовим значенням, і тоді їх номери можна сприймати як числові ознаки:

Таблиця 8.2

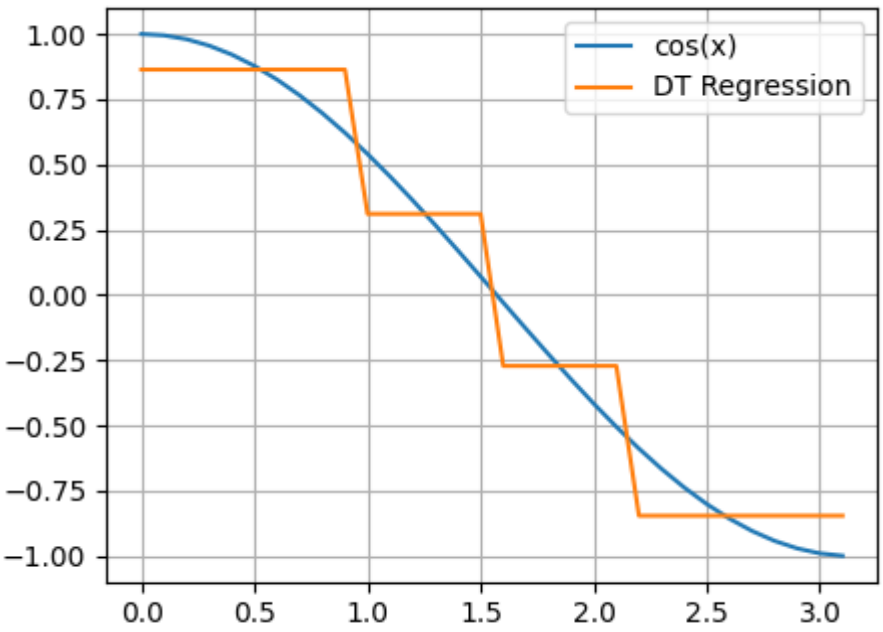
Номер	1	2	3	4	5
Категорія	Рівне	Дніпро	Львів	Харків	Київ
Середня з/п	25 000	35 000	40 000	45 000	75 000

Ось загальний принцип, покладений основою обробки категоріальних ознак як числових. На практиці саме він найчастіше і використовується.

8.5. Вирішальні дерева у задачах регресії. Алгоритм CART

На попередніх заняттях ми розглядали вирішальні дерева для завдань класифікації. Однак, у ряді випадків їх застосовують і для завдань регресії, коли алгоритм на виході формує одне речове значення (або кілька значень) для кожного вхідного вектора x. Тобто, у листі такого дерева зберігаються відповідні речові числа.

Давайте, для певності, я одразу наведу приклад такого вирішального дерева.



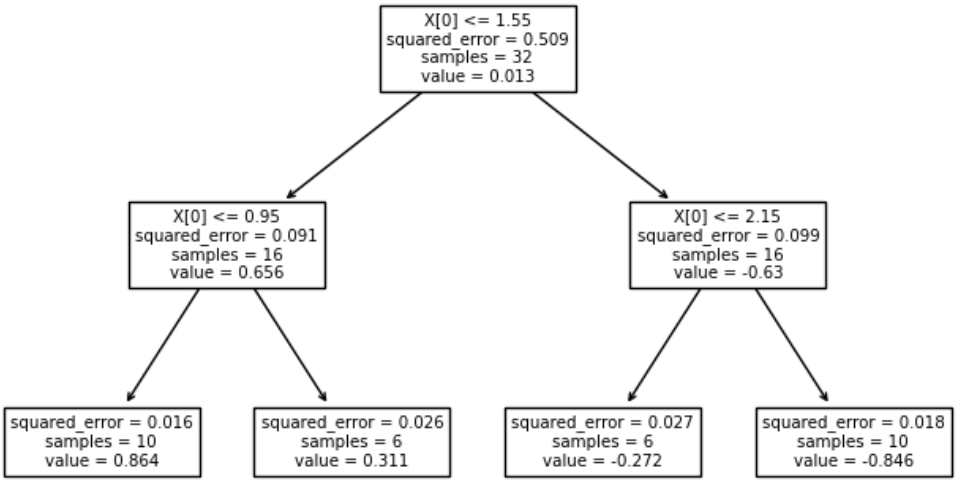


Рис. 8.22

Дивіться, спочатку наші дані є точки функції cos(x):

$$X^l = \left\{(x_i, y_i = cos(x_i))_{i=1}^{l}\right\}.$$

Тут аргумент функції cos() – це ознака, а значення функції у кожній точці – цільові змінні. Потім, за ознакою x (аргументу функції) виконується поділ вихідної множини точок на непересічні підмножини. В результаті, при глибині дерева два, отримуємо чотири підінтервали та чотири листи. Причому, у кожному аркуші зберігається одне константне речове значення, яким замінюються всі значення виділеного підмножини. Тому ми бачимо на графіку ступінчасту функцію при апроксимації косінусоїди вирішальним деревом.

Ось загальна ідея використання вирішальних дерев для завдань регресії. І тут виникають два головні питання:

За яким критерієм оцінювати якість поділу на підмножини (тобто використовувати як міру невизначеності – impurity)?

Як обчислювати речові значення отриманих листових вершинах?

Хороша новина, що на обидва ці питання є єдина відповідь. Я почну з останнього. Часто у завданнях регресії потрібно забезпечити мінімум середньоквадратичної помилки прогнозу:

$$H(R) = \sum_{(x_j, y_j) \in R} (a(x_i) - y_i)^2.$$

Тут $a(x)$ – відповідь (прогноз) моделі на вхідний вектор ознак x; R – безліч об'єктів x_i, що дійшли до деякої листової вершини v і відповідні їм цільові значення $\{y_i\}$.

Так як при використанні вирішальних дерев кожне підмножина R апроксимується деяким константним значенням:

$$b_v = a(x_i),$$

то для мінімізації квадратичного критерію, величину b_v слід обчислювати, як середнє арифметичне від усіх цільових значень y_i O R, що потрапили в листову вершину v дерева:

$$b_v = \frac{1}{|R|} \sum_{y_i \in R} y_i.$$

Можна легко довести, що ця величина найкраще описуватиме значення $\{y_i\}$ підмножини R за квадратичного критерію.

Оскільки величина $H(R)$ залежить від безлічі R. А отже, від способу розбиття деревом вихідної навчальної вибірки, то її було б логічно використовувати як міру інформативності (impurity) і вибирати в кожній проміжній вершині розбиття так, щоб максимізувався інформаційний виграш:

$$Q(R_m, j, t) = H(R_m) - \frac{|R_r|}{|R_m|} H(R_l) - \frac{|R_r|}{|R_m|} H(R_r) \rightarrow \max.$$

Тобто, ми будемо вибирати поріг t так, щоб impurity лівого та правого підмножин R_l, R_r були якнайменше.

Ось приклади розбиття вихідної послідовності деревом глибиною від одиниці до чотирьох:

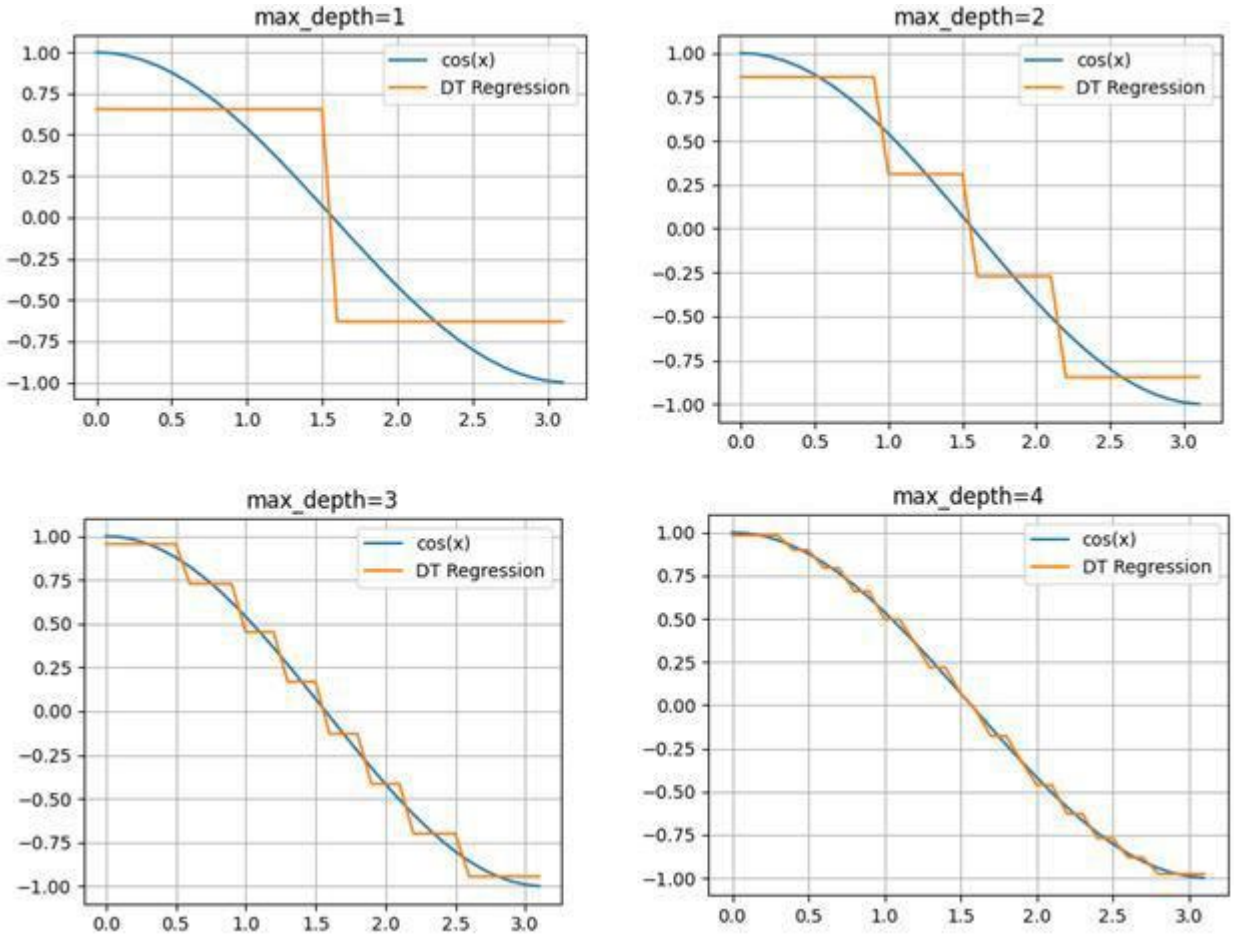


Рис. 8.23

Як бачите, спочатку виділяються два рівні підмножини і дві константи, які грубо описують графік косінусоїди. І цей опис найкращий з погляду квадратичного критерію. При збільшенні глибини отримуємо все більше і більше інформації про вихідний сигнал, і для глибини 4 цілком вгадується графік косінусоїди.

8.6. Випадкові дерева та випадковий ліс. Бутстреп та бегінг

На попередніх заняттях ми з вами познайомилися з ідеєю побудови вирішальних бінарних дерев для завдань класифікації та регресії. Однак на практиці у чистому вигляді їх майже не використовують. Але активно застосовують в ансамблевих методах, які включають два дуже ефективні підходи: бегінг і бустінг. Саме про бегінг з використанням дерев і йтиметься на цьому занятті.

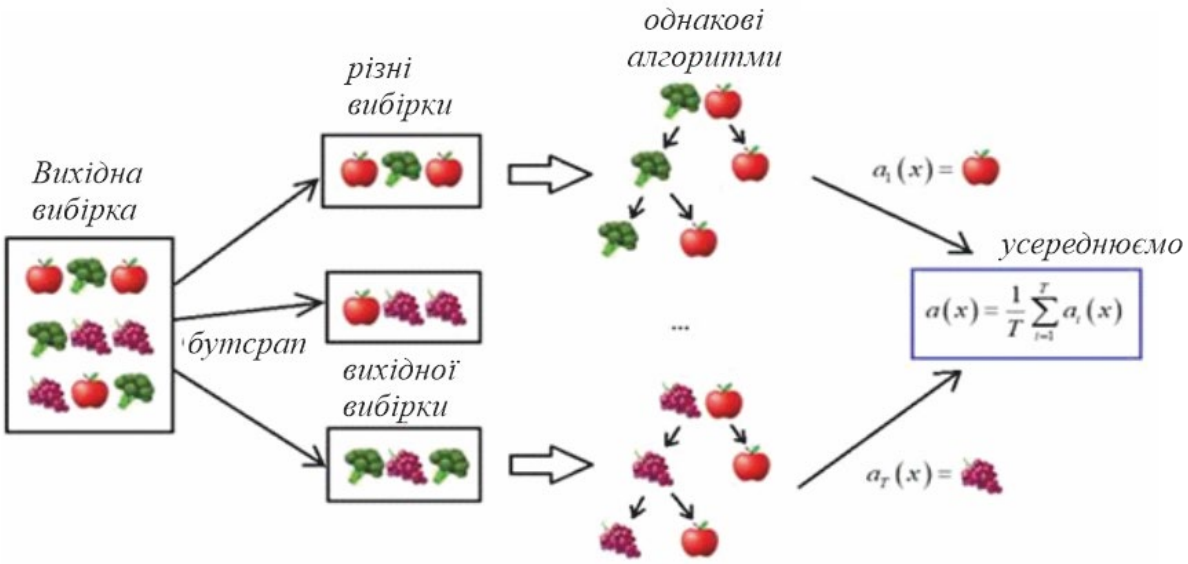


Рис. 8.24

Ідея алгоритму дуже проста і була запропонована Лео Брейманом у 1994 році. Цей підхід виявився дуже ефективним і використовується досі.

Отже, припустимо спочатку ми маємо деяка навчальна вибірка:

$$X^l = \left\{(x_i, y_i)_{i=1}^{l}\right\}.$$

На її основі ми хочемо сформувати декілька різних і, у загальному випадку, незалежних алгоритмів обробки вхідного вектора x:

$$a_1(x),\ a_2(x),\ \ldots,\ a_T(x).$$

А, потім, усереднити відповіді кожного алгоритму на формування загального рішення.

Напевно, тут одразу постає питання, навіщо нам штучно створювати кілька незалежних алгоритмів, а потім усереднювати їхні виходи? Насправді цьому є цілком логічне пояснення. Одна відома історія свідчить, як у 1906 році математик Френсіс Гальтон відвідав ринок і побачив, як продавець бика запропонував покупцям невелику лотерею: вгадати точну вагу цього бика, який важив 1198 фунтів. Від селян посипалися різні припущення.

Але жоден не назвав точного значення. Однак, усереднивши всі відповіді, вийшло значення 1197 – дуже близьке до істинного. Ось так «мудрість натовпу» вирішила це непросте завдання.

Ця ж ідея закладена в бегінгу, коли ми формуємо безліч незалежних алгоритмів, кожен видає свій варіант відповіді, а потім ми його усереднюємо, щоб отримати більш точне значення.

Цей ефект зменшення помилки за умови усереднення відповідей легко пояснити з позиції теорії ймовірностей. Припустимо, що правильне значення – вага бика – це величина θ. Тоді відповіді селян можна подати у вигляді наступної адитивної моделі:

$$x_1 = \theta + \eta_1$$
$$x_2 = \theta + \eta_2$$
$$\dots$$
$$x_T = \theta + \eta_T.$$

Тут $\{\eta_j\}$ – випадкова величина.

Зрозуміло, що помилка у відповідях кожного складає:

$$\varepsilon_j = \theta - x_j, \quad j = 1, 2, \dots, T$$

з дисперсіями помилок:

$$\sigma_j^2 = E\{\varepsilon_j^2\}.$$

Для простоти, я вважатиму, що всі помилки мають єдину дисперсію (розкид значень):

$$\sigma^2 = \sigma_1^2 = \sigma_2^2 = \dots = \sigma_T^2.$$

Тоді, дисперсія усередненої відповіді:

$$y = \frac{1}{T}\sum_{j=1}^{T} x_i$$

за умови незалежності кожної СВ $E\{x_i x_j\} = 0, \quad \forall i \neq j$ та незміщеності помилки (нульового середнього):

$$E\{\varepsilon_j\} = 0, \quad j = 1, 2, \dots, T$$

дорівнює:

$$E\{(\theta - y)^2\} = E\left\{\left(\theta - \frac{1}{T}\sum_{i=1}^{T} x_i\right)^2\right\} = E\left\{\left(\frac{1}{T}\sum_{i=1}^{T} (\theta - x_i)\right)^2\right\} =$$

$$= \frac{1}{T^2} \cdot E\left\{\left(\sum_{i=1}^{T} (\theta - x_i)\right)^2\right\} = \frac{1}{T^2} \cdot \sum_{i=1}^{T} \underbrace{E\{(\theta - x_i)^2\}}_{\sigma^2} = \frac{1}{T^2} \cdot T \cdot \sigma^2 = \frac{\sigma^2}{T}$$

Тобто дисперсія σ^2 однієї окремої відповіді (алгоритму) після усереднення всіх результатів, зменшується в T разів. Це означає, що підсумкова величина стає точнішою за T незалежних відповідей кожного алгоритму.

Зверніть увагу, я тут постійно наголошую – незалежних відповідей. Якщо відповіді (припущення про вагу бика) залежатимуть одна від одної, то просте усереднення стане не найкращим підходом, а деяких випадках може навіть погіршити окремі результати. Тому незалежність роботи алгоритмів за бегінга є ключовою умовою.

Бутстреп (bootstrap)

То як нам сформувати T незалежних алгоритмів, використовуючи лише одну навчальну вибірку? Тут є кілька ідей, але в бегінгу використовується дуже простий підхід, який зветься бутстреп (bootstrap).

Суть бутстрапа полягає у формуванні T нових навчальних вибірок на основі однієї вихідної $X^l = \left\{(x_i, y_i)_{i=1}^{l}\right\}$.

Для цього випадково вибирається k-й елемент (x_k, y_k) вибірки та копіюється в нову (у колишній він залишається, не видаляється). Потім ця операція повторюється m разів – за заданим розміром нової вибірки. Так формується нова навчальна вибірка, що складається з елементів вихідної з деякими повтореннями, тому що цілком можна кілька разів випадково відібрати один і той самий елемент. Після формування однієї вибірки, переходять до формування наступної і так T разів для T вибірок, які, очевидно, дещо відрізнятимуться один від одного. Це ідея бутстрепу.

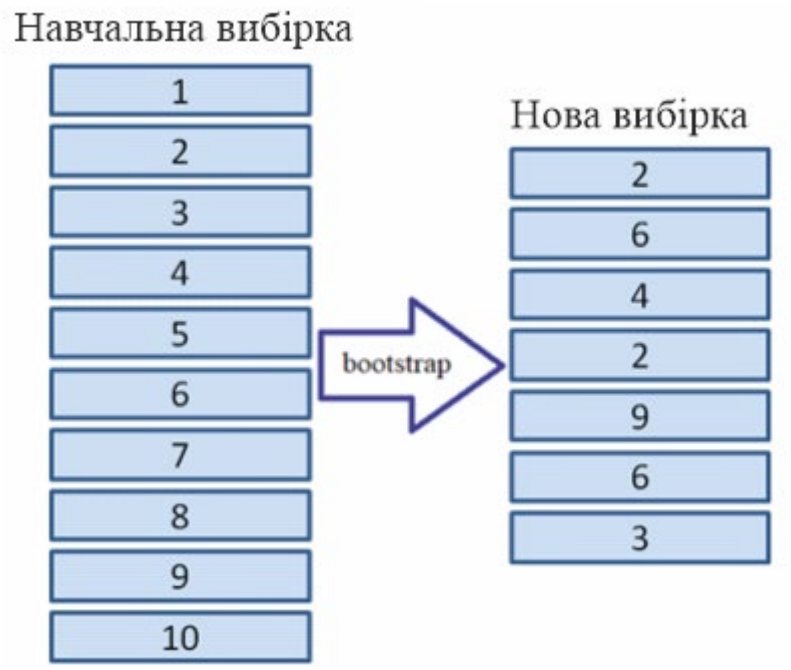


Рис. 8.25

Можна показати, що формуючи, таким чином, нову навчальну вибірку завдовжки l, вона буде використовувати, в середньому, $\left(1 - \frac{l}{e}\right) \cdot 100\% \approx 63{,}2\%$ образів з вихідної вибірки (інші повторюватимуться). Це означає, що частину вибірки в 36,8% можна використовувати як відкладену для перевірки якості алгоритму.

І це знайшло вираз у критерії під назвою out-of bag, коли ми визначаємо кількість помилок щодо об’єктів x_i, які не беруть участь у навчанні того чи іншого дерева:

$$out - of - bag(a) = \sum_{i=1}^{l} [sign(\sum_{t=1}^{T} [x_i \notin U_t] b_t(x_i)) \neq y_i] \rightarrow \min$$

Тут U_t, $t = 1,\ldots,T$ - безліч об'єктів, що становлять навчальну вибірку для дерева t. В результаті out-of bag – це незміщена оцінка узагальнюючої здатності підсумкового алгоритму $a(x)$.

Бегінг з вирішальними деревами. Випадковий ліс

Власне, саме слово bagging походить від скорочення двох англійських слів: bootstrap aggregation. Отже, маючи T випадкових вибірок довжиною m елементів ($m < l$), ми можемо за ними отримати T алгоритми (класифікації або регресії).

Наприклад, використовуючи лінійну модель:

$$a(x) = \langle w, x\rangle = w^T \cdot x$$

можна сформувати T наборів вагових векторів $w_1, w_2,\ldots, w_T$, де кожна для своєї навчальної вибірки. В результаті отримуємо T різних алгоритмів:

$$a_j(x) = \langle w_j, x\rangle, \quad j = 1, 2, \ldots, T.$$

А потім, середня відповіді від них, формуємо загальний результат:

$$a\ (x) = \frac{1}{T}\sum_{j=1}^{T} a_j(x).$$

Або, влаштовуючи голосування, вирішуємо завдання класифікації:

$$a_k\ (x) = sign\left(\frac{1}{T}\sum_{j=1}^{T} a_j(x)\right).$$

Як бачите, теоретично все просто. Але тут є один важливий нюанс: алгоритми у своїй сукупності повинні охоплювати якнайбільше можливих наслідків для кожного вхідного вектора x і, крім того, формувати якомога незалежні відповіді. У цьому сенсі звичайні лінійні моделі не дуже придатні при композиції (зокрема, за усереднення відповідей).

Набагато найкращі результати дають вирішальні дерева, побудовані незалежно на кожній сформованій навчальній вибірці (на етапі бустрепу).

Як не дивно, відданість дерев до перенавчання відіграє позитивну роль. Вони, по-перше, виходять досить різноманітними і, по-друге, описують різні результати для вхідних векторів x. Потім, при усередненні результатів, ефект перенавчання природно нівелюється (зменшується) і підсумкове вихідне значення виявляється досить точним і стійким до окремих викидів. У ряді завдань точність виявляється вищою за всі інші підходи машинного навчання. Саме тому бутстреп швидко завоював свою популярність.

Щоб вирішальні дерева виходили ще більш різноманітними та формували менш залежні відповіді, пропонується при їх навчанні у кожній проміжній вершині випадковим чином відбирати деяку кількість ознак $m < n$ і вже серед них відбирати найкращі для розгалуження. Набори таких дерев називають випадковим лісом (random forest). Причому було показано, що в задачах класифікації число $m = \lfloor\sqrt{n}\rfloor$, а у завданнях регресії $m = \lfloor\frac{n}{3}\rfloor$. Є теоретичні викладки, чому це так (кому цікаво, можна знайти в літературі з машинного навчання). Самі ознаки і пороги для розгалуження вибираються, зазвичай, за критерієм Джині (він швидше обчислюється, ніж ентропійний і призводить до

тим же результатам). А усічень дерев уже не роблять, вони залишаються перенавченими.

Як я вже зазначав, узагальнення та стійкість вихідного значення визначатиметься усередненням незалежних відповідей від кожного дерева. Причому, для випадкового лісу криві якості на навчальній та перевірочній вибірках у середньому зменшуються зі збільшенням числа дерев T:

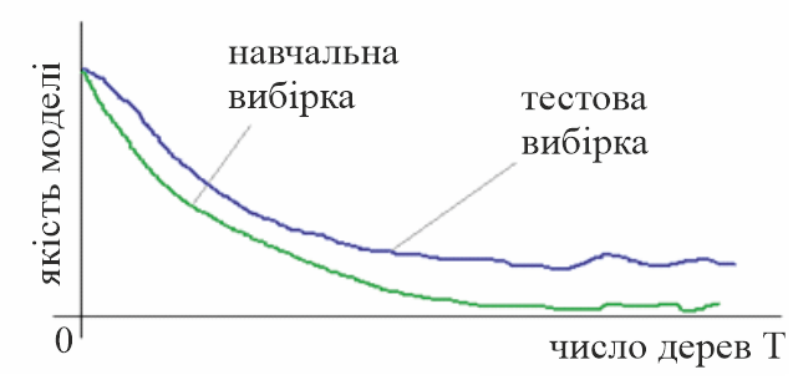


Рис. 8.27

Тобто зі зростанням числа T вирішальних дерев у випадковому лісі підсумкова модель не перенавчається, а лише досягає деякого граничного рівня якості.

Це дуже хороша властивість випадкового лісу, тому що ми, фактично, отримуємо алгоритм, в якому немає гіперпараметрів, що складно налаштовуються. Єдиний параметр T можна взяти, наприклад, 100, а потім 500 і порівняти результати на тестовій вибірці. Тобто підібрати його дуже просто. Інші гіперпараметри для побудови вирішальних дерев можна вибирати з позиції здорового глузду. Головне, щоб дерева виходили глибокими, гарантуючи малі усунення у відповідях.

Нагадаю, що під усуненням тут розуміється прагнення до нуля (у середньому) помилки прогнозу: $E\{(x - a(x))\} \to 0$.

Переваги та недоліки випадкового лісу

Як і будь-який алгоритм, випадкові ліси мають свої переваги та недоліки.

До переваг можна віднести:

- має високу точність прогнозів (на більшості завдань працює краще за лінійні алгоритми); точність можна порівняти з точністю бустингу;
- практично не чутливий до викидів у даних через випадкове семлювання вибірок методом бутсрепу;
- не чутливий до масштабування (і взагалі до будь-яких монотонних перетворень) значень ознак, пов'язаних з вибором випадкових підпросторів;
- не вимагає ретельного налаштування параметрів, що добре працює «з коробки»;
- здатний ефективно обробляти дані з великою кількістю ознак та класів;

- рідко перенавчається, на практиці додавання дерев майже завжди лише покращує композицію (до певного, граничного рівня);

- добре працює із пропущеними даними; зберігає хорошу точність, якщо більшість даних пропущена;

- можуть бути розширені до нерозмічених даних, що призводить до можливості робити кластеризацію та візуалізацію даних, виявляти викиди;

- легко розпаралелювати і масштабувати (збільшувати кількість дерев та їх глибину).

Недоліки випадкового лісу

- на відміну одного дерева, результати випадкового лісу складніше інтерпретувати;

- алгоритм працює гірше за багато лінійних методів, коли у вибірці дуже багато розріджених ознак (тексти, Bag of words);

- Випадковий ліс не вміє екстраполювати, на відміну від тієї ж лінійної регресії;

- алгоритм схильний до перенавчання на деяких завданнях, особливо на сильно зашумлені дані;

- для даних, що включають категоріальні змінні з різною кількістю рівнів, випадкові ліси упереджені на користь ознак з великою кількістю рівнів: коли ознака має багато рівнів, дерево буде сильніше підлаштовуватися саме під ці ознаки, так як на них можна отримати більш високе значення оптимізованого функціоналу (інформаційний виграш);

- більший розмір моделей, що виходять. Потрібно O(N·K) пам’яті для зберігання моделі, де K – кількість дерев.

На цьому ми завершимо це заняття. Сподіваюся, ви зрозумів, що з себе є ідея бегінга взагалі і як він працює для випадкового лісу.

8.7 Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Суть логічних методів класифікації.
2. Дайте визначення методам побудови вирішальних дерев.
3. Дайте визначення критерію Джині (Gini impurity).
4. Що таке помилка класифікації (Misclassification error).
5. Дайте визначення методам побудови жадібним алгоритмам ID3 (Iterative Dichotomiser 3).
6. Переваги та недоліки жадібної стратегії.
7. Дайте визначення методам усічення дерева (pruning).
8. Дайте визначення методам обробки перепусток у даних.

9. Дайте визначення методам Обробка категоріальних ознак.
10. Як працює Алгоритм CART(Classification and Regression Trees).
11. Дайте визначення методам бутстрепу.
12. Дайте визначення методам бегінгу.
13. Переваги та недоліки випадкового лісу.

8.8. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.

2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA

3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

4. Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale FPGA-based convolutional networks. In R. Bekkerman, M. Bilenko, and J. Langford, editors, Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press.

Лекція №9. Бустінг

Результати навчання, які формуються:

ДРН8: Будувати алгоритми AdaBoost для класифікації та навчання нейронних мереж.

9.1. Введення у бустінг (boosting). Алгоритм AdaBoost під час класифікації

На попередньому занятті ми з вами познайомилися з ідеєю бегінга (bagging), коли формується T незалежних простих алгоритмів та обчислюється вихідне значення у вигляді усереднення відповідей від кожного з них:

$$a(x) = \frac{1}{T}\sum_{j=1}^{T} a_i(x).$$

Ми підкреслювали, що такий підхід буде ефективним, якщо відповіді алгоритмів статистично незалежні між собою.

Математично це обмеження можна записати так:

$$E\{a_j(x) \cdot a_i(x)\}=0, \qquad \forall i,j : i \neq j.$$

Але в цій схемі є два тонкі моменти, де на практиці алгоритми $a_1(x)$, $a_2(x)$, $a_T(x)$, все ж таки видають кілька залежні значення (в загальному випадку, повної незалежності досягти неможливо); і другий – всі алгоритми зважуються з тим самим ваговим коефіцієнтом $1/T$, тобто, значимість кожного належить рівною.

Як ви розумієте, це дуже жорсткі умови і можна вважати, що вони завжди порушуються під час реалізації бегінга. Отже, ми не досягаємо потенційної якості при композиції алгоритмів.

Як можна було б покращити цей підхід? У 1995 році два вчені Йоав Фройнд (Yoav Freund) і Роберт Шапір (Robert Schapiro) запропонували більш загальну схему для композиції алгоритмів – з різними вагами:

$$a(x) = \sum_{t=1}^{T} a_t \cdot b_t(x) \ ,$$

де $\{\alpha_T\}$ – ваги (позитивні числа); $\{b_t(x)\}$ – прості алгоритми обробки вхідного вектора x. Причому вважатимемо, що алгоритми видають речове значення: $b_t(x) \in \mathbb{R}$.

Наприклад, завдання класифікації – це може бути величина відступу (margin), який для лінійних алгоритмів ми визначали як скалярне множення:

$M = \langle w, x\rangle = w^T \cdot x$

Або ймовірність приналежності до того чи іншого класу. І так далі. Тобто алгоритми $\{b_t(x)\}$ видають не жорсткі (кінцеві) рішення, а точніші (проміжні) речові. У цьому випадку композиція зазвичай призводить до кращих результатів.

Отже, при об'єднанні алгоритмів за формулою (1) нам необхідно вміти обчислювати вагові коефіцієнти і будувати набір T алгоритмів так, щоб мінімізувався заданий показник якості. Давайте для визначеності далі розглядатимемо завдання бінарної класифікації.

Тоді показник якості логічно вибрати, як число неправильно класифікованих образів:

$$Q_T = \sum_{i=1}^{l} [M_i < 0] = \sum_{i=1}^{l} [y_i \cdot a(x_i) < 0] = \sum_{i=1}^{l} \Big[y_i \cdot \sum_{t=1}^{T} a_t b_t(x_i)_{\backsim a(x_i)} < 0 \Big].$$

Нагадаю, що коли відступ для i-го образу менший за нуль ($M_t < 0$), значить сталася помилка класифікації і квадратні дужки (нотація Айзерсона) повернуть 1. Інакше повертається 0.

Зверніть увагу, ми не накладаємо жодних жорстких обмежень на алгоритми $\{b_t(x)\}$ та вагові коефіцієнти $\{a_t\}$. Єдине, вагові коефіцієнти зазвичай беруться позитивними.

Дивлячись на отриманий функціонал якості, ми розуміємо, що його потрібно оптимізувати і за ваговими коефіцієнтами, і алгоритмами:

$$Q_T = \sum_{i=1}^{l} [y_i \cdot \sum_{t=1}^{T} a_t b_t(x_i) < 0] \rightarrow min_{\{a_i\},\{b_i\}}.$$

Але це досить складне математичне завдання, тому Фройнд і Шапіро запропонували використати такі дві евристики:

– «жадібна» побудова композиції – пошук (вибір) поточного алгоритму b_t та вагового множника a_t при фіксації раніше знайдених алгоритмів та вагових множників: $b_1(x),\ldots,b_{t-1}(x)$ та $a_1,\ldots,a_{t-1}$. Тобто, на кожному кроці t ми обчислюємо лише один ваговий коефіцієнт a_t і навчаємо лише один алгоритм b_t переважно на тих образах навчальної вибірки, на яких попередні алгоритми показали слабкі результати.

– апроксимувати вихідний (пороговий) функціонал якості гладкою функцією втрат, що диференціюється, щоб ми могли чисельно або аналітично вирішувати оптимізаційне завдання.

Ці два підходи покладено основою практично всіх алгоритмів бустінгу (boosting). Тобто, ми намагаємося побудувати T щодо простих алгоритмів (класифікації чи регресії), а потім обчислити зважену суму їх виходів для покращення результуючого значення відповідно до обраного показника якості. Це і є сенс бустінгу алгоритмів.

Алгоритм AdaBoost. Щоб усе це краще зрозуміти, розглянемо конкретний приклад - алгоритм AdaBoost (скорочення від Adaptive Boosting), який і запропонували Фройнд з Шапіро стосовно завдань бінарної класифікації з мітками класів $Y = \{-1, +1\}$.

Для апроксимації порогового функціоналу якості було запропоновано використовувати експоненційну функцію втрат:

$$Q_T = \sum_{i=1}^{l} [M_i < 0] \leq \sum_{i=1}^{l} exp(-M_i) = \tilde{Q}_T.$$

Якщо розписати цей вираз, то отримаємо вираз:

$$Q_T \leq \tilde{Q}_T = \sum_{i=1}^{l} exp(-y_i \cdot \sum_{t=1}^{T} a_t b_t(x_i)).$$

Тепер дивіться. Відповідно до першої евристики, ми шукатимемо найкращий алгоритм $b_T(x)$ і вага a_T при фіксованих попередніх алгоритмах $b_1(x),\ldots,b_{T-1}(x)$ та вагах $a_1,\ldots,a_{T-1}$.

Тому перепишемо функціонал якості $\tilde{Q}_T$, наступним чином:

$$Q_T \leq \tilde{Q}_T = \sum_{i=1}^{l} exp\left(-y_i \cdot \sum_{t=1}^{T-1} a_t b_t(x_i)\right)_{w_i} \cdot \exp(-y_i \cdot \alpha_T \cdot b_T(x_i)).$$

Обидві формули – те саме, але на другий ми явно (окремо) винесли $b_T(x)$, a_T. І перед другою експонентою з'являється множник (вага) w_t, тому що попередні алгоритми та коефіцієнти зафіксовані.

Давайте проаналізуємо та подивимося, який сенс мають ці ваги. По-перше, вони обчислюються для кожного об'єкта $\{x_i\}$ навчальної вибірки, тобто це фактично ваги об'єктів. По-друге, множник буде тим більше, чим неактивнішим буде відступ (margin). Тобто, множник зростає для об'єктів, що погано класифікуються.

В результаті, поточний алгоритм $b_T(x)$ більшою мірою при навчанні враховуватиме ці погано класифіковані образи і налаштовуватиметься на них. Це і є реалізація принципу бустінгу – послідовно навчати алгоритми на об'єктах, на яких попередня композиція показала найгірші результати.

Далі, за алгоритмом, ваги $\{w_i\}$ нормуються під час пошуку кожного наступного алгоритму за формулою:

$$\tilde{w}_i = \frac{w_i}{\sum_{j=1}^{l} w_j}, \; i = 1, 2, \ldots, l.$$

Тобто ми робимо так, щоб у сумі нормовані ваги давали одиницю:

$$\sum_{i=1}^{l} \tilde{w}_t = 1.$$

Завдяки цьому вони не будуть йти в нуль або в нескінченність, а приймати цілком розумні значення. Це важливо під час реалізації алгоритму на комп'ютері. У результаті кожного поточного алгоритму $b(x)$ ми можемо вирахувати частку вірних і неправильних класифікацій:

$$N(b) = \sum_{i=1}^{l} \tilde{w}_i \cdot [b(x_i) \neq y_i];$$

$$P(b) = \sum_{i=1}^{l} \tilde{w}_i \cdot [b(x_i) = y_i] = 1 - N(b).$$

Очевидно, найкращий поточний алгоритм має мінімізувати частку невірних класифікацій:

$$b_T = N(b).$$

(Це відповідає обраному показнику якості). І далі, Фройнд з Шапіро довели, що найкраще (оптимальне) значення вагового коефіцієнта при експоненційній функції втрат і знайденого алгоритму $b_T(x)$ можна обчислити за простою формулою: $\alpha_t = \frac{1}{2} ln \frac{1-N(b_t)}{N(b_t)}$.

Все це і є основою алгоритму AdaBoost. Фактично, ми кожної ітерації (від 1 до T) навчаємо поточний алгоритм $b_j(x)$ на зважених об'єктах навчальної вибірки. Саме навчання виконується будь-яким відомим нам способом. А потім для нього обчислюється коефіцієнт a_j.

Зараз ми в деталях подивимося на роботу цього алгоритму, коли як алгоритми $\{b_j(x)\}$ виступають вирішальні дерева. Але спочатку я наведу псевдокод загального алгоритму AdaBoost:

Вхід: навчальна вибірка X_l і параметр T (число алгоритмів композиції)

Вихід: набір базових алгоритмів $b_1(x), \ldots, b_T(x)$ з вагами $\alpha_1, \ldots, \alpha_T$

1: початкова ініціалізація ваг об'єктів: $w_i = 1 / l$, $i = 1,...,l$,
2: для всіх $t = 1,...,T$
3: знайти найкращий поточний алгоритм за правилом:
$b_t = N(b)$
4: обчислити оптимальний коефіцієнт:
$\alpha_t = \frac{1}{2} ln \frac{1-N(b_t)}{N(b_t)}$
5: оновити ваги об'єктів:
$w_i = w_i \cdot \exp(-\alpha_t y_i b_t(x_i))$; $i = 1,...,l$
6: нормувати ваги об'єктів:
$w_{sum} = \sum_{i=1}^{l} w_j$; $w_i = w_i / w_{sum}$; $i = 1,...,l$

Реалізація алгоритму AdaBoost для задач класифікації на Python

А тепер, як і обіцяв, наведу реалізацію алгоритму AdaBoost на Python з використанням вирішальних дерев як базових алгоритмів $\{b_t(x)\}$.

Припустимо, у нас є навчальна вибірка у вигляді наступного набору точок двох класів:

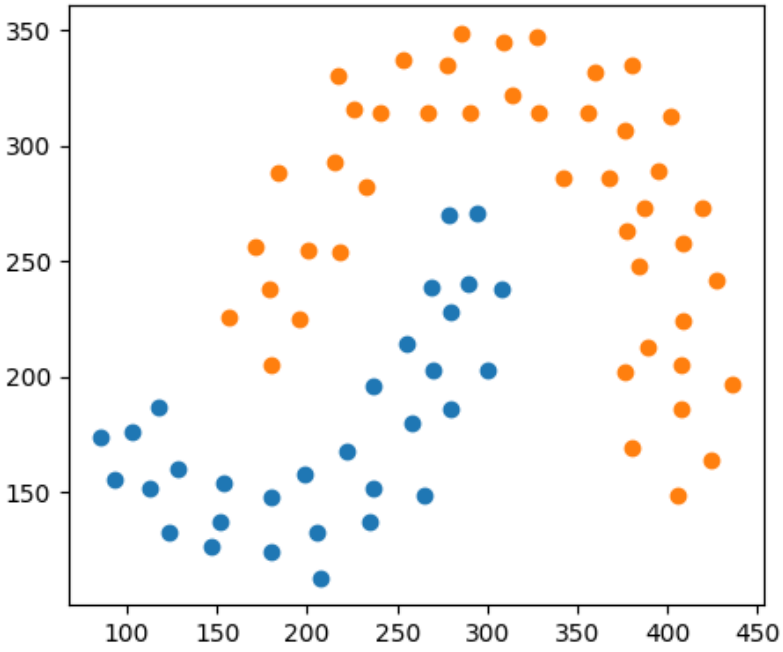


Рис. 11.1

Ми їх розділяти (класифікувати) композицією з T дерев глибиною два.
Програму можна розглянути: Додаток 3.

У результаті отримуємо наступні результати:

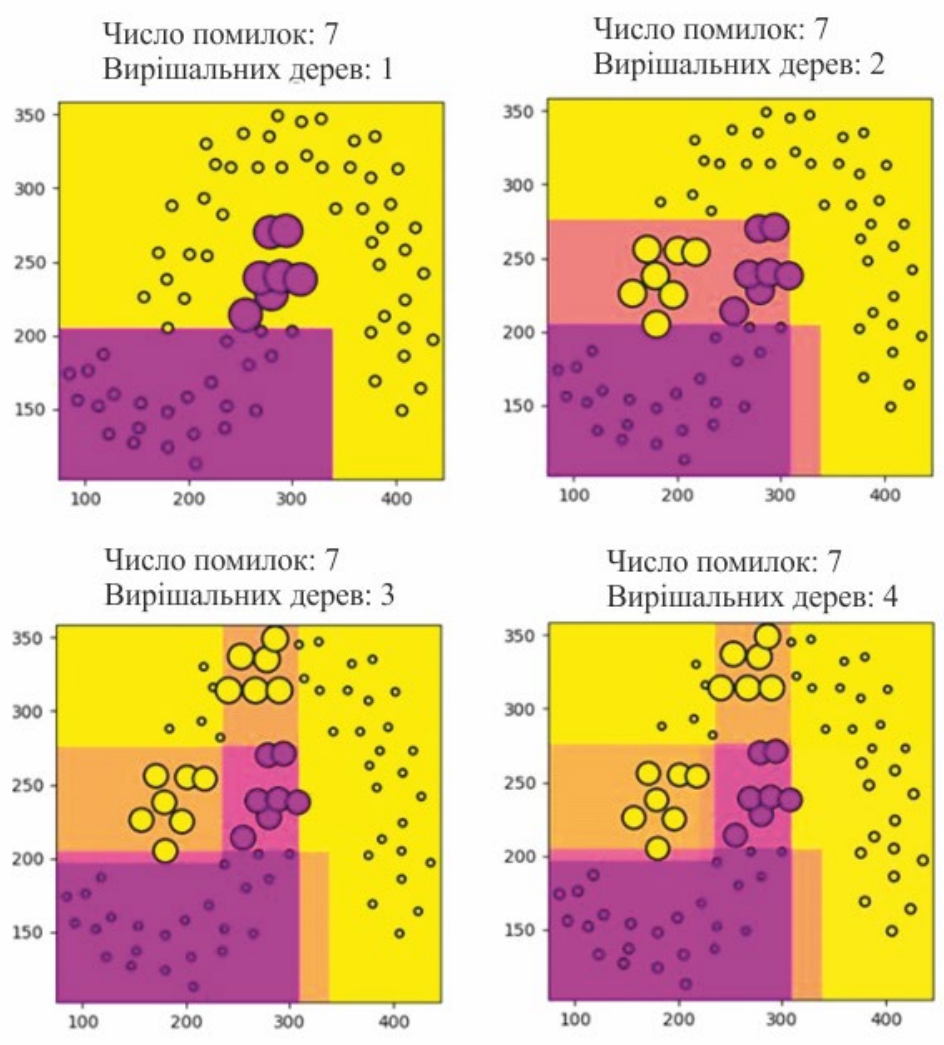


Рис. 11.2

Очікується, що при збільшенні числа дерев у композиції кількість помилок також зменшується. При цьому ми використовували дуже прості дерева рівня два. Кожен з них окремо не зміг би впоратися із завданням, але в композиції виходить безпомилкова класифікація на виборці. І, можливо, тут у вас виникне питання.

Якщо алгоритм так добре підлаштовується під навчальну вибірку, то, можливо, він перенавчився і на тестових даних буде показувати значно гірші результати? Саме так передбачали на самому початку появи бустінгу і вважали, що чим більше взяти алгоритмів у композиції, тим сильніше він підлаштовуватиметься під навчальну вибірку і, відповідно, сильніше перевчитися. Але на подив дослідників картина виявилася кардинально іншою:

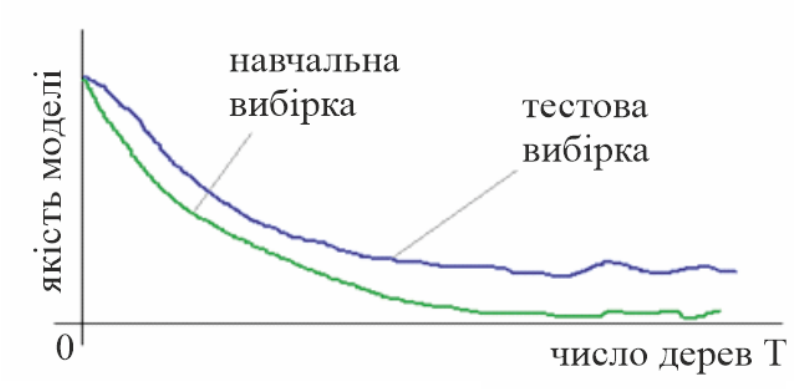


Рис. 11.3

Тобто бустінг поводиться так само як і бегінг зі збільшенням числа алгоритмів. І це дуже чудово! Це означає, що ми можемо отримувати дуже хороші результати класифікації або регресії, просто збільшуючи кількість простих алгоритмів, що входять до бустінгу. Завдяки цій властивості він набув дуже широкої популярності і є буквально настільним інструментом інженерів у галузі машинного навчання.

Наприклад, знаменитий Матрікснет використовує бустінг над вирішальними деревами для формування ранжування у пошуковій видачі. Щоправда там використовується алгоритм не AdaBoost, а досконаліша техніка. Але про деякі ці аспекти ми поговоримо вже на наступних заняттях.

9.2. Алгоритм AdaBoost у задачах регресії

На попередньому занятті ми з вами побачили, як працює алгоритм AdaBoost (Adaptive Boosting) стосовно завдань бінарної класифікації з мітками класів

$Y = \{-1, +1\}$. Але чи залишається питання, як можна використовувати цей підхід у завданнях регресії? Саме про це зараз і йтиметься.

Я нагадаю, що при бустінгу ми навчаємо T алгоритмів, а потім обчислюємо зважену суму:

$$a(x) = \sum_{i=1}^{T} a_i b_i(x).$$

Причому пошук терезів і алгоритмів виконується за «жадібним» принципом, де ми фіксуємо всі раніше знайдені елементи $b_1(x),\ldots, b_{T-1}(x)$ і $\alpha_1,\ldots, \alpha_{T-1}$ для знаходження поточних $b_T(x)$ і α_T.

Оскільки вирішуємо завдання регресії, критерій якості зазвичай формують під час використання квадратичної функції втрат:

$$Q_T = \frac{1}{2}\sum_{i=1}^{l} (y_i - a(x_i))^2 \to \min$$

Тобто ми проходимо по всіх об'єктах навчальної вибірки $X^l = \left\{(x_i, y_i)_{i=1}^{l}\right\}$ та обчислюємо суму квадратів помилок.

Якщо розписати величину $a(x)$, то отримаємо:

$$Q_T = \frac{1}{2}\sum_{i=1}^{l} \left(y_i - \sum_{j=1}^{T} a_j b_j(x_i)\right)^2 =$$

$$\frac{1}{2}\sum_{i=1}^{l} \left(y_i - \sum_{j=1}^{T-1} b_j(x_i) - b_T(x_i)\right)^2.$$

Тут можна помітити, що, по-перше, завдання лінійної регресії множники $\{\alpha_j = 1\}$ можна відкинути, т.к. вони лише визначають масштаби вихідних відповідей алгоритмів.

Цей масштаб може формувати і сам алгоритм $\tilde{b}_j(x) = a_j b_j(x)$.

І, по-друге, при фіксованих попередніх вагах та алгоритмах, ми фактично на кроці T мінімізуємо залишкові значення:

$$S_{i,T-1} = y_i - \sum_{j=1}^{T-1} b_j(x_i),\ i = 1, 2, \ldots, l.$$

Далі нам потрібно визначитися з вибором сімейства алгоритмів $\{b_j(x)\}$.

Завдання регресії. Частим випадком є вирішальні дерева, як і завдання класифікації. Взагалі, бустінг над вирішальними деревами – це класика – найчастіший варіант застосування. Хоча, можна брати і будь-які інші сімейства, наприклад, набір лінійних алгоритмів або будь-які інші. Головне, щоб ми могли відносно швидко (з обчислювальної точки зору) формувати, а потім використати ці алгоритми.

Отже, якщо вибрати вирішальні дерева, то в задачах регресії, при квадратичній функції втрат, у листках зберігаються середні значення цільових міток $\{y_i\}$ об'єктів, які у них потрапляють. Це саме те, що нам потрібно.

Якщо уявити, що до певної листової вершини потрапили M об'єктів $R_v = \left\{(x_i, y_i)_{i=1}^{M}\right\}$ навчальної вибірки, то оптимальне значення для їх опису, можна знайти з рівняння:

$$Q = \frac{1}{2}\sum_{i \in R} (y_i - c)^2 \to \min;\ \frac{dQ}{dc} = \frac{1}{2}\sum_{i \in R} (y_i - c) = 0,$$

звідки

$$c = \frac{1}{|R_v|}\sum_{i \in R} y_i.$$

Зауважте, що при іншій функції втрат, величина с може обчислюватися по-іншому.

У результаті, псевдокод алгоритму AdaBoost стосовно завдань регресії, матиме вигляд:

Вхід: навчальна вибірка X^l та параметр T (число алгоритмів у композиції)

Вихід: набір базових алгоритмів $b_1(x),\ldots, b_T(x)$

1: початкова ініціалізація залишків: $S_{i,0} = y_i,\ i = 1,\ldots,l$

2: для всіх $t = 1,\ldots,T$

3: знайти найкращий поточний алгоритм за правилом:

$b_t = Q_t$

4: оновити залишки:

$S_{i,t} = S_{i,t-1} - b_t(x_i),\ \ i = 1,\ldots,l$

Реалізація алгоритму AdaBoost для задач регресії на Python

Як бачите, все досить просто. Давайте реалізуємо цей алгоритм на Python. Припустимо, що нам потрібно апроксимувати (відновити) значення функції:

$y_i = \sin(x_i) + \eta_i, \ \ i = 1,...,l$

де $\{\eta_i\}$ - гаусівські СВ з нульовим середнім та дисперсією 0,1.

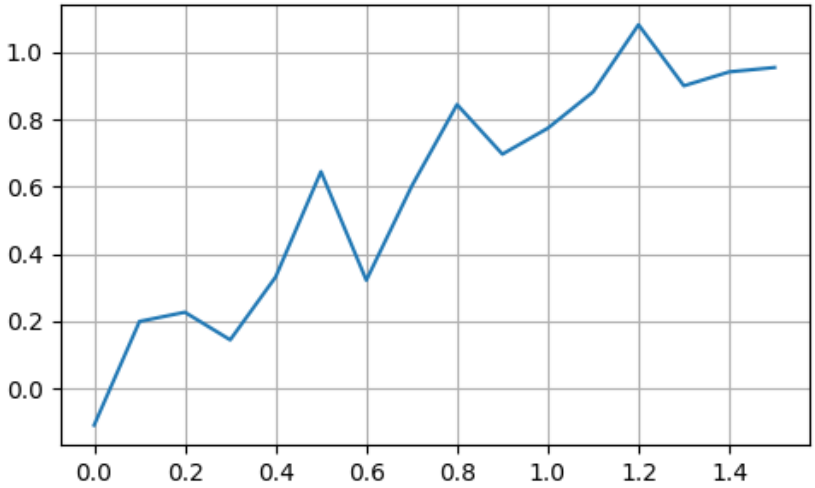


Рис. 11.4

Як базові алгоритми виступатимуть вирішальні дерева заданої максимальної глибини.

Отримуємо наступні результати відновлення вихідного сигналу:

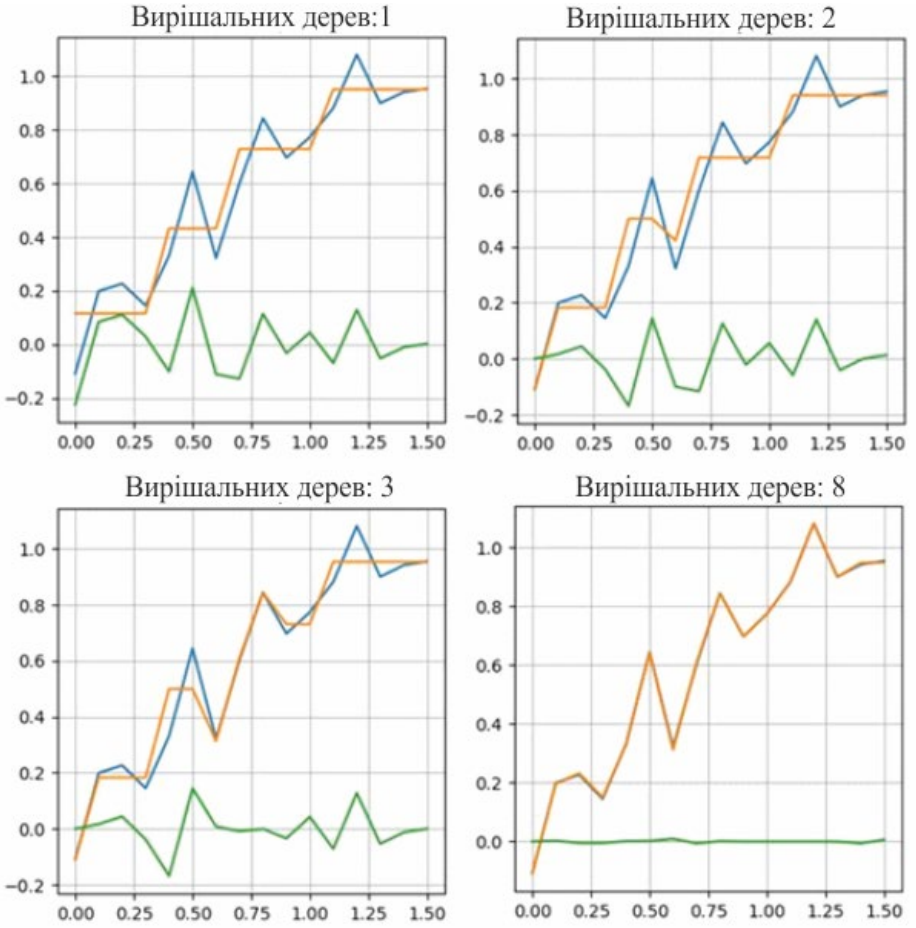


Рис. 11.5

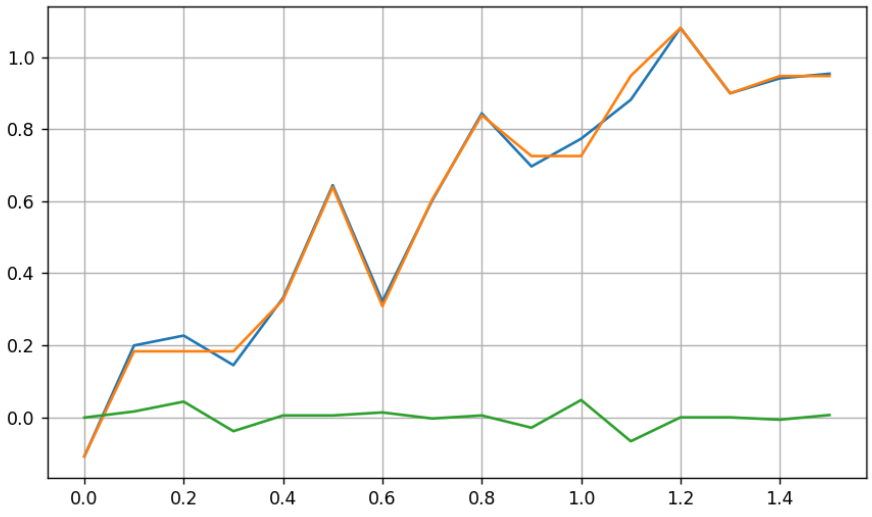


Рис. 11.6

Як бачите, зі збільшенням числа базових алгоритмів зростає точність опису вихідних цільових значень $\{y_i\}$. І за восьми вирішальних дерев ми майже точно описали цільові змінні.

Основні рекомендації. Ми з вами розглянули класичний алгоритм бустінгу AdaBoost для завдань класифікації та регресії.

При цьому в задачах класифікації використовується експоненційна функція втрат $L(x)$ =exp(-M), а в завданнях регресії – квадратична $L(x) = (y - a(x))^2$.

У результаті ці функції призводять до дещо різних алгоритмів бустінгу.

Далі, у бустінгу найчастіше використовуються вирішальні дерева як базові алгоритми, або навіть вирішальні пні, тобто дерево з однією кореневою вершиною і двома листами:

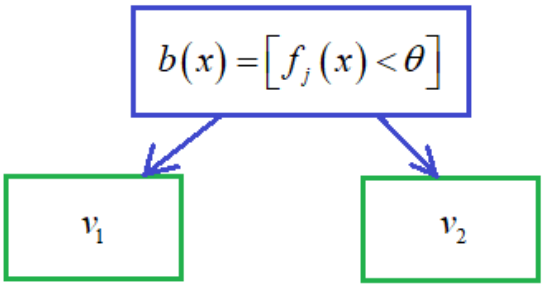


Рис. 9.6

При цьому рекомендується вибирати сімейство простих алгоритмів, які не мають хороших узагальнюючих властивостей. Наприклад, якщо взяти той же *SVM* і спробувати над ним реалізувати бустінг, особливого ефекту не буде. Алгоритми, які власними силами добре вирішують завдання класифікації, регресії чи ранжирування об'єднувати в композицію немає сенсу. Як показала практика, покращень особливих досягти не виходить. А ось дерева при об'єднанні дають нову якість результуючого алгоритму і в ряді випадків оминають той самий *SVM*, нейронні мережі та інші ефективні методи.

Наступний момент, на який слід звернути увагу, – наявність шумових викидів в навчальній вибірці. За таких даних класичний бустінг починає працювати значно гірше. Однак, з цим можна боротися, якщо в алгоритмі класифікації AdaBoost відсіювати об'єкти з більшими значеннями ваги. Я нагадаю, що вага об'єкта поступово зростає, якщо він погано обробляється композицією. І, найімовірніше, є шумовим. Тому при подальшому навчанні наступних алгоритмів такі об'єкти рекомендується просто відкидати.

Недоліки алгоритму AdaBoost

- надмірна чутливість до викидів.
- AdaBoost будує «чорні ящики» – алгоритми, що важко інтерпретуються, т.к. зрозуміти вихідне значення композиції, наприклад дерев, стає дуже складно.
- не вдається будувати короткі композиції (для малих T) із «сильних» алгоритмів (наприклад, *SVM* та нейронні мережі).

- необхідна велика навчальна вибірка (бегінг може обходитися короткішими).

Я думаю, що тепер ви загалом уявляєте, як працює класичний алгоритм AdaBoost. На наступному занятті ми продовжимо цю тему і побачимо, як можна узагальнити цей підхід довільних функцій втрат, використовуючи градієнтні алгоритми.

9.3. Питання для самоконтролю

Питання самоконтролю сформовані на основі викладеного матеріалу і надаються для ознайомлення. Для зручності питання поділені за відповідними пунктами лекції та послідовністю викладення.

1. Які евристики використовуються в бустінгу (boosting).
2. Принципи створення алгоритму AdaBoost.
3. Використання алгоритму AdaBoost в задачах регресії.
4. Використання алгоритму AdaBoost в задачах класифікації.
5. Переваги та недоліки алгоритму AdaBoost.

9.4. Рекомендована література

1. Kevin P. Murphy «Machine Learning: A Probabilistic Perspective», 2012.

2. Murphy, K. P. (2012). Machine Learning: a Probabilistic Perspective. MIT Press, Cambridge, MA, USA

3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.

4. Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale FPGA-based convolutional networks. In R. Bekkerman, M. Bilenko, and J. Langford, editors, Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press.

Оцінювання результатів навчання за дисципліною

Зміст засобів діагностики спрямовано на контроль рівня сформованості знань, умінь/навичок, комунікації, відповідальності і автономії студента за вимогами НРК до 7-го кваліфікаційного рівня (магістрів) під час демонстрації регламентованих робочою програмою результатів навчання.

Студент на контрольних заходах має виконувати завдання, орієнтовані виключно на демонстрацію дисциплінарних результатів навчання. Засоби діагностики, що надаються студентам на контрольних заходах у вигляді завдань для поточного та підсумкового контролю, формуються шляхом конкретизації вихідних даних та способу демонстрації дисциплінарних результатів навчання.

Засоби діагностики (контрольні завдання) для поточного та підсумкового контролю дисципліни затверджуються кафедрою. Види засобів діагностики та процедур оцінювання для поточного та підсумкового контролю дисципліни подано нижче.

<table>
<tr><td colspan="3">Поточний контроль</td><td colspan="2">Підсумковий контроль</td></tr>
<tr><td>навчальне заняття</td><td>засоби діагностики</td><td>процедури</td><td>засоби діагностики</td><td>процедури</td></tr>
<tr><td>лекції</td><td>контрольні завдання за кожною темою</td><td>виконання завдання під час лекцій</td><td rowspan="3">комплексна контрольна робота (ККР)</td><td rowspan="3">визначення середньозваженого результату поточних контролів;

виконання ККР під час заліку за бажанням студента</td></tr>
<tr><td rowspan="2">практичні</td><td>контрольні завдання за кожною темою</td><td>виконання завдань під час практичних занять</td></tr>
<tr><td>індивідуальне завдання</td><td>виконання завдань під час самостійної роботи</td></tr>
</table>

Під час поточного контролю лекційні заняття оцінюються шляхом визначення якості виконання контрольних конкретизованих завдань. Практичні заняття оцінюються якістю виконання практичних робіт та індивідуального завдання.

Якщо зміст певного виду занять підпорядковано декільком описам кваліфікаційних рівнів, то інтегральне значення оцінки може визначатися з урахуванням вагових коефіцієнтів, що встановлюються викладачем.

За наявності рівня результатів поточних контролів з усіх видів навчальних занять не менше 60 балів, підсумковий контроль здійснюється без участі студента шляхом визначення середньозваженого значення поточних оцінок.

Незалежно від результатів поточного контролю кожен студент під час заліку має право виконувати ККР, яка містить завдання, що охоплюють ключові дисциплінарні результати навчання.

Кількість конкретизованих завдань ККР повинна відповідати відведеному часу на виконання. Кількість варіантів ККР має забезпечити індивідуалізацію завдання.

Значення оцінки за виконання ККР визначається середньою оцінкою складових (конкретизованих завдань) і є остаточним.

Інтегральне значення оцінки виконання ККР може визначатися з урахуванням вагових коефіцієнтів, що встановлюється кафедрою для кожної складової опису кваліфікаційного рівня НРК.

Сертифікація досягнень студентів здійснюється за допомогою прозорих процедур, що ґрунтуються на об'єктивних критеріях відповідно до Положення університету «Про оцінювання результатів навчання здобувачів вищої освіти». Досягнутий рівень компетентностей відносно очікуваних, що ідентифікований під час контрольних заходів, відображає реальний результат навчання студента за дисципліною.

Оцінювання навчальних досягнень здійснюється за рейтинговою (100-бальною) та інституційною шкалами. Остання необхідна (за офіційною відсутністю національної шкали) для конвертації (переведення) оцінок мобільних студентів.

Навчальна дисципліна зараховується, якщо студент отримав підсумкову оцінку не менше 60-ти балів. Нижча оцінка вважається академічною заборгованістю, що підлягає ліквідації відповідно до Положення про організацію освітнього процесу НТУ «ДП».

Додаток 1.

Гаусівський байєсівський класифікатор

1. Вхідні дані.

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1)
# вихідні параметри розподілу двох класів
r1 = 0.8
D1 = 1.0
mean1 = [0, -3]
V1 = [[D1, D1 * r1], [D1 * r1, D1]]
r2 = 0.7
D2 = 2.0
mean2 = [0, 3]
 V2 = [[D2, D2 * r2], [D2 * r2, D2]]
```

2. Основні розрахунки.

```
# моделювання навчальної вибірки
N = 1000
x1 = np.random.multivariate_normal(mean1, V1, N).T
x2 = np.random.multivariate_normal(mean2, V2, N).T
# обчислення оцінок МО та коваріаційних матриць
mm1 = np.mean(x1.T, axis=0)
mm2 = np.mean(x2.T, axis=0)
a = (x1.T - mm1).T
VV1 = np.array([[np.dot(a[0], a[0]) / N, np.dot(a[0], a[1]) / N],
                [np.dot(a[1], a[0]) / N, np.dot(a[1], a[1]) / N]])

a = (x2.T - mm2).T
VV2 = np.array([[np.dot(a[0], a[0]) / N, np.dot(a[0], a[1]) / N],
                [np.dot(a[1], a[0]) / N, np.dot(a[1], a[1]) / N]])

# модель гаусівського байєсівського класифікатора
Py1, L1 = 0.5, 1  # ймовірність появи класів
Py2, L2 = 1 - Py1, 1  # та величини штрафів неправильної класифікації

b = lambda x, v, m, l, py: np.log(l * py) - 0.5 * (x - m) @ np.linalg.inv(v) @ (x
- m).T - 0.5 * np.log(np.linalg.det(v))

x = np.array([0, -4])  # вхідний вектор у форматі (x, y)
a = np.argmax([b(x, VV1, mm1, L1, Py1), b(x, VV2, mm2, L2, Py2)])
```

3. Виведення результатів

```
print(a)

plt.figure(figsize=(4, 4)) # виведення графіків
plt.title(f"Кореляції: r1 = {r1}, r2 = {r2}")
plt.scatter(x1[0], x1[1], s=10)
plt.scatter(x2[0], x2[1], s=10)
plt.show()
```

Додаток 2.

Алгоритм кластеризації Ллойда (*K*-середніх).

1. Вхідні дані.

```
import time
import numpy as np
import matplotlib.pyplot as plt
import openpyxl

book = openpyxl.open("ML, lab4.xlsx", read_only=True)
sheet = book.active
# номер першого варіанту (1, 2), другого (2, 3) і т.д
for row in range(1, 2):
  x = [(sheet[row][0].value, …, …, sheet[row][89].value)]
```

2. Основні розрахунки

```
M = np.mean(x, axis=0)    # обчислення середніх за кожною координатою
D = np.var(x, axis=0)        # обчислення дисперсії за кожною координатою
K = 3                                 # число кластерів
ma = [np.random.normal(M, np.sqrt(D / 10), 2) for n in range(K)]    #
початкові центри кластерів
# ma = [np.array(x[0]), np.array(x[1])]
ro = lambda x_vect, m_vect: np.mean((x_vect - m_vect) ** 2)       #
евклідова метрика
COLORS = ('green', 'blue', 'brown', 'black')     # кольорів має бути не менше
кластерів (>= K)
plt.ion()
n = 0
while n < 10:
    X = [[] for i in range(K)]      # ініціалізація порожнього двовимірного
списку для зберігання об'єктів кластерів
    for x_vect in x:
        r = [ro(x_vect, m) for m in ma]        # обчислення відстаней для
поточного образу до центрів кластерів
        X[np.argmin(r)].append(x_vect)       # додавання образу до кластера з
найближчим центром
    ma = [np.mean(xx, axis=0) for xx in X]    # перерахунок центрів
кластерів
```

3. Виведення результатів

```
    plt.clf()
```

```
    # відображення знайдених кластерів
    for i in range(K):
        xx = np.array(X[i]).T
        plt.scatter(xx[0], xx[1], s=10, color=COLORS[i])
    # відображення центрів кластерів
    mx = [m[0] for m in ma]
    my = [m[1] for m in ma]
    plt.scatter(mx, my, s=50, color='red')
    plt.draw()
    plt.gcf().canvas.flush_events()
#    plt.savefig(f"lloyd {n+1}.png")
    time.sleep(0.2)
    n += 1
plt.ioff()
# відображення знайдених кластерів
for i in range(K):
    xx = np.array(X[i]).T
    plt.scatter(xx[0], xx[1], s=10, color=COLORS[i])
# відображення центрів кластерів
mx = [m[0] for m in ma]
my = [m[1] for m in ma]
plt.scatter(mx, my, s=50, color='red')
plt.show()
```

Додаток 3.

Реалізація алгоритму AdaBoost на Python з використанням вирішальних дерев як базових алгоритмів.

1. Вхідні дані.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
import openpyxl

def get_grid(data):
    x_min, x_max = data[:, 0].min() - 10, data[:, 0].max() + 10
    y_min, y_max = data[:, 1].min() - 10, data[:, 1].max() + 10
    return np.meshgrid(np.arange(x_min, x_max, 1), np.arange(y_min, y_max,
1))

book = openpyxl.open("ML, lab6.xlsx", read_only=True)
sheet = book.active
# номер першого варіанту (1, 2), другого (2, 3) і т.д
for row in range(1, 2):
  t = [(sheet[row][0].value, …, …, sheet[row][89].value)]
n1 = len(t[0])
n2 = len(t[1])
```

2. Основні розрахунки

```
train_data = np.r_[t[0], t[1]]
train_labels = np.r_[np.ones(n1) * -1, np.ones(n2)]

# x, y = train_data[:, 0], train_data[:, 1]
# plt.scatter(x[train_labels == -1], y[train_labels == -1])
# plt.scatter(x[train_labels == 1], y[train_labels == 1])
# plt.show()

XN = len(train_data)    # довжина навчальної вибірки
T = 1                   # число алгоритмів у композиції
max_depth = 2           # максимальна глибина вирішальних дерев
w = np.ones(XN) / XN    # початкові значення ваги для об'єктів вибірки
algs = []               # список з отриманих алгоритмів
alfa = []               # список з обчислених ваг для композиції
```

```
  for n in range(T):
      # створюємо та навчаємо вирішальне дерево з вагами об'єктів w
      algs.append(DecisionTreeClassifier(criterion='gini',
max_depth=max_depth))
      algs[n].fit(train_data, train_labels, sample_weight=w)

      predicted = algs[n].predict(train_data) # формуємо прогнози отриманого
дерева за навчальною вибіркою
      N = np.sum(np.abs(train_labels - predicted) / 2) / XN   # обчислюємо
частку невірних класифікацій
      alfa.append( 0.5 * np.log((1 - N) / N) if N != 0 else np.log((1-1e-8) / 1e-8) )
# обчислюємо вагу для поточного алгоритму
      # перераховуємо ваги об'єктів вибірки
      w = w * np.exp(-1 * alfa[n] * train_labels * predicted)
      w = w / np.sum(w)
  # обчислюємо число помилок класифікації на основі отриманої
композиції
  predicted = alfa[0] * algs[0].predict(train_data)
  for n in range(1, T):
      predicted += alfa[n] * algs[n].predict(train_data)

  N = np.sum(np.abs(train_labels - np.sign(predicted)) / 2)
```

3. Виведення результатів

```
      print(f"Кількість помилок на навчальній вибірці: {N} при композиції
{T} вирішальних дерев")
  # Відображаємо отримані результати класифікації
  xx, yy = get_grid(train_data)
  predicted = alfa[0] * algs[0].predict(np.c_[xx.ravel(),
yy.ravel()]).reshape(xx.shape)
  for n in range(1, T):
      predicted += alfa[n] * algs[n].predict(np.c_[xx.ravel(),
yy.ravel()]).reshape(xx.shape)
  plt.pcolormesh(xx, yy, predicted, cmap='spring', shading='auto')
  plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, s=5000 * w,
cmap='spring', edgecolors='black', linewidth=1.5)
  plt.show()
```

Навчальне видання

Желдак Тимур Анатолійович
Владико Олександр Борисович

МАШИННЕ НАВЧАННЯ

Конспект лекцій
для здобувачів ступеня магістра
зі спеціальності 124 Системний аналіз

Видано в авторській редакції.

Електронний ресурс.
Підписано до видання 24.06.2024. Авт. арк. 11,2.

Національний технічний університет «Дніпровська політехніка».
49005, м. Дніпро, просп. Дмитра Яворницького, 19.

157/157