Міністерство освіти і науки України
Сумський державний університет

В. О. Харченко

ОСНОВИ МАШИННОГО НАВЧАННЯ

Навчальний посібник

Рекомендовано вченою радою Сумського державного університету

Суми
Сумський державний університет
2023

УДК 004.032.26:001.103(075.8)(0.034)
Х 22

Рецензенти:

О. В. Лисенко – доктор фізико-математичних наук, професор, професор кафедри прикладної математики та моделювання складних систем Сумського державного університету;

Н. О. Красношлик – кандидат технічних наук, доцент, доцент кафедри прикладної математики та інформатики Навчально-наукового інституту інформаційних та освітніх технологій Черкаського національного університету ім. Богдана Хмельницького

Рекомендовано до видання
вченою радою Сумського державного університету
як навчальний посібник
(протокол № 15 від 29 червня 2023 року)

Харченко В. О.

Х 22 Основи машинного навчання : навч. посіб. / В. О. Харченко. – Суми : Сумський державний університет, 2023. – 264 с.

У навчальному посібнику наведені теоретичні основи, методи та алгоритми машинного навчання, приклади їх застосування й завдання для виконання практичних робіт із типовими прикладами подання результатів. Видання укладене в співробітництві із Сумською IT-компанією CPCS.

Рекомендований для студентів закладів вищої освіти спеціальностей «Прикладна математика» та «Комп'ютерні науки».

УДК 004.032.26:001.103(075.8)(0.034)

Зміст

ПЕРЕДМОВА

Цей навчальний посібник містить основні математичні концепції класичного машинного навчання, детальний огляд основних видів завдань машинного навчання та методів їх розв'язання, а також приклади задач із їх розв'язуванням. Окремо для кожного методу наведено поставлення задачі для практичної роботи і приклад поданняі результатів реалізації кожного алгоритму.

Для полегшення сприйняття матеріалу, викладеного в цьому навчальному посібнику вважається, що читач ознайомлений з основами лінійної алгебри, зокрема, з аспектами роботи з векторами та матрицями; основами математичного аналізу, статистичного аналізу та методами оптимізації. Для виконання практичних робіт слухачі повинні мати навички програмування, зокрема мовою Python.

У першому розділі викладені основні концепції машинного навчання, основні поняття, типи задач машинного навчання та наведені приклади прикладних задач. У другому розділі наведено математичну формалізацію проблеми машинного навчання. У розділах 3 та 4 обговорюються задачі машинного навчання з учителем: регресія та класифікація. Для задач класифікації розглянуто чотири основних методів: дерева прийняття рішень, k-найближчих сусідів, метод опорних векторів та метод логістичної регресії. Розділи 5–7 містять огляд методів навчання без учителя. Зокрема, у розділі 5 проведено детальний розбір методів зменшення розмірності даних та виділення головних компонент. У розділі 6 обговорюються задачі кластеризації. Тут проведено огляд основних методів k-середніх та DBSCAN. Розділ 7 стосується задачі пошуку асоціативних, де розглянуто два основні алгоритми, Apriori та FP-дерева. У розділі 8 наведено огляд ансамблевих методів навчання, де зосереджено увагу на двох типах ансамблів, а саме, беггенгу та бустінгу. Останній розділ стосується огляду методів навчання з підкріпленням.

Автор висловлює вдячність Олександру Алфімову – директору IT компанії «CPCS» та backend-розробнику компанії «CPCS» Ігорю Князю за консультації та допомогу в структуруванні цього навчального посібника.

Розділ 1

Вступ до машинного навчання

Машинне навчання (Machine Learning, ML) – розділ штучного інтелекту, присвячений розумінню та розробленню методів, що дозволяють машинам «навчатися», тобто методів, у яких використовують дані для покращання продуктивності комп'ютера в певному наборі завдань. Алгоритми машинного навчання створюють модель на основі вибіркових даних, відомих як навчальні дані, щоб робити прогнози чи ухвалювати рішення без явного програмування для цього. Алгоритми машинного навчання використовують у широкому спектрі завдань, таких як медицина, фільтрація електронної пошти, розпізнавання мовлення, сільське господарство, виявлення шахрайства, виявлення загроз зловмисного програмного забезпечення, автоматизація бізнес-процесів, комп'ютерний зір, проєктування безпілотних апаратів тощо, де важко або неможливо розробити звичайні алгоритми для виконання необхідних завдань. Машинне навчання, яке застосовують у бізнес-проблемах, також називають прогнозною аналітикою.

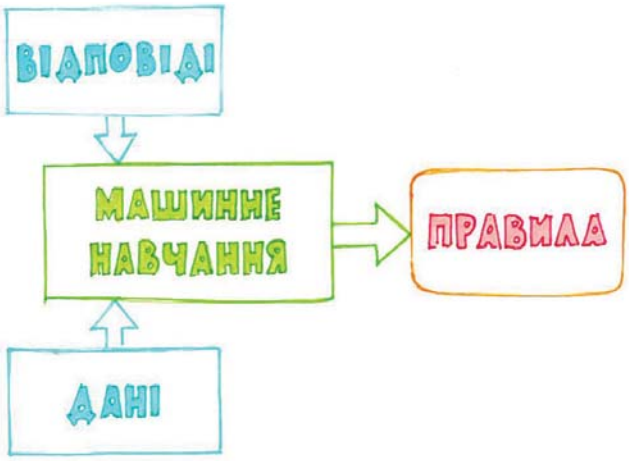
Машинне навчання дає компаніям уявлення про тенденції поведінки клієнтів і бізнес-операційні моделі, а також підтримує розроблення нових продуктів. Багато провідних сучасних компаній, таких як Facebook, Google і Uber, роблять машинне навчання центральною частиною своєї діяльності. Машинне навчання стало значним конкурентоспроможним фактором для багатьох компаній. Машинне навчання тісно пов'язане з обчислювальною статистикою, зосереджуваною на прогнозуванні за допомогою комп'ютерів, але не все машинне навчання є статистичним навчанням. Вивчення математичної оптимізації надає методи, теорію та сфери застосування в галузі машинного навчання. Інтелектуальний аналіз даних є спорідненою галуззю дослідження, що базується на дослідницькому аналізуванні даних за допомогою навчання.

1.1 Основні поняття

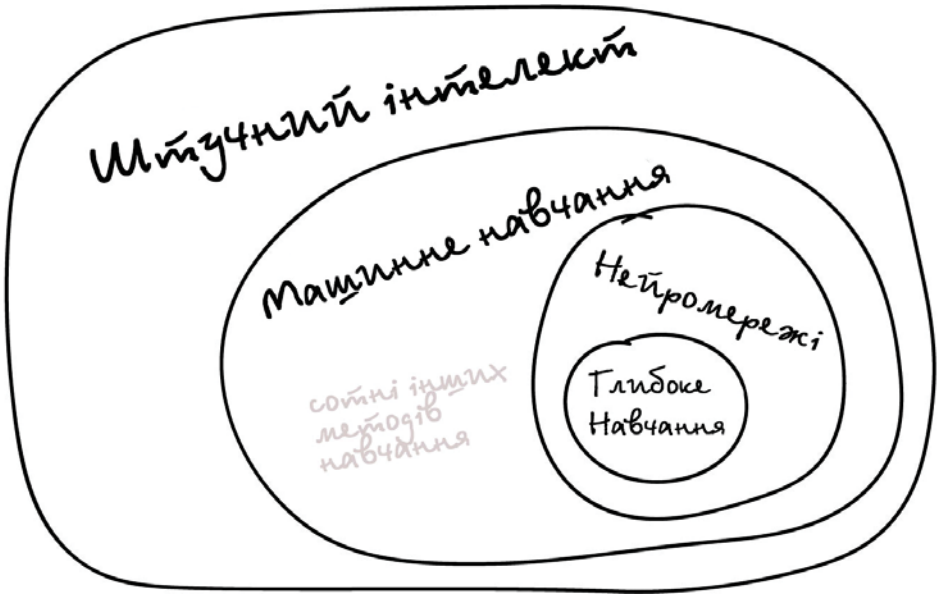
Перш ніж говорити про навчання, почнемо з аналізу термінології. **Data Science (наука про дані)** – це загальне найменування дисциплін із вивчення даних, **Machine Learning (машинне навчання)** – це підрозділ **Data Science**,що займається розбудовою розумних моделей. Такі моделі можна використовувати для передбачення купівлі товару користувачем, рекомендацій у соцмережах (рекомендаційні системи), розпізнавання зображень тощо. **Data Science**-спеціалісти займаються дослідженнями. В іноземних компаніях такій посаді відповідають позиції research-інженерів – це переважно математики, які працюють із теоретичною частиною алгоритмів та досліджують різноманітні закономірності. **Machine Learning**-інженери, у свою чергу, займаються побудовою моделей з урахуванням одержаних даних. Але в деяких країнах такий поділ існує лише теоретично. В Україні **Data Science** та **Machine Learning** раніше використовували як слова-синоніми, та ці поняття вже починають розділяти. У наших реаліях вакансії, де необхідне знання Machine Learning, найчастіше називають Data Scientist і навпаки. Тому, для успішної роботи з даними, необхідно освоїти й те, й інше.

Інтелектуальні системи без здатності до навчання ухвалюють рішення на основі даних, які до них надходять, і правил, написаних людиною-програмістом. Наприклад, калькулятор може додати 2 до 2, тому що хтось запрограмував для нього правила додавання. Для цього комп'ютеру не довелося вивчати тисячі прикладів додавання, програміст просто знав, як це працює, і переклав мовою, зрозумілою комп'ютеру. Це прекрасно працює, коли сам програміст знає правила, які хоче запрограмувати.

Проте, в реальному житті це не завжди так. Програміст не завжди знає, за якими саме правилами повинна працювати програма. І тут на допомогу приходять системи з машинним навчанням на основі прикладів. Основна їх властивість – це здатність знаходити правила, використовуючи наявні приклади.



Машинне навчання (МН) – це розділ штучного інтелекту.



Штучний інтелект – назва всієї галузі знань, наприклад біологія, фізика чи хімія.

Машинне навчання – розділ штучного інтелекту.

Нейронні мережі (нейромережі) – один із видів машинного навчання. Популярний, проте існують й інші.

Глибоке (глибинне) навчання – архітектура нейромереж, один з підходів до їх побудови та навчання.

1.1.1 Три складові навчання

Мета машинного навчання – передбачити результат за вхідними даними. Чим різноманітніші вхідні дані, тим простіше машині знайти закономірності й тим точніший результат. Отже, якщо ми хочемо навчити машину, нам потрібні три речі: дані, ознаки, алгоритми.

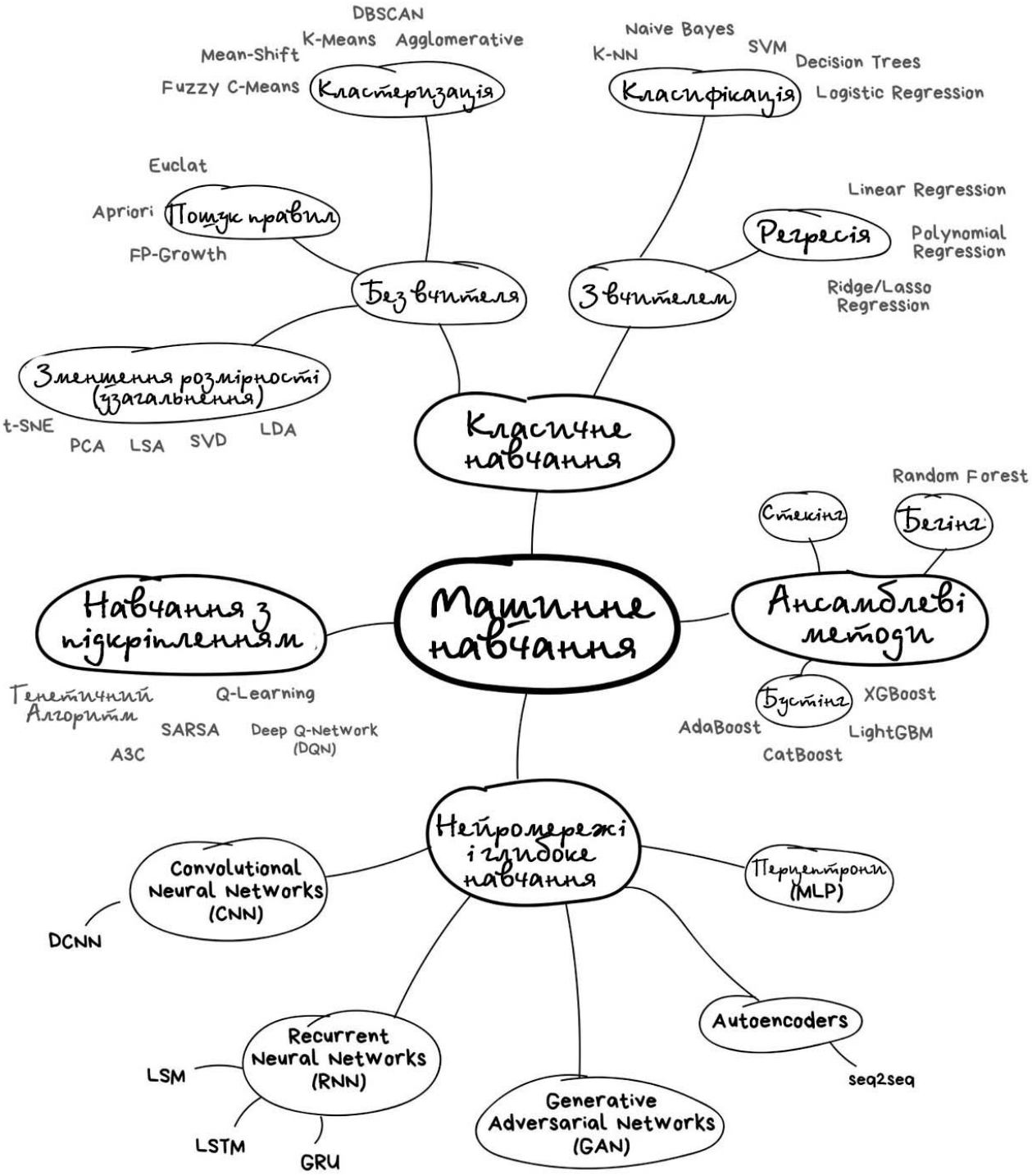
Дані. Хочемо виявляти спам – потрібні приклади спам-листів; передбачати курси акцій – потрібна історія цін; дізнатися інтереси користувача – потрібні його «лайки» або «пости». Даних потрібно якомога більше. Їх збирають по різному: вручну – процес тривалі-ший, даних менше, зате без помилок; автоматично – просто «зливають» машині все, що знайшлося. Наприклад, google використовує своїх користувачів для безкоштовної розмітки: ReCaptcha, яка просить «знайти на фотографії всі дорожні знаки», – це воно і є.

Ознаки (features) – властивості, характеристики. Ними можуть бути пробіг автомобіля, стать користувача, ціна акцій, навіть лічильник частоти появи слова у тексті. Машина повинна знати, на що їй конкретно дивитися. Добре, коли дані просто лежать в таблицях – назви їх колонок і є ознаками. Коли ознак багато, модель працює повільно й неефективно. Найчастіше відбір правильних (принципових, основних) ознак займає більше часу, ніж усе інше навчання. Але бувають і зворотні ситуації, коли користувач сам вирішує відібрати лише «правильні» на його погляд, ознаки і вносить у модель суб'єктивність – модель починає помилятися.

Алгоритми. Зазвичай одну й ту задачу майже завжди можна розв'язати різними методами (способами). Від вибору методу залежать точність, швидкість роботи і розмір готової моделі. Але є один нюанс: якщо дані «брудні» (нечіткі, неточні, з пропусками), то навіть найкращий алгоритм не допоможе. У такому разі рекомендують зібрати як можна більше даних.

1.1.2 Карта машинного навчання

Загалом карту світу машинного навчання можна подати таким способом:



Виділяють чотири основні типи машинного навчання.

Класичне навчання поділяють на навчання «з учителем» та навчання «без учителя».

Навчання з учителем (Supervised Learning)

У цьому разі машина має «наставника» – учителя, який говорить їй, як правильно вчинити. Вчитель заздалегідь відмічає всі потрібні дані, щоб машина навчалася на конкретних прикладах. Так він показує, що на цій світлині автомобіль, на іншій – велосипед.

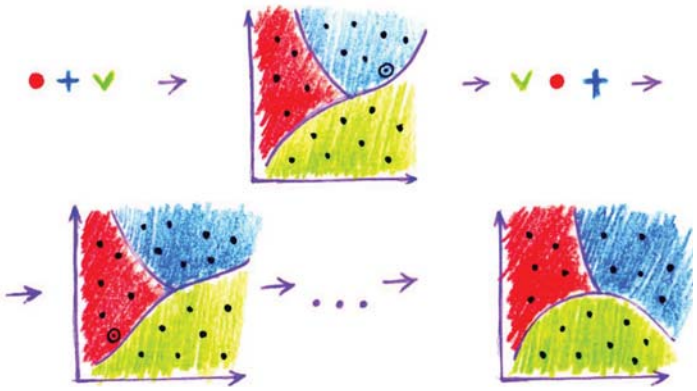
Учителем не завжди буває програміст, який стоїть над машиною й контролює кожну її дію. У термінах машинного навчання вчитель – це саме втручання людини в процес оброблення інформації. З учителем машина навчається набагато краще й швидше, тому для виконання практичних завдань такі алгоритми використовують частіше.

Навчання без учителя (Unsupervised Learning)

У навчанні без учителя машині просто вивалюють купу фотографій на стіл і кажуть: «розберися, що тут до чого». Дані в такому разі не розмічені, і машина сама намагається знайти закономірності. На практиці такі алгоритми використовують рідше, зазвичай як методи аналізування та підготовки даних, а не як основний алгоритм, що виконує конкретні завдання за допомогою цих даних. У разі, якщо розмічення даних неможливе, вдаються до методів навчання без учителя.

Ансамблеві методи

Основна ідея – використання паралельно чи послідовно декількох стандартних методів навчання для одержання більш якісного результату роботи алгоритму.

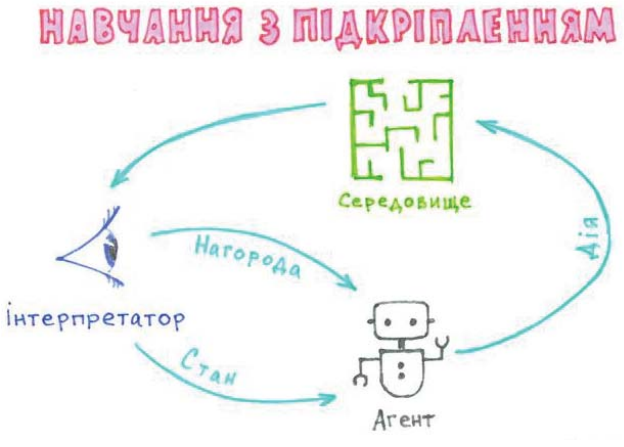


Сьогодні використовують для:

- всього, де можна скористатися класичними алгоритмами;
- пошукові системи;
- розпізнавання об'єктів.

Навчання з підкріпленням (Reinforcement Learning)

Навчання з підкріпленням менше подібне до попередніх видів, оскільки нагадує швидше той штучний інтелект, яким нас намагаються вразити у фантастичних фільмах. Такі алгоритми використовують не там, де потрібно проаналізувати дані, а там, де потрібно вижити в реальному середовищі.



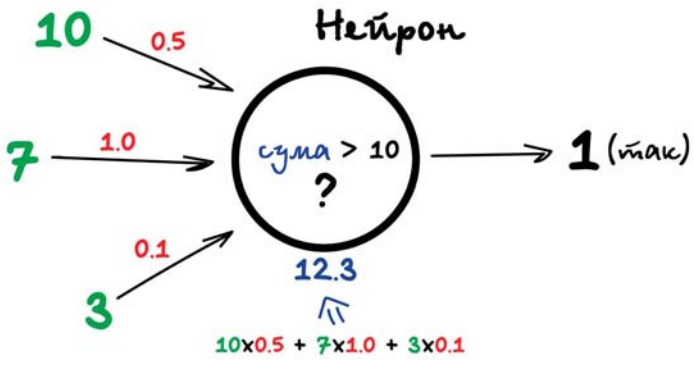
Середовищем може бути що завгодно: як реальний світ, так і симуляція, і навіть комп'ютерна гра. Наприклад, існують роботи, які навчилися грати в різні ігри, просто поринувши в середовище, або автопілот Tesla, який у симуляції вчиться не збивати пішоходів.

Знання про довкілля таким роботам корисні, але знати весь світ їм необов'язково. Завдання таких машин – не розрахувати всі ходи, а мінімізувати помилки або максимізувати вигоду. Навчання з підкріпленням дуже подібне до реального навчання людей –

машину карають за помилки і заохочують за правильні вчинки. Наприклад, кількість комбінацій у грі Го перевищує кількість атомів у Всесвіті, тому всі комбінації машині запам'ятати неможливо. Алгоритм AlphaGo просто обирає найкращий вихід із ситуації, що склалася, і робить це краще за людину. Успіхи машин у комп'ютерних іграх, спорті або технології автопілота мають досить ефективний вигляд. Та насправді алгоритмів навчання з підкріпленням не так багато, цей напрямок лише починає розвиватися. Саме тому за ними, безсумнівно, майбутнє.

Нейронні мережі

Будь-яка нейромережа – це набір нейронів і зв'язків між ними. Нейрон найкраще уявляти собі просто як функцію з великою кількістю входів і одним виходом. Завдання нейрона – взяти цифри зі своїх входів, виконати над ними функцію і віддати результат на вихід. Простий приклад корисного нейрона: знайти суму всіх цифр із входів і якщо їх сума більша за N видати на вихід одиницю, інакше – нуль.



Зв'язки – це канали, через які нейрони надсилають один одному цифри. Кожний зв'язок має свою вагу – її єдиний параметр, який можна умовно уявити як міцність зв'язку. Якщо через зв'язок із вагою 0,5 проходить число 10, воно перетворюється на 5. Сам нейрон не розбирається, що до нього надійшло й підсумовує все. Ваги потрібні, щоб керувати, на які входи нейрон повинен реагувати, а на які – ні.

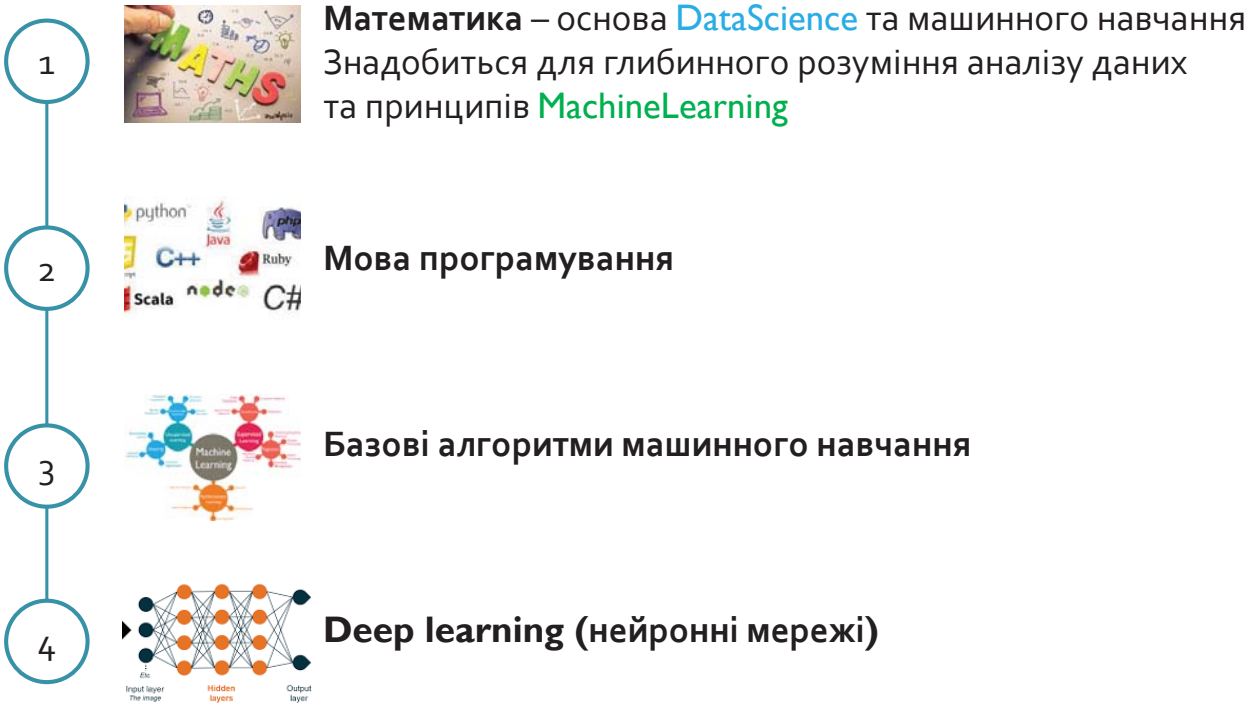
Щоб мережа не перетворилася на анархію, нейрони вирішили пов'язувати не як завгодно, а шарами. Всередині одного шару нейрони ніяк не пов'язані, але з'єднані з нейронами попереднього й наступного шарів. Дані в такій мережі розміщені точно в одному напрямку – від входів першого шару до виходів останнього.

Перенесення навчання (Transfer learning)

. Навчаємо модель для однієї проблеми й використовуємо результат навчання для модифікації даних від іншої. Наприклад, фільтр, за допомогою якого ви робите селфі подібними до творінь Вінсента ван Гога та який домальовує заячі вуха у відеоконференції або змушує Обаму вимовляти ваші слова його ж голосом і з його мімікою.

1.1.3 Освоєння машинного навчання

Процес вивчення Data Science та Machine Learning можна поділити на чотири блоки:



Математика. Для початку давайте розберемося, чи потрібна математика в роботі з Data Science і Machine Learning. Короткою відповіддю буде: так, потрібна. Безумовно, є багато прикладів

того, як успішні Data Scientists займають призові місця на Kaggle-змаганнях, не маючи технічної освіти. Але навіть вони погодяться, що знання математики дає значну перевагу в роботі з Data Science. Незважаючи на те, що майже всі алгоритми реалізуються в бібліотеках **Python**, розуміння базових математичних концепцій значно спростить ваше навчання та виконання прикладних завдань. Крім того, в більшості статей про машинне навчання містяться математичні викладки, читати які без знань математики буде важко.

Для успішної роботи мінімально потрібно розуміти чотири розділи математики:

- основи лінійної алгебри;
- основи математичного аналізу (інтегрування, похідні та частинні похідні);
- методи оптимізації;
- основи теорії ймовірностей та математичної статистики.

Мова програмування. Для роботи з даними ви повинні вміти програмувати. Наприклад, щоб завантажити дані, розпарсити, синтезувати нові ознаки або втілити в життя будь-яку іншу вашу ідею. Основною мовою програмування більшості Data Science фахівців є **Python**, яка сама по собі дуже проста і в ній реалізовано безліч бібліотек для оброблення та аналізування даних.

Базові алгоритми. Для того щоб освоїти принципи машинного навчання, необхідно знати основні наявні класи задач Machine Learning, алгоритми та які підходи дозволяють вирішити той чи інший клас задач. Також необхідно розрізняти алгоритми різних спеціалізацій, розуміти їх переваги та недоліки. Машинне навчання є практичною дисципліною, тому дуже важливо застосовувати здобуті знання на реальних даних. На платформі Kaggle можна знайти безліч датасетів, на яких можна розібрати рішення інших учасників та попрактикувати свої аналітичні навички. І згодом можна спробувати це зробити на якомусь відкритому конкурсі.

Deep learning. Маючи базове розуміння принципів машинного навчання і знання Python, можна розпочинати вивчення Deep

Learning. Це один із розділів машинного навчання, основою якого є використання нейронних мереж.

1.1.4 Основні бібліотеки Python для машинного навчання

Сьогодні існує величезна кількість бібліотек для машинного та глибокого навчання. Щоб полегшити завдання вибору, ми розглянемо лише найпопулярніші та найнеобхідніші бібліотеки, які покривають усі базові потреби для початку роботи з Machine Learning та Deep Learning.

NumPy: багатовимірні масиви та лінійна алгебра

Основний функціонал NumPy полягає в підтриманні багатовимірних масивів даних та швидких алгоритмів лінійної алгебри. Саме тому NumPy – ключовий компонент Scikit-learn, SciPy та Pandas. Зазвичай NumPy використовують як допоміжну бібліотеку для виконання різних математичних операцій із структурами даних Pandas, тому варто вивчити її базові можливості.

Pandas: вилучення та підготовка даних

Аналізування та підготовка даних найчастіше займають більшу частину часу при виконанні завдань машинного навчання. Дані можуть бути одержані в CSV, JSON, Excel або іншому структурованому (або не дуже) форматі, і виникає необхідність обробити їх, для того щоб застосовувати в моделях машинного навчання. Для цього використовують бібліотеку Pandas. Це потужний інструмент, що дозволяє швидко аналізувати, модифікувати та готувати дані для подальшого використання в інших бібліотеках, таких як Scikit-learn, TensorFlow або PyTorch. У Pandas можна завантажувати дані з різних джерел: SQL-баз, CSV-, Excel-, JSON-файлів та інших менш популярних форматів. Коли дані завантажені в пам'ять, із ними можна виконувати безліч різних операцій для аналізування, трансформації, заповнення відсутніх значень та очищення набору

даних. Pandas дозволяє виконувати безліч SQL-подібних операцій над наборами даних: об'єднання, групування, агрегування тощо. Також вона надає вбудований набір популярних статистичних функцій для базового аналізування.

Matplotlib і Seaborn: побудова графіків та візуалізація

Matplotlib – це стандартний інструмент у наборі датаінженера. Він дозволяє створювати різноманітні графіки та діаграми для візуалізації одержаних результатів. Графіки, створені Matplotlib, легко інтегруються в Jupyter Notebook. Це дозволяє візуалізувати дані та результати, одержані під час оброблення моделей. Для цієї бібліотеки створено багато додаткових пакетів.

Один із найбільш популярних – Seaborn. Його основна перевага – готовий набір найчастіше використовуваних статистичних діаграм і графіків.

Scikit-learn – найкраща бібліотека для класичних алгоритмів машинного навчання

Scikit-learn – одна з найпопулярніших бібліотек машинного навчання на сьогодні. Вона підтримує більшість алгоритмів навчання як з учителем, так і без: лінійна та логістична регресія, метод опорних векторів (SVM), Naive Bayes-класифікатор, градієнтний бустинг, кластеризація, KNN, k-середні та багато інших.

Крім того, Scikit-learn містить безліч корисних утиліт для підготовки даних та аналізування результатів. Ця бібліотека переважно призначена для класичних алгоритмів машинного навчання, тому її функціонал для нейронних мереж дуже обмежений, а для завдань глибокого навчання її зовсім не можна використовувати.

Tensorflow та Keras – бібліотеки глибокого навчання

Будь-яка бібліотека глибокого навчання містить три ключові компоненти: багатовимірні масиви (вони ж тензори), оператори лінійної алгебри та обчислення похідних. У Tensorflow, бібліотеці глибокого навчання

від Google, добре реалізовані всі три компоненти. Поряд із CPU вона підтримує обчислення на GPU і TPU (тензорних процесорах Google). На цей час це найпопулярніша бібліотека глибокого навчання.

Keras – це надбудова над Tensorflow, за допомогою якої можна вирішувати безліч проблем останньої. Її головна особливість – це можливість будувати архітектуру нейронної мережі з використанням Python DSL.

PyTorch – альтернативна бібліотека глибокого навчання

PyTorch – це друга за популярністю бібліотека глибокого навчання після Tensorflow, створена у Facebook. Її сильний бік полягає в тому, що вона була розроблена для Python і тому використовує його стандартні ідіоми. Порівняно з Tensorflow тут поріг входу набагато нижчий, а будь-яку нейронну мережу можна побудувати з використанням стандартних класів та об'єктів об'єктно-орієнтованого програмування. Також її легше налагоджувати, тому що код виконується як звичайний Python-код -– немає етапу компіляції, як у Tensorflow. Тому можна скористатися навіть Python-відладником. Якщо порівнювати з Keras, PyTorch – багатослівніший, але менш магічний. PyTorch теж має свою надбудову – це бібліотека fastai. Вона дозволяє виконати більшість стандартних завдань глибокого навчання як кілька рядків коду.

1.2 Типи задач машинного навчання

Найпопулярніші задачі машинного навчання:

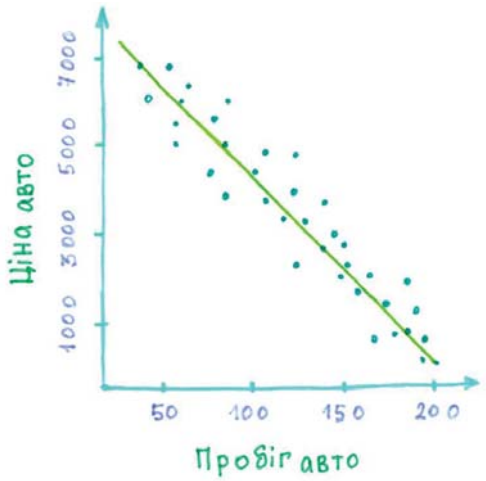
- регресія;
- прогнозування;
- класифікація;
- кластеризація;

- зменшення розмірності;
- виявлення аномалій;
- пошук правил.

1.2.1 Регресія

Регресію дуже полюбляють аналітики та фінансисти, тому що вона може показати залежності між різними факторами процесу. Регресія показує або передбачає взаємозв'язок між процесом та його чинником.

Наприклад, як на ціну будинку впливає той чи інший район розміщення? Або що більше впливає на вартість авто: рік випуску чи пробіг? Машина намагається побудувати криву на графіку, яка відображає залежність. Проте, на відміну від людини з крейдою та дошкою, вона це робить із застосуванням математики.



Моделі регресійного аналізу зазвичай використовують, щоб показати або передбачити взаємозв'язок між процесом і тим, що цей процес може спровокувати. Тут варто пам'ятати, що така кореляція – не завжди причинність, тобто навіть пряма в простій лінійній регресії, яка добре відображає залежності між даними, може не дати конкретної відповіді про причинно-наслідковий зв'язок. Саме тому регресійний аналіз не використовують для інтерпретації причинно-наслідкових зв'язків між змінними. Однак такий аналіз може засвідчити проте, як і наскільки змінні пов'язані одна з одною, а визначення причин та наслідків – це предмет глибших досліджень за допомогою інших алгоритмів і методів.

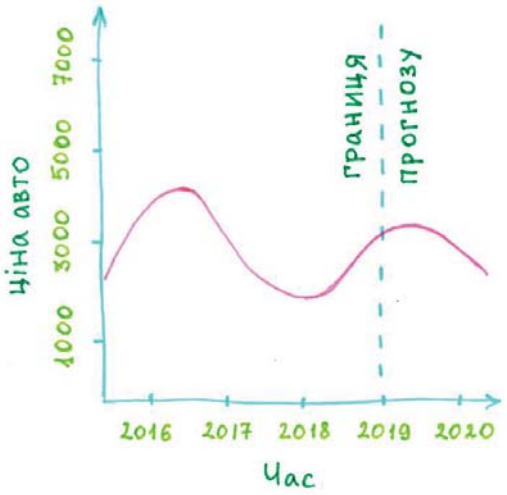
Графік регресії може показати позитивний зв'язок, негативний зв'язок або відсутність зв'язку процесу з тими чи іншими факторами. Якщо лінія регресії є горизонтальною або вертикальною – між змінними ніякого зв'язку немає, тобто якщо повернутися до прикладу з авто та пробігом, де на одній осі буде ціна авто, а на

інший – його пробіг, горизонтальна або вертикальна пряма регресії буде означати, що пробіг ніяк не впливає на вартість авто. Якщо зі зростанням **x** зростає **y** (нижня частина графіка перетинає вісь, а верхня прагне в поле графіка) – залежність позитивна, тобто ціна буде зростати зі збільшенням пробігу, якщо навпаки – негативна, тобто чим більший пробіг, тим менша ціна.

1.2.2 Прогнозування

Сьогодні вигадування сценаріїв майбутнього це не сюжет фантастичної книги, а є завданням для математиків і фахівців із машинного навчання.

Прогностичних моделей існує безліч, і всі вони виконують завдання передбачення часових рядів – знаходження майбутніх значень залежно від часу. Прогнозування використовують у медичній діагностиці, оцінюванні кредитоспроможності, передбаченні попиту, під час ухвалення рішень на фінансових ринках.



Алгоритм під час побудови прогнозу містить такі елементи.

1. Аналізування часової послідовності на предмет наявності пропущених значень і значень, що випадають, та корекція цих значень. Для знаходження пропущених значень і значень, що випадають, також існують окремі алгоритми. Під час збирання реальних даних за деякий час вони можуть зникнути або бути введеними неправильно. Тому їх потрібно визначити експериментально.

2. Визначення наявності тренду і його типу. Визначення періодичності в послідовності.

3. Перевірка послідовності на стаціонарність, тобто того, що процес у майбутньому буде розвиватися так само, як у минулому й сьогоденні. У реальній природі таких процесів не існує, але

процеси, характеристики яких змінюються дуже повільно, можна зарахувати до стаціонарних. Наприклад, кількість хліба, яку українці з'їдають щодня, практично не залежить від погоди та пори року або інших зовнішніх факторів. І хоча ця цифра буде змінюватися, її динаміка буде незначною, тому такий процес можна сміливо назвати стаціонарним.

4. Аналізування послідовності на предмет необхідності попереднього оброблення. Всі дані повинні мати однакові характеристики й не відрізнятися між собою.

5. Вибір моделі залежно від даних (стаціонарні чи ні) та бажаного результату (короткотерміновий чи довготерміновий прогноз тощо).

6. Визначення параметрів моделі. Прогноз на підставі обраної моделі.

7. Оцінювання точності прогнозування моделі. Для цього прогноз зазвичай будують для вже відомих даних і порівнюють їх. Наприклад, у нас є дані про ціни на пальне з 2000 до 2022 року. Для оцінювання точності моделі ми будуємо прогноз цін на 2022 рік на підставі даних із 2000 до 2021 року й порівнюємо одержані дані з реальними.

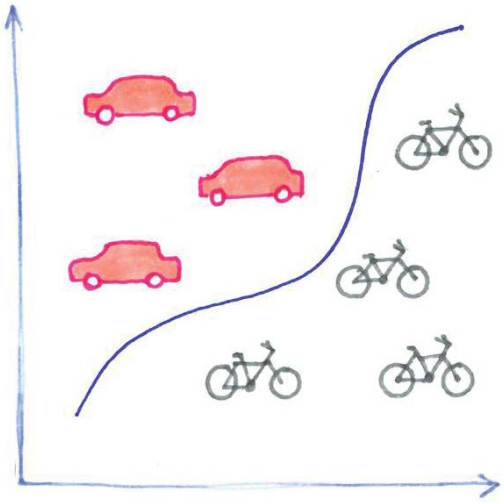
8. Аналіз похибки обраної моделі.

9. Визначення адекватності обраної моделі і, якщо результат виявиться незадовільним (недостовірним чи недостатньо точним), її заміна і повернення до попередніх пунктів.

1.2.3 Класифікація

Алгоритми класифікації дозволяють розділити об'єкти відповідно до зазначених заздалегідь класів.

Найпростішим із технічного погляду завданням класифікації є бінарна класифікація – коли об'єкти потрібно розподілити між двома класами. Наприклад, на вході є транспортні засоби й ми знаємо, що це або автомобілі, або велосипеди. Машина за певними алгоритмами розподіляє кожний із транспортних засобів до одного з визначених класів (авто або велосипед).



У багатокласовій класифікації кількість класів може досягати декількох тисяч, і рішення стає значно складнішим. Також бувають класи, що перетинаються, (у таких випадках об'єкт може одночасно належати до декількох класів), і нечіткі класи (якщо належність до того чи іншого класу визначається ймовірністю).

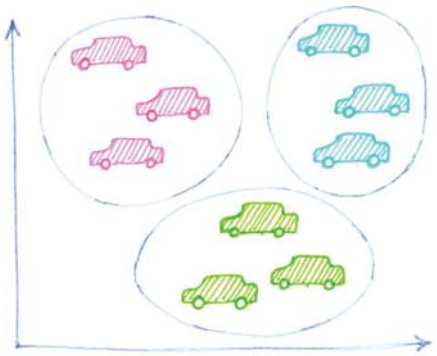
Сьогодні алгоритми класифікації використовують для великої кількості завдань: визначення мови, спам-фільтрів, визначення шахрайства (якщо зловмисник сплачує за послуги вкраденими коштами), пошуку подібних документів тощо.

Щоб класифікація спрацювала, потрібно мати розмічені дані з категоріями та ознаками, які машина буде вчитися визначати. Залежно від певних ознак алгоритм визначає, до якого з класів можна віднести об'єкт.

1.2.4 Кластеризація

Алгоритми кластеризації дозволяють розділити об'єкти в разі якщо класи заздалегідь не зазначені, а кластери повинні бути сформовані за подібністю елементів. Алгоритми кластеризації використовують у завданнях сегментації ринку, для аналізування нових даних, стиснення зображень.

Машина визначає подібні ознаки об'єктів та групує їх у кластери. Кількість кластерів, як із класами, може бути визначена дослідником або самою машиною. Також дослідник може задати ознаки, за якими потрібно розділити вибірку, або машина визначить їх сама.

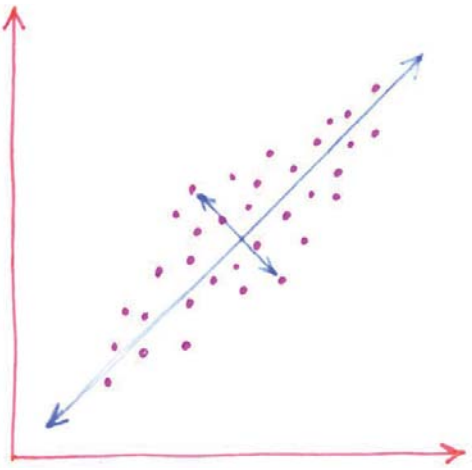


1.2.5 Зменшення розмірності

Мета – зведення великої кількості ознак до меншої для зручності їх подальшого застосування. Використовують для побудови рекомендаційних систем, пошуку подібних документів чи візуалізації.

Завдання – задача, в якій машина з величезної кількості даних виявляє ті, що мають будь-які закономірності.

Методи зменшення розмірності даних дозволяють зібрати конкретні ознаки в абстракції більш високого рівня. Практична користь методів зменшення розмірності в тому, що можна отримати абстракцію, об'єднавши кілька ознак в одну: автомобілі вище ніж 2 метри з двома великими задніми колесами – трактори.



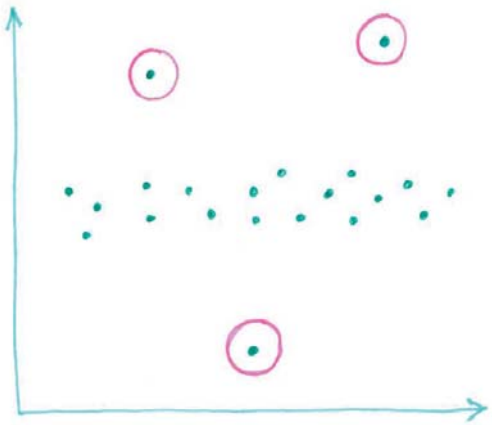
Ще одне застосування завдань зменшення розмірності – рекомендаційні системи, алгоритми, що намагаються передбачити, які об'єкти будуть цікаві користувачеві, маючи певні дані про його профіль. Такі системи не враховують оцінки всіх користувачів, а спираються лише на переваги самого користувача й пропонують йому, наприклад, подивитися певний фільм.

1.2.6 Виявлення аномалій

Мета – виявити аномальні відхилення від стандартних випадків. На перший погляд, завдання дуже подібне до завдання кла-

сифікації, проте має істотну відмінність: аномалії – явище рідкісне, тому вибірок, на яких можна навчити машинну модель, або дуже мало, або зовсім немає. Саме тому методи класифікації тут не працюють.

Наприклад, клієнт банку багато років знімає зі свого рахунку приблизно однакові суми з приблизно однаковою періодичністю, перебуваючи в Україні. Та одного разу приходить запит на підтвердження транзакції з цього рахунку на велику суму, проте людина, яка робить запит, перебуває в Індії. Така транзакція, ймовірно, буде вважатися аномалією й потребуватиме додаткової перевірки.



На практиці аномалії допомагають виявити шахрайство у банку, медичні проблеми або помилки в тексті.

1.2.7 Пошук правил

Завдання пошуку правил шукає закономірності в потоці даних. Наприклад, якщо в супермаркеті між полицею з маслом і касою поставити стійку з хлібом, з високою ймовірністю кількість проданого хліба зросте. Але тут усе просто – хліб часто купують із маслом, а коли товарів багато, виявлення таких правил може істотно збільшити прибуток.

Ще більш істотним є передбачення поведінки користувача на інтернет-ресурсах. За яким товаром він повернеться? Які товари можна продати «навздогін» до вже заданого? На які розділи сайту направити користувача, щоб він залишив у вас більше грошей?

1.3 Опис за ознаками

Типи ознак:

Поділ ознак на *якісні* та *кількісні* – основоположний принцип поділу ознак на типи. Щоб визначити тип, потрібно з'ясувати, чи об'єктивно можна виміряти досліджувану характеристику за допомогою чисел.

Якісні ознаки. В інформації наведені характеристики, вимірювані числами, тоді як самі спостереження можна поділити на кількість груп. Інформацію, що зберігається в такому типі змінної, важко виміряти, а виміри можуть бути суб'єктивними. Смак, колір автомобіля, архітектурний стиль, сімейне становище – усе це типи якісних ознак. Такі ознаки також називають *категоріальними*.

- **Номінальні ознаки** служать для позначення змінних, що не мають кількісного вираження. Номінальні ознаки не мають порядку, тому в разі зміни порядку значень підсумковий результат не змінюється. У науці про дані можна використовувати пряме кодування, щоб перетворити номінальні ознаки на числові.

- **Порядкові ознаки** – це суміш числових і категоріальних

ознак. Ознаки можна розбити на категорії, але числа, що асоціюються з кожною категорією, мають значення. Наприклад, рейтинг ресторану від 0 (найнижчий) до 4 (найвищий) зірок – це приклад порядкових ознак. Порядкові ознаки часто обробляють як категоріальні, коли під час побудови діаграм і графіків ознаки поділяють на впорядковані групи. Однак на відміну від категоріальних числа порядкових ознак мають математичне значення. Таким чином, порядкові ознаки – це майже те саме, що й номінальні, з тією лише відмінністю, що в номінальних порядок не має значення.

Порядкові шкали зазвичай використовують для вимірювання нечислових властивостей, таких як щастя, рівень задоволеності клієнтів, успішність студентів у класі, рівень кваліфікації тощо. Такі ознаки можна узагальнювати за допомогою частотності, пропорцій, відсоткових часток, а візуалізувати – за допомогою колових і стовпчастих діаграм. Крім того, можна використовувати відсотки, медіану, моду, міжквартильний розмах.

На додаток до порядкових та номінальних є особливий тип категоріальних ознак – бінарні (двійкові). Бінарні ознаки набувають лише два значення: «так» або «ні», що можна подати як 1 та 0. Бінарні ознаки широко використовуються в класифікаційних моделях машинного навчання. Як приклади бінарних ознак можна навести такі ситуації: скасувала людина підписку чи ні, купила машину чи ні.

Кількісні ознаки. Інформацію записують у вигляді чисел, воня являє собою об'єктивне вимірювання чи підрахунок. Температура, вага, кількість транзакцій – ось приклади кількісних ознак. Такі ознаки також називають *числовими*.

- **Дискретні кількісні ознаки** – це підрахунок випадків наявності характеристики, результату, предмета, діяльності. Ці виміри неможливо поділити на більш дрібні частини без втрати сенсу. Наприклад, у сім'ї може бути одна чи дві машини, але їх не може бути 3/2. Таким чином, існує кінцева кількість

можливих значень, які можна зареєструвати в процесі спостережень. У дискретних змінних можна підрахувати та оцінити інтенсивність потоку подій або зведену кількість (медіана, мода, середньоквадратичне відхилення). Наприклад, у 2014 році кожна американська сім'я мала в середньому по 2,11 транспортного засобу. Звичайний спосіб графічного подання дискретних змінних – стовпчасті діаграми, де кожний окремий стовпчик є окремим значенням, а висота стовпчика означає його пропорцію до цілого.

- **Неперервні ознаки** можуть набувати практично будь-якого числового значення і можуть бути поділені на менші частини, включаючи дробові та десяткові значення. Неперервні змінні часто вимірюють за шкалою. Якщо ви вимірюєте висоту, вагу, температуру, то маєте справу з безперервними ознаками. Наприклад, середній зріст в Індії становить 5 футів 9 дюймів ($\sim$ 175 см) для чоловіків і 5 футів 4 дюйми ($\sim$ 162 см) – для жінок.

  Неперервні ознаки поділяють на два типи.

  – *Інтервальні ознаки* – це впорядковані одиниці, що мають однакову відмінність одна від одної. Таким чином, ми говоримо про інтервальні ознаки, коли є змінна, яка містить упорядковані числові значення, і нам відомі точні відмінності цих значень. Прикладом може бути температура в заданому місці. Проблема зі значеннями інтервальних ознак полягає в тому, що в них немає «абсолютного нуля».

  – *Ознаки співвідношення*, що також є впорядкованими одиницями з однаковими відмінностями один від одного. Це практично те саме, що й інтервальні ознаки, проте ознаки співвідношення мають «абсолютний нуль». Відповідні приклади – висота, вага, довжина.

Працюючи з неперервними ознаками, можна використовувати майже всі методи: процентиль, медіану, міжквартильний розмах, середнє арифметичне, моду, середньоквадратичне відхилення, амплітуду.

Для візуалізації неперервних даних можна скористатися гістограмою чи діаграмою розмаху. За допомогою гістограми можна визначити середнє значення та крутість розподілу, мінливість і модальність. У цьому разі, гістограма не показує викидів, для цього потрібно використовувати діаграму розмаху.

1.4 Поставлення задачі навчання

- Попередньо проаналізувати дані.
- Підібрати (написати) метод навчання й аналізування.
- Поділити дані на дві частини:
  - навчальну вибірку (більшу частину);
  - тестувальну вибірку.
- Навчити машину за наявною базою даних (навчальною вибіркою) приймати необхідне рішення – побудувати алгоритм прийняття рішень.
- Перевірити побудований алгоритм на тестувальній вибірці.
- Провести перевірку алгоритму в реальному світі.

Наведемо приклад поставлення задачі.

Необхідно зробити прогноз, чи зіграє гравець у теніс за певних погодних умов?

Дані (ознаковий опис):
Існують дані (N записів) щодо попередніх ігор гравця за певних погодних умов.

Ознаки:

1. погодні умови (сонячно, похмуро, дощ);
2. вологість (нормальна, висока);
3. вітер (слабкий, сильний).

Результат – відбулася чи не відбулася гра.

Задача – навчити машину за наявною базою даних (навчальною вибіркою) приймати необхідне рішення – побудувати алгоритм прийняття рішень.

Дані: 14 записів (об'єктів) із трьома ознаками:

1. Поділяємо всі дані на навчальну та тестувальну вибірки.
2. Обираємо алгоритм навчання (метод) та навчаємо машину на навчальній вибірці.
3. Перевіряємо алгоритм на тестувальній вибірці – оптимізуємо параметри алгоритму.
4. Прогнозуємо результат для нових вхідних даних.

№	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No
15	Rain	High	Weak	???

1.5 Приклади прикладних задач

1.5.1 Задачі медичної діагностики

Об'єкт – пацієнт у певний момент часу.
Можливі класи: діагноз, спосіб лікування, результат захворювання.
Приклади можливих ознак:

- бінарні: стать, головний біль, слабість, нудота та ін.;
- кількісні: зріст, вага, тиск, пульс, кількість гемоглобіну в крові, доза препарату та ін.

Особливості задачі:

- зазвичай багато пропусків у даних;
- необхідно виділяти синдроми – поєднання симптомів;
- необхідно оцінювати імовірність негативного результату.

1.5.2 Задачі кредитного скорингу

Об'єкт – заявка на видачу банком кредиту.
Можливі класи: 1 (так) або 0 (ні).
Приклади можливих ознак:

- бінарні: стать, наявність телефону, наявність автомобіля, наявність нерухомого майна, наявність кредитів в інших банках та ін.;
- номінальні: місце проживання, місце роботи, посада;
- кількісні: заробітна плата, дохід родини, сума кредиту та ін.

Особливості задачі:

- необхідно оцінювати ймовірність дефолту.

1.5.3 Задача передбачення відпливу клієнтів

Об'єкт – абонент у певний момент часу.
Можливі класи: 1 (так – піде) або 0 (ні – не піде).
Приклади можливих ознак:

- бінарні: чи корпоративний клієнт, чи користується послугами та ін.;
- номінальні: тарифний план, регіон проживання;
- кількісні: тривалість дзвінків (вхідних та вихідних), кількість повідомлень (СМС), кількість інтернету (МБ), частота оплати послуг на ін.

Особливості задачі:

- дуже великі вибірки даних;
- важко виділяти ознаки за «сирими» даними;
- необхідно оцінювати імовірність відпливу клієнта.

1.5.4 Задача прогнозування вартості нерухомості

Об'єкт – квартира в Києві.
Результат передбачення: вартість квартири.
Приклади можливих ознак:

- бінарні: наявність метро поблизу, балкона, ліфта, охорони, паркінгу та ін.;
- номінальні: район міста, тип будинку (цегляний / панельний / блочний / моноліт);
- кількісні: кількість кімнат, площа квартири, поверх, відстань до метро, вік будинку та ін.

Особливості задачі:

- вибірка неоднорідна, вартість змінюється з часом;
- різні ознаки;
- необхідне перетворення ознак на числові.

1.5.5 Задача відкриття нового ресторану

Об'єкт – місце для відкриття нового ресторану.
Результат передбачення: прибуток ресторану через один рік.
Приклади можливих ознак:

- бінарні: наявність поблизу шкіл, офісів, метро та ін.;
- номінальні: район міста;
- кількісні: середній вік потенційних клієнтів, їх достаток та ін.

Особливості задачі:

- мало об'єктів, багато ознак;
- різні ознаки;
- різнорідні об'єкти (можливо краще будувати окремі моделі для великих та малих міст).

1.6 Машинне навчання на даних складної структури

- **Статистичний машинний переклад**
  Об'єкт – речення мовою оригіналу.
  Відповідь – речення іншою мовою.

- **Переведення мови і текст**
  Об'єкт – аудіозапис мови людини.
  Відповідь – текстовий запис мови.

- **Комп'ютерний зір (автопілот автомобіля)**
  Об'єкт – зображення або відеоряд.
  Відповідь – рішення (об'їхати, зупинитися, ігнорувати).

- **Передумови успішного розв'язування задач зі складними даними**
  - великі та чисті дані (Big Data);
  - методи зменшення розмірності вхідних даних;
  - методи оптимізації для задач великої розмірності;
  - глибокі нейромережеві архітектури;
  - зростання обчислювальних потужностей (GPU).

- **Особливості даних:**
  - різнорідні (ознаки виміряні за допомогою різних шкал);
  - неповні (виміряні не всі, є прогалини);
  - неточні (є похибки);
  - суперечливі (об'єкти однакові, а відповіді різні);
  - надлишкові (дуже великі, не вміщуються в пам'ять машини);
  - недостатні (об'єктів менше, ніж ознак);
  - неструктуровані (відсутній ознаковий опис).

Розділ 2

Математична формалізація задачі машинного навчання

2.1 Три складових задачі машинного навчання

1. Множина об'єктів X^N (інформаційний стан): $X = \{x_i, \ldots, x_N\}$.
2. Множина відповідей Y (оцінювання, передбачення, прогноз) – існує лише для задач «навчання з учителем».
3. Цільова функція (Target function) $y : X \to Y$ — невідома залежність, яка для кожного інформаційного стану ставить у відповідність певну відповідь.

Кожний об'єкт характеризується певним вектором ознак:

$$f_j : \ X \to D_j, \ j = 1, \ldots, M.$$

Ознаки об'єктів у числовому поданні:

- бінарна ознака $D_j = \{0, 1\}$;
- номінальна ознака $|D_j| < \infty$;
- порядкова ознака $|D_j| < \infty$, D_j упорядковано;
- кількісна ознака $D_j = \mathbb{R}$.

Вектор $(f_1(x), \ldots, f_M(x))$ – ознаковий опис об'єкта x.

Для N об'єктів $\{x_i, \ldots, x_N\}$ *набір даних* – це **матриця** F **«об'єкти-ознаки»** розміром $N \times M$:

$$F = ||f_j(x_i)||_{N \times M} = \begin{pmatrix} f_1(x_1) & \cdots & f_M(x_1) \\ \vdots & \vdots & \vdots \\ f_1(x_N) & \cdots & f_M(x_N) \end{pmatrix}.$$

Типі відповідей для різних задач

Задачі навчання з учителем

- Задачі класифікації (classification):
  - $Y = \{-1, 1\}$ або $Y = \{0, 1\}$ – класифікація на два класи;
  - $Y = \{1, K\}$ – класифікація на K класів;
  - $Y = \{0, 1\}^K$ – класифікація на K класів, що можуть перетинатися.
- Задачі відтворення регресії (regression):
  $Y = \mathbb{R}$ або $Y = \mathbb{R}K$.
- Задачі ранжування (ranking, learning to rank):
  Y – кінцева упорядкована множина.

Задачі навчання без учителя

Відповідей немає. Необхідно щось робити із самими об'єктами.

2.2 Модель алгоритмів для задач навчання з учителем

Вхідні дані:

- множина об'єктів X^N (інформаційний стан): $X = \{x_i, \ldots, x_N\}$;
- кожний об'єкт характеризується певним вектором ознак $f_j : X \to D_j,\ j = 1, \ldots, n$;
- відомі відповіді Y^N: $Y = \{y_i, \ldots, y_N\}$.

Завдання:

- знайти алгоритм a: $X \to Y$, що визначає функцію Y та наближає її на всій множині X.

Етапи навчання

1. Проводимо первинне оброблення даних: перевіряємо дані на «чистоту», оцінюємо кореляції між ознаками f_i та f_j $(i \neq j)$; ознак f_i, $i = 1, \ldots, n$ з цільовим вектором Y.
2. Ділимо всі дані випадковим чином на навчальну (X_{train}, Y_{train}) розміром $\ell < N$: $X_{train} = \{x_1, \ldots, x_\ell\}$, $Y_{train} = \{y_1, \ldots, y_\ell\}$ та тестову (X_{test}, Y_{test}) розміром $N - \ell$: $X_{test} = \{x_{\ell+1}, \ldots, x_N\}$, $Y_{test} = \{y_{\ell+1}, \ldots, y_N\}$, зазвичай у співвідношенні 70 % на 30 %.
3. Визначаємося з методом навчання μ.
4. **Етап навчання (train).** Для матриця «об'єкти-ознаки» розміром $\ell \times n$ із використанням цільового вектору Y^ℓ та обраного методу навчання μ будуємо алгоритм прийняття рішень a:

$$\begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \cdots & \cdots & \cdots \\ f_1(x_\ell) & \cdots & f_n(x_\ell) \end{pmatrix} \xrightarrow{y} \begin{pmatrix} y_1 \\ \cdots \\ y_\ell \end{pmatrix} \xrightarrow{\mu} a$$

5. **Етап тестування (test).** Для матриці «об'єкти-ознаки» розміром $k \times n$, $k = N - \ell$ з використанням побудованого алгоритму a одержуємо відповіді $a(x'_j)$:

$$\begin{pmatrix} f_1(x'_1) & \cdots & f_n(x'_1) \\ \cdots & \cdots & \cdots \\ f_1(x'_k) & \cdots & f_n(x'_k) \end{pmatrix} \xrightarrow{a} \begin{pmatrix} a(x'_1) \\ \cdots \\ a(x'_k) \end{pmatrix}.$$

2.2.1 Функціонали якості

Для оцінювання якості роботи алгоритму a використовують функціонал якості (loss function) $\mathcal{L}(a, x)$, що визначає величину помилки алгоритму $a \in A$ на кожному об'єкті $x \in X$.
Функція втрат для задач класифікації:

- $\mathcal{L}(a, x) = [a(x) \neq y(x)]$ – індикатор помилки.

Функція втрат для задач регресії:

- $\mathcal{L}(a, x) = |a(x) - y(x)|$ – абсолютне значення помилки;
- $\mathcal{L}(a, x) = (a(x) - y(x))^2$ – квадратична помилка.

Емпіричний ризик – функціонал якості роботи алгоритму a на вибірці X^ℓ:

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

2.2.2 Зведення задачі навчання до задачі оптимізації

Метод мінімізації емпіричного ризику (Empirical Risk Minimization, ERM):

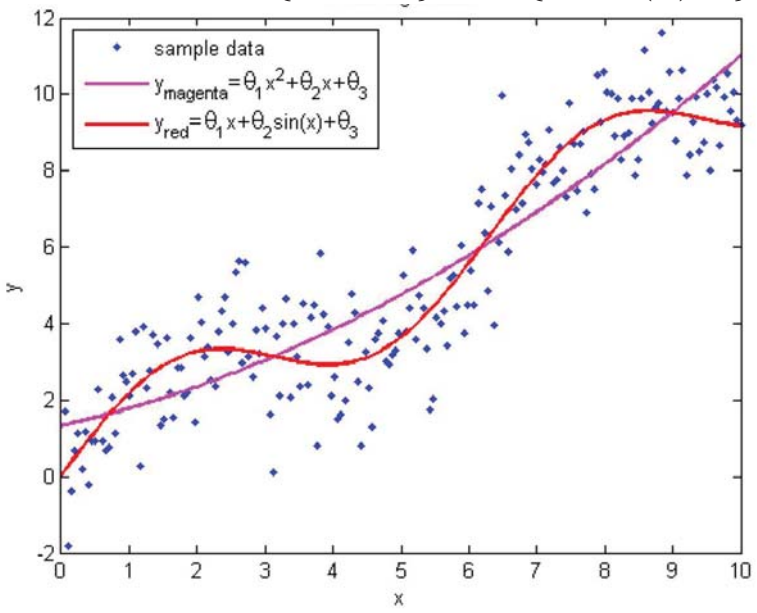
$$\mu(X^\ell) = \arg\min_{a \in A} Q(a, X^\ell),$$

дозволяє побудувати такий алгоритм a, який буде давати мінімальну сумарну помилку на тестовій вибірці X^ℓ.

Навчання регресії – оптимізація:

- задача регресії: $X = \mathbb{R}^n$, $Y = \mathbb{R}$;
- n числових ознак $f_j(x)$, $j = 1, \ldots, n$;
- навчальна вибірка: $X^\ell = (x_i, y_i)_{i=1}^\ell$;
- тестова вибірка: $X^k = (x'_i, y'_i)_{i=1}^k$;
- модель регресії – лінійна: $a(x, \theta) = \langle x, \theta \rangle = \sum\limits_{j=1}^{n} \theta_j f_j(x)$, $\theta \in \mathbb{R}^n$;

$\ell = 200$, $n = 3$: $\{x^2, x, 1\}$ або $\{x, \sin(x), 1\}$;



- квадратична функція втрат: $\mathcal{L}(a, x) = (a(x) - y(x))^2$;
- метод навчання на навчальній вибірці $X^\ell = (x_i, y_i)_{i=1}^\ell$ – метод найменших квадратів як окремий випадок ERM:

$$Q(\theta) = \sum_{i=1}^{\ell} (a(x_i, \theta) - y_i)^2 \to \min_\theta$$

- перевірка за тестовою вибіркою $X^k = (x'_i, y'_i)_{i=1}^k$:

$$Q'(\theta) = \frac{1}{k} \sum_{i=1}^{k} \left(a(x'_i, \theta) - y'_i\right)^2.$$

Навчання класифікації – оптимізація:

- задача класифікації: $X = \mathbb{R}^n$, $Y = -1, 1$;
- n числових ознак $f_j(x)$, $j = 1, \ldots, n$;
- навчальна вибірка: $X^\ell = (x_i, y_i)_{i=1}^\ell$;
- тестова вибірка: $X^k = (x'_i, y'_i)_{i=1}^k$;
- модель класифікації – лінійна: $a(x, \theta) = \text{sign}\langle x, \theta\rangle = \text{sign} \sum\limits_{j=1}^n \theta_j f_j(x)$, $\theta \in \mathbb{R}^n$;
- функція втрат – бінарна або її апроксимація:

$$\mathcal{L}(a, y) = [ay < 0] = [\langle x, \theta\rangle y < 0] \leq \mathcal{L}(\langle x, \theta\rangle y);$$

- метод навчання на навчальній вибірці $X^\ell = (x_i, y_i)_{i=1}^\ell$ – метод градієнтного спуску як окремий випадок ERM:

$$Q(\theta) = \sum_{i=1}^\ell [\langle x_i, \theta\rangle y_i < 0] \leq \sum_{i=1}^\ell \mathcal{L}(\langle x_i, \theta\rangle y_i) \to \min_\theta;$$

- перевірка за тестовою вибіркою $X^k = (x'_i, y'_i)_{i=1}^k$:

$$Q'(\theta) = \frac{1}{k} \sum_{i=1}^k \left[\langle x'_i, \theta\rangle y'_i < 0\right].$$

2.2.3 Метод градієнтного спуску для мінімізації емпіричного ризику

Мінімізація емпіричного ризику (регресія, класифікація):

$$Q(\theta) = \sum_{i=1}^{\ell} \mathcal{L}_i(\theta) \to \min_{\theta}.$$

Мінімізація методом градієнтного спуску:

- θ_0 – початкове наближення;
- $\theta^{(t+1)} = \theta^{(t)} - h \cdot \nabla Q(\theta^{(t)})$, $\nabla Q(\theta^{(t)}) = \left(\frac{\partial Q(\theta)}{\partial \theta_j}\right)_{j=0}^{n}$,
  h – градієнтний крок (темп навчання); t – індекс епохи навчання;
- $\theta^{(t+1)} = \theta^{(t)} - h \sum\limits_{i=1}^{\ell} \nabla \mathcal{L}_i(\theta^{(t)})$.

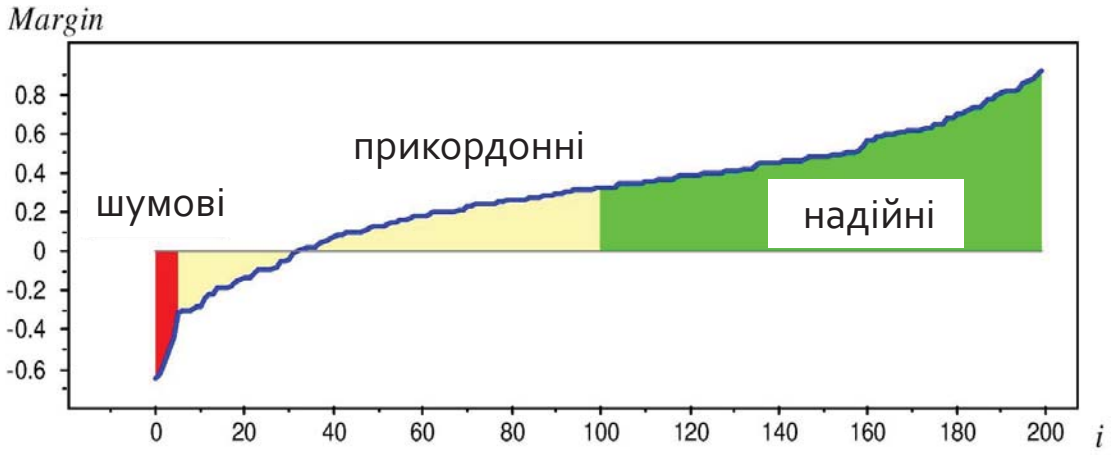
Ідея прискорення збіжності алгоритму: брати (x_i, y_i) по одному й відразу поновлювати вектор ваг θ.

2.2.4 Поняття відступу (marging) для класифікаторів

Роздільчий класифікатор $a(x, \theta) = \text{sign}(x, \theta)$:

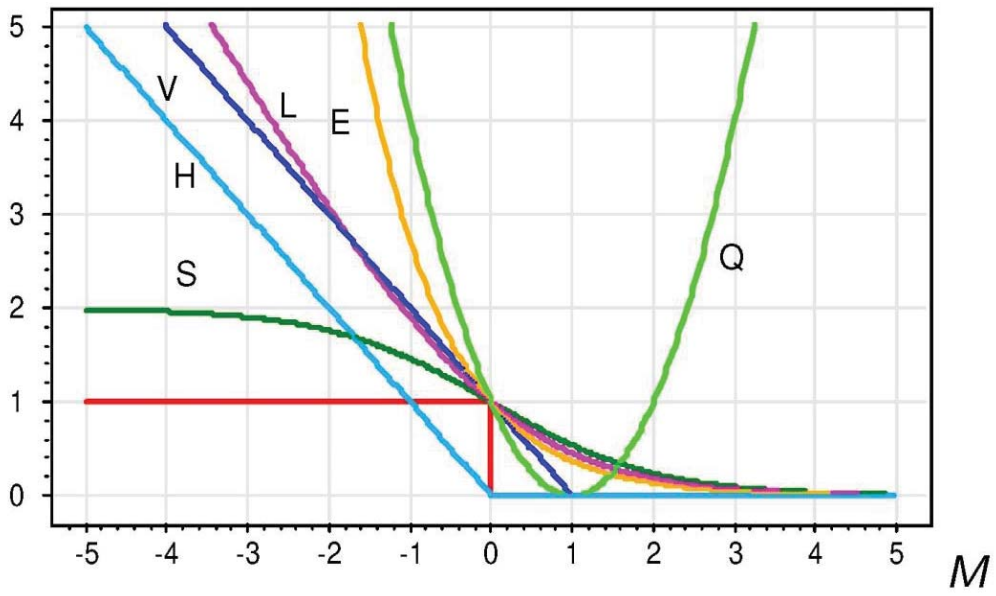
- $g(x, \theta)$ – роздільна (дискримінантна) функція;
- $g(x, \theta) = 0$ – рівняння роздільної поверхні;
- $M_i(\theta) = g(x_i, w)y_i$ – відступ (marging) об'єкта x_i;
- $M_i(\theta) < 0 \Leftrightarrow$ алгоритм a помиляється на об'єкті x_i.

Ранжування об'єктів за зростанням відступів $M_i(\theta)$.



2.2.5 Неперервні апроксимації порогової функції втрат

Часто використовувані неперервні функції втрат $\mathcal{L}(M)$:



$V(M) = (1-M)_+$	–	кусково-лінійна (SVM);
$H(M) = (-M)_+$	–	кусково-лінійна (Hebb's rule);
$L(M) = \log_2(1+e^{-M})$	–	логарифмічна (LR);
$Q(M) = (1-M)^2$	–	квадратична (FLD);
$S(M) = 2(1+e^{M})^{-1}$	–	сигмоїдна (ANN);
$E(M) = e^{-M}$	–	експоненціальна (AdaBoost);
$[M < 0]$	–	порогова функція втрат.

2.2.6 Проблема перенавчання

Приклад Рунге – Апроксимація функції поліномом

Завдання: апроксимувати функцію $y(x) = \frac{1}{1+25x^2}$ поліномом ступеня n на відрізку $x \in [-2; 2]$.

Модель поліноміальної регресії

$a(x, \theta) = \theta_0 + \theta_1 x + \cdots + \theta_n x^n$ – поліном ступеня n.

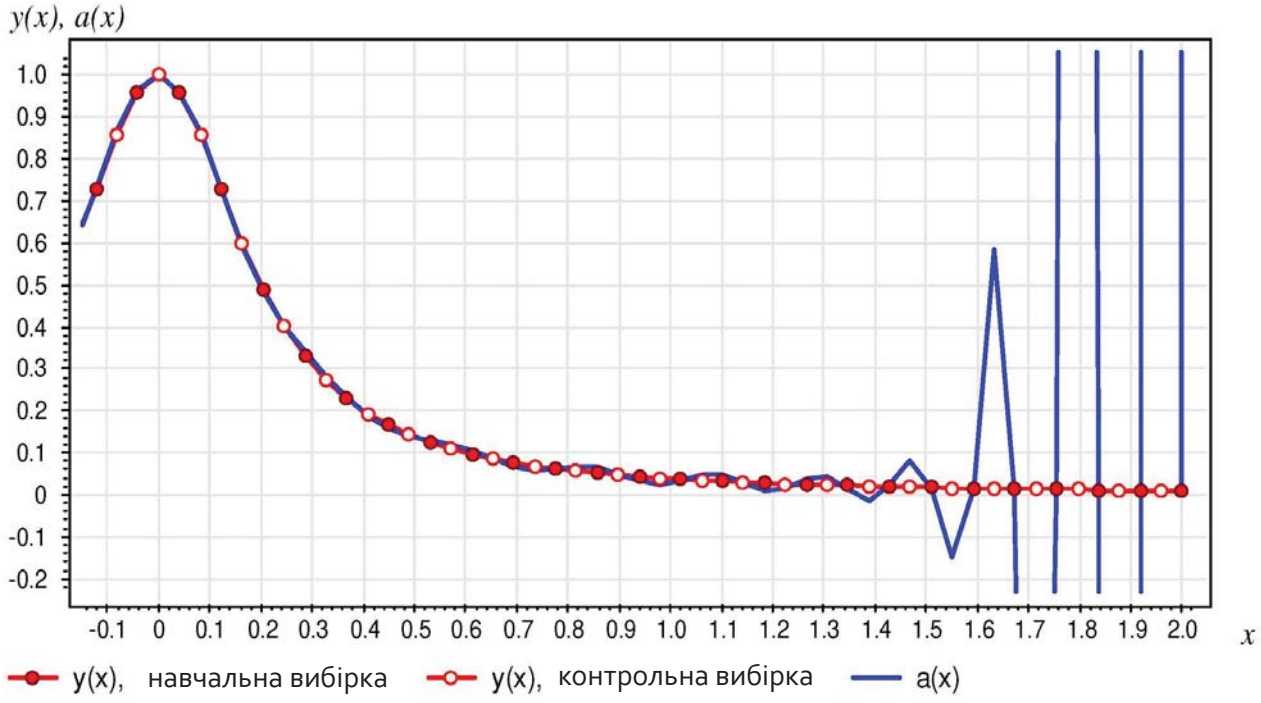
Навчання методом найменших квадратів:

$$Q(\theta, X^\ell) = \sum_{i=1}^{\ell} (\theta_0 + \theta_1 x_i + \cdots + \theta_n x_i^n - y_i)^2 \to \min_{\theta_0, \ldots \theta_n};$$

- навчальна вибірка: $X^\ell = \left\{ x_i = 4\frac{i-1}{\ell-1} - 2 | i = 1, \ldots, \ell \right\}$;
- контрольна вибірка: $X^k = \left\{ x_i = 4\frac{i-1/2}{\ell-1} - 2 | i = 1, \ldots, \ell - 1 \right\}$.

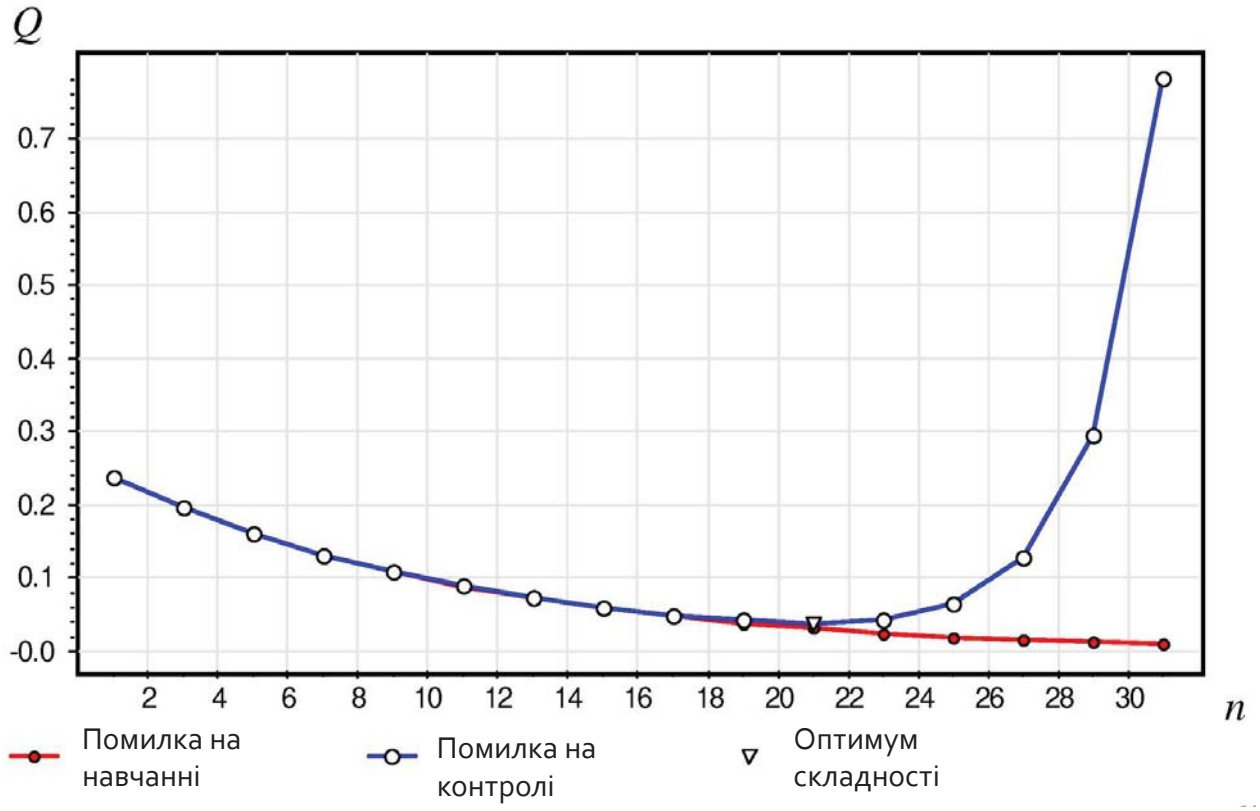
Запитання: Що відбувається з $Q(\theta, X^\ell)$ та $Q(\theta, X^k)$ у разі збільшення ступеня полінома n?

$$y(x) = \frac{1}{1 + 25x^2}, \qquad n = 38, \qquad \ell = 50;$$

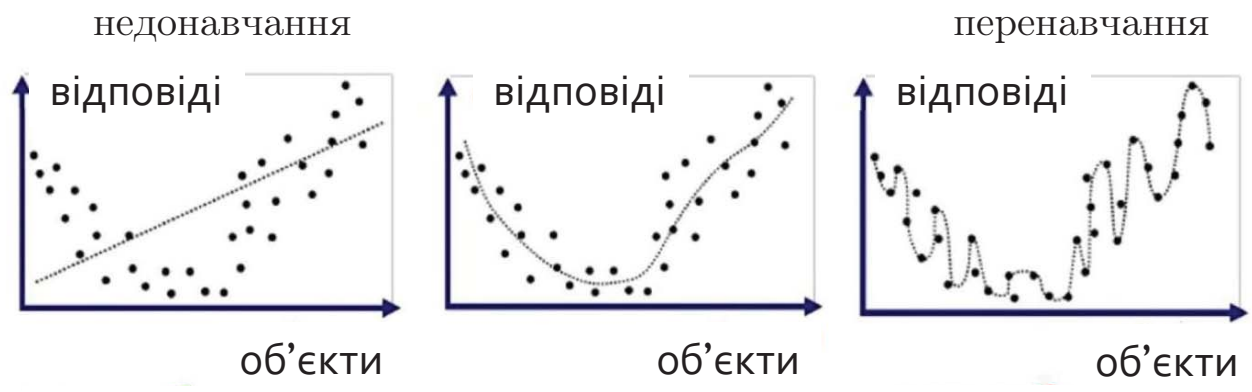


- помилка під час навчання: $Q(\mu(X^\ell), X^\ell)$;
- помилка під час контролю: $Q(\mu(X^\ell), X^k)$.

Залежність Q від ступеня полінома n



Перенавчання – це коли $Q(\mu(X^\ell), X^k) \gg Q(\mu(X^\ell), X^\ell)$.



Недонавчання (underfitting): модель досить проста, недостатня кількість параметрів n (ознак).
Перенавчання (overfitting): модель досить складна, завелика кількість параметрів n (ознак).

Перенавчання – ключова проблема в машинному навчанні.
Через що виникає перенавчання:

- надлишкові параметри в моделі $g(x, \theta)$ «витрачаються» на надмірно точну підгонку за навчальною вибіркою;
- вибір алгоритму a з множини алгоритмів A відбувається за неповною інформацією X^ℓ.

Як виявити перенавчання:

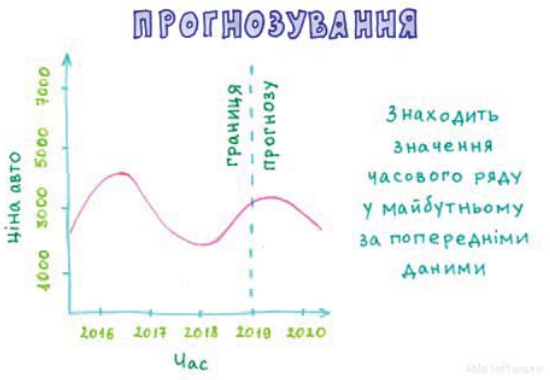
- емпірично, шляхом розбиття вибірки на навчальну (train) і тестову (test), причому на тестовій вибірці повинні бути відомі правильні відповіді.

Позбутися його не можна. Як його мінімізувати:

- накладати обмеження на θ (регуляризація);
- мінімізувати одну з теоретичних оцінок.

Розділ 3

Навчання з учителем: задачі прогнозування



У цьому розділі буде надано основний теоретичний матеріал щодо методів прогнозування для розв'язання задач передбачення часових рядів (знаходження майбутніх значень залежно від часу) та виявлення закономірностей між незалежними змінними, що спостерігаються / вимірюються (їх також називають регресорами, ознаками), і цільовою змінною. Буде розглянуто:

- моделі регресії: лінійну, поліноміальну;
- метод навчання регресії – метод найменших квадратів;
- методи регуляризації.

Буде наведено приклад розв'язування задачі регресійного аналізу, поставлено завдання для практичної роботи та наведено приклад подання результатів.

3.1 Регресія

Регресія – статистичний метод, що дозволяє будувати математичні залежності значень даних, у яких можна прогнозувати майбутні значення. Регресія належить до класу задач навчання з учителем, оскільки оперує з відомим цільовим вектором.

Мета регресійного аналізу – пошук зв'язку між вектором вхідних незалежних змінних (даних що спостерігаються/вимірюються) (x_i) та залежної від них змінною – цільовим значенням y. Наприклад, знаючи дозування певного препарату і температуру пацієнта потрібно спрогнозувати його тиск (тут незалежні змінні, що спостерігаються/вимірюються: дозування й температура, цільова змінна – тиск); за днем місяця та вологістю повітря (змінні, що спостерігаються) – температуру повітря (цільова змінна). Завдання відновлення залежності між такими парами змінних називають регресією. Регресію розрізняють за двома типами: парна та множинна. Відмінність двох типів полягає в кількості незалежних змінних (одна ознака або вектор ознак), що впливають на y.

Залежності між спостережуваними та цільовою змінними можуть бути будь-якими, зокрема як завгодно складними. Тому під час проведення регресійного аналізу підбирають модель, яка за значеннями спостережуваних змінних буде видавати значення цільової змінної, максимально «близьке» до істинного, з обмеженого конкретного набору моделей. Одним із найбільш простих класів (наборів) моделей є клас лінійних моделей. Модель регресії одержує на вхід значення спостережуваних змінних, перетворює їх, комбінує й видає значення цільової змінної.

3.1.1 Метрики оцінювання

Оцінювання будь-якої моделі машинного навчання – найважливіше завдання, що супроводжує моделювання даних. Крім того, деякі метрики допомагають оцінити саму підігнану модель. Розглянемо основні метрики оцінювання моделі лінійної регресії.

MAE (mean absolute error) – середня абсолютна помилка – універсальна метрика, що дозволяє дізнатися про різницю між фактичними та прогнозованими значеннями. Її розраховують за такою формулою:

$$MAE = \frac{1}{\ell} \sum_{i=1}^{\ell} \left(|y_i - \hat{y}_i| \right),$$

де

- ℓ – кількість даних (розмір вибірки);
- y_i – відомий результат;
- $\hat{y}_i$ – прогнозована відповідь.

MSE (mean squared error) – середня квадратична помилка, яку можна розглядати як уточнену MAE, оскільки вона допомагає знаходити помилки за допомогою квадратичної різниці між фактичними та прогнозованими значеннями:

$$MSE(y, \hat{y}) = \frac{1}{\ell} \sum_{i=1}^{\ell} (y_i - \hat{y}_i)^2 .$$

RMSE (root mean squared error) – корінь квадратний із середньої квадратичної помилки – також показує різницю між фактичними та прогнозованими значеннями, витягуючи корінь квадратний із середньої квадратичної помилки:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} (y_i - \hat{y}_i)^2}.$$

MSLE (mean squared logarithmic error) – середня квадратична логарифмічна помилка – знаходить помилки за допомогою квадратичної різниці логарифмічно перетворених прогнозованих та фактичних значень:

$$MSLE(y, \hat{y}) = \frac{1}{\ell} \sum_{i=1}^{\ell} \left(\ln(y_i + 1) - \ln(\hat{y}_i + 1) \right)^2 .$$

Щоб уникнути визначення натурального логарифму нуля, до обох значень, відомого та прогнозованого, додають одиницю. Цей показник найкраще використовувати, коли значення залежної змінної y мають експоненціальне зростання, наприклад кількість населення, середні продажі товару за проміжок років тощо. Цей показник штрафує занижену оцінку більше, ніж завищену оцінку.

RMSLE (root mean squared logarithmic error) – корінь квадратний із середньої квадратичної логарифмічної помилки – використовує логарифмічно перетворені прогнозовані та фактичні значення, які перевіряють за коренем квадратним із середньої квадратичної помилки:

$$RMSLE(y, \hat{y}) = \sqrt{\frac{1}{\ell}\sum_{i=1}^{\ell}\left(\ln(y_i+1) - \ln(\hat{y}_i+1)\right)^2}.$$

$\mathbf{R^2}$ (R-squared) – R-квадрат – також вважають універсальною метрикою, застосовуваною для оцінювання ефективності регресійної моделі. Коефіцієнт детермінації R^2 отримують за допомогою визначення частки дисперсії залежної змінної y, прогнозованої вектором незалежних змінних $\{x_i\}$. Він дає вказівку на відповідність і, отже, міру того, наскільки добре прогнозовані значення, ймовірно, будуть передбачені моделлю, через частку поясненої дисперсії. Розраховуються цю метрику за такою формулою:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{\ell}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{\ell}(y_i - \overline{y})^2}, \quad \overline{y} = \sum_{i=1}^{\ell} y_i.$$

Adjusted $\mathbf{R^2}$ – скоригований $\mathbf{R^2}$ – необхідний, якщо до даних додають нові ознаки. Ця метрика компенсує недоліки R-квадрата, які зменшуються або збільшуються зі збільшенням дисперсії ознак, її розраховують за формулою:

$$AdjustedR^2(y, \hat{y}) = 1 - \left(1 - R^2\right)\frac{\ell - 1}{\ell - k - 1},$$

де k – кількість незалежних змінних (ознак).

3.1.2 Модель лінійної регресії

Лінійна модель – модель, що виражається лінійною функцією. Слово «лінійна» у назві пояснюється тим, що графіком такої функції є пряма. Загалом лінійні функції однієї змінної (одна ознака x) записують так: $y(x) = kx + b$, де k і b – довільні числа. У разі, якщо є декілька спостережуваних змінних (вектор ознак $X = \{x_1, \ldots, x_n\}$), лінійні функції мають вигляд:

$$y(x_1, \ldots, x_n) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n,$$

де $w_0, w_1, \ldots, w_n$ – також довільні числа. Графік цієї функції теж пряма, але вже в n-вимірному просторі. Таким чином, лінійна регресія – моделювання залежності між спостережуваними та цільовими змінними за допомогою лінійних функцій.

Завдання підбору оптимальних коефіцієнтів $w_0, w_1, \ldots, w_n$ є навчанням моделі, що проводиться спеціальними алгоритмами оптимізації. Отже, задача регресії – задача оптимізації. Оптимальність коефіцієнтів визначається функцією втрат, що відображає, наскільки прогнози моделі близькі до цільових змінних. Ці коефіцієнти називають вагами моделі, оскільки за значенням коефіцієнта можна зрозуміти, наскільки значуща змінна (ознака) при ньому: чим більший коефіцієнт, тим більшу вагу має відповідна ознака (змінна) і, навпаки, якщо коефіцієнт близький до 0, то змінна при ньому ніяк не впливає на прогноз. Тому лінійна регресія легко інтерпретована, тобто за моделлю можна зрозуміти, які змінні важливі, а які – ні. На практиці лінійну регресію використовують в різних сферах. Умовно їх можна поділити на дві основні категорії.

Прогнозування. Якщо необхідно зробити прогноз на основі минулих даних, а залежні та незалежні змінні мають лінійну кореляцію, використовують модель лінійної регресії. До цієї категорії можна віднести прогнозування ситуації на фондовому ринку, прогноз погоди, прогнозування продажу тощо.

Оптимізація міцності зв'язків. Іноді під час аналізування даних може знадобитися визначити тип зв'язку й силу зв'язку двох або більше змінних. У таких ситуаціях використовують

лінійну регресію, що допомагає зрозуміти, який вигляд матиме змінна у разі змінювання інших змінних. Оптимізацію міцності зв'язку застосовують у медицині, роздрібній торгівлі, сільському господарстві.

Навчання моделі лінійної регресії зазвичай проводять методом найменших квадратів із використанням середньоквадратичної помилки MSE.

Нехай існує n-вимірний простір ознак $x_1, \ldots, x_n$. Для побудови моделі лінійної регресії $\hat{y} = w_0 + w_1 x_1 + \ldots + w_n x_n$ будемо вважати, що нульова ознака для всіх об'єктів дорівнює одиниці, $x_0 = 1$:

$$\hat{y}(\vec{x}) = w_0 + w_1 x_1 + \ldots + w_n x_n = \sum_{i=0}^{n} w_i x_i = \vec{x}^T \vec{w}.$$

Емпіричний ризик (функція вартості) набирає форми середньоквадратичної помилки:

$$\mathcal{L}(X, \vec{y}, \vec{w}) = \frac{1}{\ell} \sum_{i=1}^{\ell} \left(y_i - \vec{x}_i^T \vec{w}_i\right)^2 = \frac{1}{\ell} ||\vec{y} - X\vec{w}||^2 = \frac{1}{\ell} (\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w}),$$

рядки матриці X – це ознаковий опис об'єктів. Один з алгоритмів навчання μ такої моделі – це метод найменших квадратів. Обчислимо похідну функції вартості:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial}{\partial w} \frac{1}{\ell} \left(\vec{y}^T \vec{y} - 2\vec{y}^T X\vec{w} + \vec{w}^T X^T X\vec{w}\right) = \frac{1}{\ell} \left(-2X^T \vec{y} + 2X^T X\vec{w}\right),$$

прирівняємо до нуля і знайдемо розв'язок у явному вигляді:

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \quad \Rightarrow \quad \frac{1}{\ell} \left(-2X^T \vec{y} + 2X^T X\vec{w}\right) = 0,$$

$$-X^T \vec{y} + X^T X\vec{w} = 0 \quad \Rightarrow \quad X^T X\vec{w} = X^T \vec{y},$$

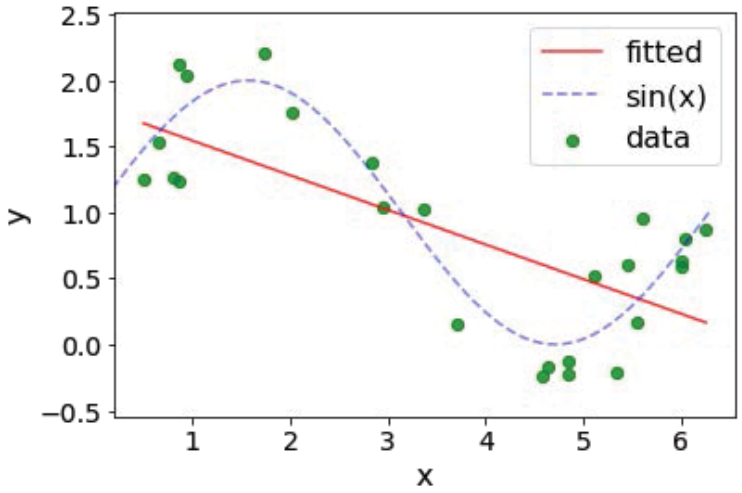
і зрештою

$$\vec{w} = \left(X^T X\right)^{-1} X^T \vec{y}.$$

Це й є алгоритм навчання моделі лінійної регресії.

Приклад

Розглянемо приклад навчання лінійної регресі для одновимірного простору ознак (лише одна ознака – незалежна змінна x). Згенеруємо вибірку: братимемо випадкову точку з функції $\sin(x)$ і додаватимемо до неї випадкову величину (шум), таким чином одержимо цільову змінну. Для одержаної вибірки проведемо навчання та розрахуємо коефіцієнти w_0 та w_1 моделі лінійної регресії $\hat{y} = w_0 + w_1 x$. Сформовані вибірка (data), функція $\sin(x)$ та побудована лінійна регресійна залежність (fitted) показані на рисунку.



Як бачимо, лінія не дуже збігається з даними та кривою $\sin(x)$. Середньоквадратична помилка дорівнює 0,271 умовних одиниць. Очевидно, що якщо б замість лінії ми використовували криву третього порядку, то результат був би набагато кращим.

3.1.3 Поліноміальна регресія

У лінійній регресії ми обмежували простір лише лінійними функціями від ознак. Давайте розширимо простір до всіх поліномів ступеня p. Тоді в нашому випадку однієї $n = 1$ ознаки x функція поліноміальної регресії матиме такий вигляд:

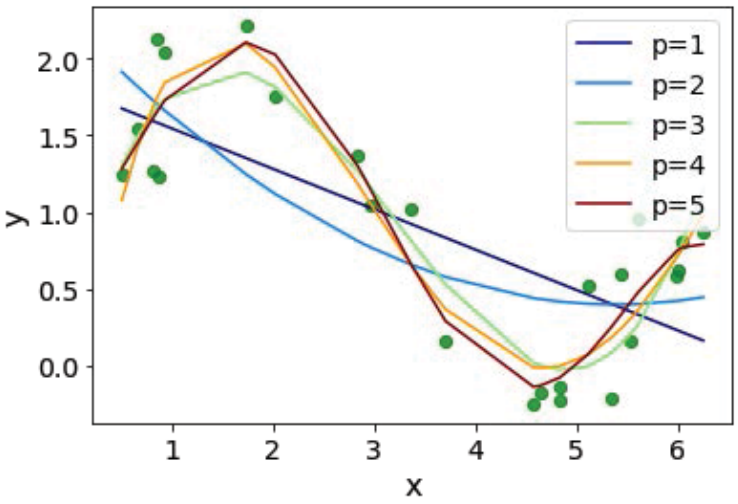
$$\hat{y_p}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_p x^p = \sum_{i=0}^{p} w_i x^i.$$

У цьому разі якщо, як і у випадку лінійної регресії, зафіксувати $x_0 = 1$, ввести певну заміну змінних $x = x_1$ та додаткові ознаки:

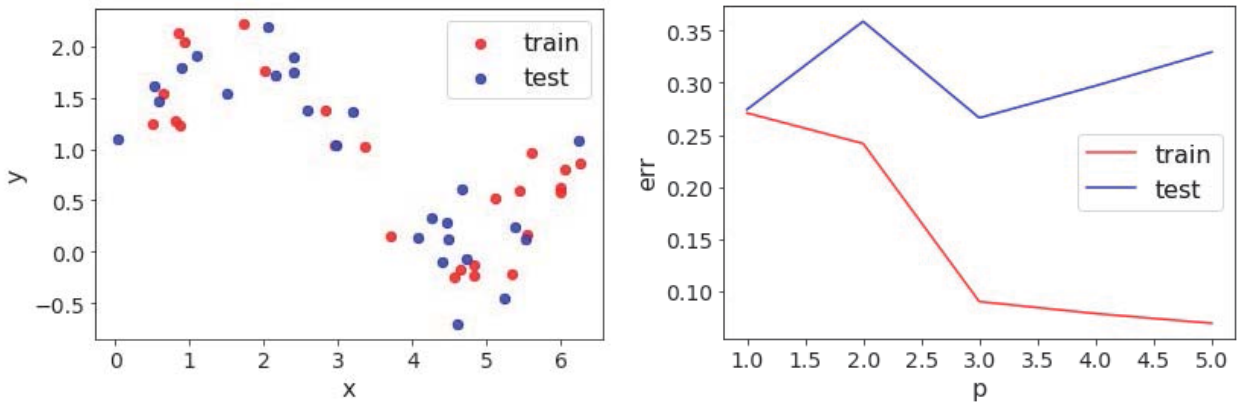
$x_2 = x_1^2$, $x_3 = x_1^3$, …, $x_p = x_1^p$, то ефективно переходимо від поліноміальної регресії ступеня p для однієї ознаки до лінійної регресії у просторі p ознак:

$$\hat{y_p}(x) \to \sum_{i=0}^{p} w_i x_i = \vec{x}^T \vec{w}, \quad x_i = x^i.$$

Отже, кожну ознаку математично виражають через першу і якщо заздалегідь перерахувати всі ступені ознак, то завдання знову зводиться до описаного вище алгоритму – методу найменших квадратів, коли розмірність n вектора ваг $\vec{w}$ визначають ступенем полінома p : $n = p + 1$. Використовуючи одержаний алгоритм навчання $\vec{w} = \left(X^T X\right)^{-1} X^T \vec{y}$, розглянемо результати для декількох значень p.



З одержаних результатів бачимо, що поліном другого ступеня ($p = 2$) все ще не досить гарно описує наші дані. Поліном третього порядку ($p = 3$) непогано описує наші дані. Далі проаналізуємо зміну середньої квадратичної помилки за зміни ступеня полінома для навчальної ($train$) й тестової ($test$) вибірок. Для цього аналогічно згенеруємо тестову вибірку. Навчальна й тестова вибірки подано на рисунку зліва, середньоквадратичні помилки для обох вибірок наведено справа.

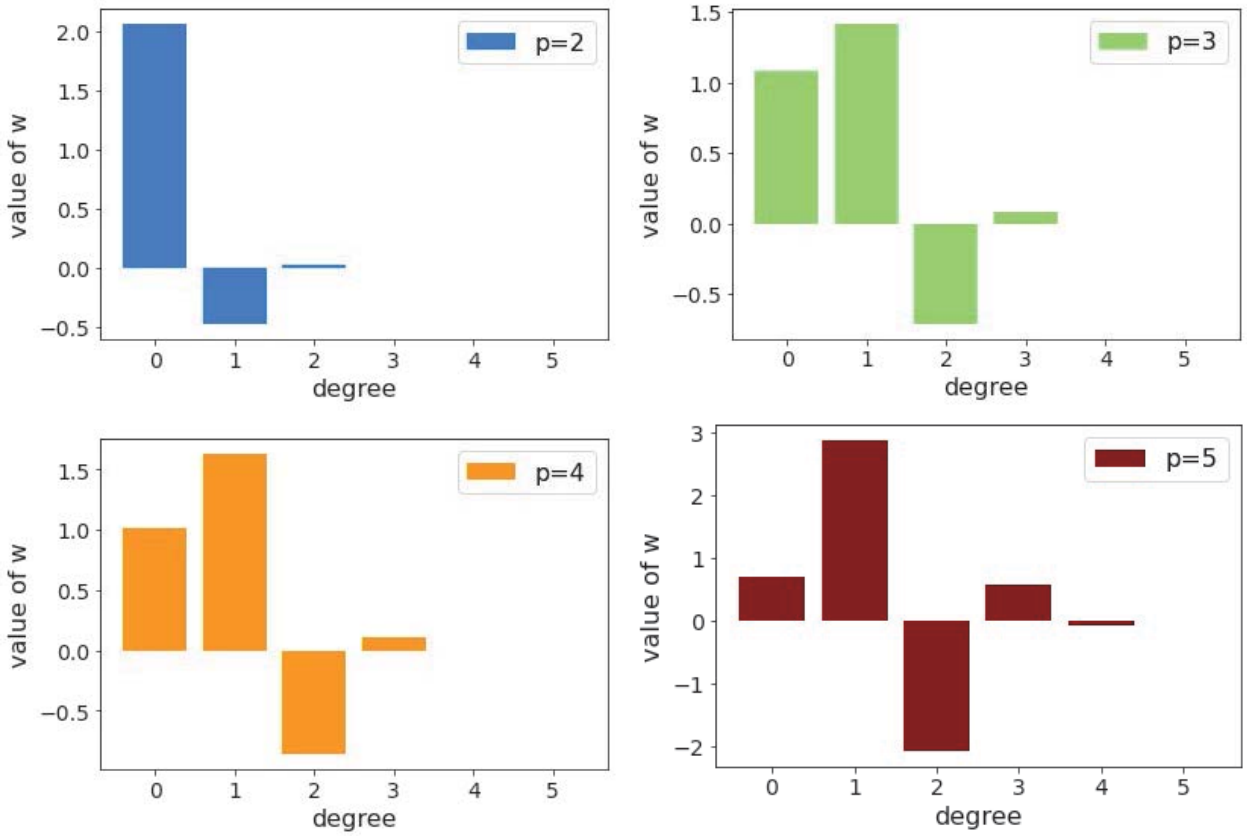


Бачимо, що в разі збільшення ступеня полінома середньоквадратична помилка для навчальної вибірки продовжує зменшуватися, хоча ми начебто були впевнені, що саме кубічний поліном повинен найкраще описувати наші дані.

Це явна ознака перенавчання, яку можна помітити з візуалізації навіть не використовуючи тестовий набір даних: у разі збільшення ступеня полінома вище від третього модель починає інтерполювати дані замість екстраполяції. Іншими словами, графік функції проходить точно через точки з тренувального набору даних, причому чим вищий ступінь полінома, тим через більшу кількість точок він проходить. Ступінь полінома відбиває складність моделі. Таким чином, складні моделі, в яких ступенів вільності досить багато, можуть запам'ятати весь тренувальний набір, повністю втрачаючи узагальнювальну здатність. Це і є проявом негативного боку принципу мінімізації емпіричного ризику.

Для тестової вибірки помилка поводить себе немонотонно в разі збільшення ступеня полінома. У цьому разі мінімальна помилка досягається якраз для кубічного полінома ($p = 3$). Отже, комбінуючи результати щодо помилок для обох навчальної й тестової вибірок доходимо висновку, що саме кубічний поліном якнайкраще описує вибірки. Необхідно зазначити, що на практиці в разі використання поліноміальної регресії саме поліном третього ступеня використовують найчастіше.

Розглянемо значення ваг w_i – параметрів поліноміальної регресії для кожного випадку $p = 2, \ldots, 5$.



Бачимо, що зі збільшенням ступеня полінома розмах значень коефіцієнтів зростає майже експоненціально. Крім того, вони ще й можуть набути різних за знаком значень.

Аналогічно до поліноміальної регресії можна використовувати лінійну комбінацію різних функціональних залежностей $f_i(x)$ від незалежної змінної:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i f_i(x).$$

У такому разі навчання методом найменших квадратів дозволить установити ваги $\vec{w}$, аналіз яких допоможе потім спростити модель, вилучаючи неістотні функціональні залежності, роблячи регресійну залежність більш простою без втрати узагальнювальних властивостей для досліджуваної вибірки.

3.1.4 Регуляризація

Розширення навчання лінійної моделі називають методами регуляризації. Вони спрямовані як на мінімізацію суми квадратів

помилки моделі на навчальних даних (із використанням методу найменших квадратів), так і на зниження складності моделі (наприклад, кількість або абсолютний розмір суми всіх коефіцієнтів у моделі), щоб запобігти перенавченню або виправити некоректно поставлене завдання.

Виділяють два популярні приклади процедури регуляризації лінійної регресії.

Гребенева регресія (Ridge Regression) – метод найменших квадратів модифікують так, щоб мінімізувати квадрат абсолютної суми коефіцієнтів (так звана регуляризація L_2).

Лассо-регресія (Lasso Regression) – метод найменших квадратів модифікують так, щоб звести до мінімуму абсолютну суму коефіцієнтів (так звана регуляризація L_1).

Ці методи ефективні у використанні, якщо є колінеарність у вхідних даних, і метод найменших квадратів відповідає навчальним даним.

Регуляризація L_2

Регуляризації L_2 досягають додаванням деякої апріорної інформації до умови задачі, наприклад так:

$$\mathcal{L}_{reg}\left(X, \vec{y}, \vec{w}\right) = \mathcal{L}\left(X, \vec{y}, \vec{w}\right) + \lambda R\left(\vec{w}\right),$$

де λ – коефіцієнт регуляризації, що визначає, наскільки необхідно враховувати умову R. У попередньому розділі ми побачили, що амплітуда значень коефіцієнтів w_i занадто велика. Спробуємо зменшити її, додавши обмеження L^2 на норму вектора параметрів:

$$R\left(\vec{w}\right) = \frac{1}{2}\left\|\vec{w}\right\|_2^2 = \frac{1}{2}\sum_{j=1}^{n} w_j^2 = \frac{1}{2}\vec{w}^T\vec{w}.$$

Нова функція вартості набере вигляду

$$\mathcal{L}\left(X, \vec{y}, \vec{w}\right) = \frac{1}{2}\left(\vec{y} - X\vec{w}\right)^T\left(\vec{y} - X\vec{w}\right) + \frac{\lambda}{2}\vec{w}^T\vec{w}.$$

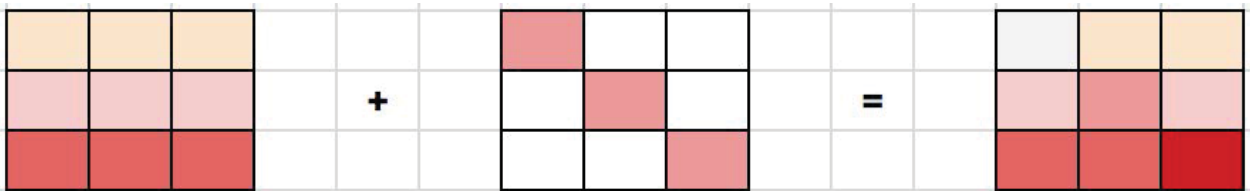
Обчислимо похідну за параметрами:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \vec{w}} &= \frac{\partial}{\partial \vec{w}}\left(\frac{1}{2}\left(\vec{y}-X\vec{w}\right)^T\left(\vec{y}-X\vec{w}\right)+\frac{\lambda}{2}\vec{w}^T\vec{w}\right)\\ &= \frac{\partial}{\partial \vec{w}}\left(\frac{1}{2}\left(\vec{y}^T\vec{y}-2\vec{y}^TX\vec{w}+\vec{w}^TX^TX\vec{w}\right)+\frac{\lambda}{2}\vec{w}^T\vec{w}\right)\\ &= -X^T\vec{y}+X^TX\vec{w}+\lambda\vec{w}.\end{aligned}$$

І знайдемо розв'язок у явному вигляді:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \vec{w}} = 0 &\Leftrightarrow -X^T\vec{y}+X^TX\vec{w}+\lambda\vec{w}=0\\ &\Leftrightarrow X^TX\vec{w}+\lambda\vec{w}=X^T\vec{y}\\ &\Leftrightarrow \left(X^TX+\lambda E\right)\vec{w}=X^T\vec{y}\\ &\Leftrightarrow \vec{w}=\left(X^TX+\lambda E\right)^{-1}X^T\vec{y},\end{aligned}$$

E – одинична діагональна матриця. Таку регресію називають гребеневою регресією (ridge regression). А гребенем є якраз діагональна матриця, яку ми додаємо до матриці X^TX з лінійно залежними колонками, в результаті одержувана матриця несингулярна.



Для такої матриці число обумовленості дорівнюватиме $\frac{e_{\max}+\lambda}{e_{\min}+\lambda}$, де e_x – це власні числа матриці. Таким чином, збільшуючи параметр регуляризації, ми зменшуємо число обумовленості, а обумовленість завдання покращується.

Регуляризація L_1

Спробуємо обмежити вектор параметрів моделі, використовуючи L^1-норму:

$$R\left(\vec{w}\right)=\left\|\vec{w}\right\|_1=\sum_{j=1}^{m}\left|w_j\right|.$$

Тоді задача набере вигляду

$$\mathcal{L}\left(X, \vec{y}, \vec{w}\right) = \frac{1}{2n} \sum_{i=1}^{n} \left(\vec{x_i}^T \vec{w} - y_i \right)^2 + \lambda \sum_{j=1}^{m} |w_j| \,.$$

Порахуємо похідну за параметрами моделі:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} \left(\vec{x_i}^T \vec{w} - y_i \right) \vec{x_i} + \lambda \text{sign}(w_j).$$

На жаль, така задача не має розв'язку у явному вигляді. Для пошуку гарного наближеного розв'язку дуже часто на практиці використовують метод **градієнтного спуску** (див. розділ 2.2.3), згідно з яким формула оновлення ваг набере вигляду:

$$\vec{w}_{\text{new}} := \vec{w} - \alpha \frac{\partial \mathcal{L}}{\partial \vec{w}},$$

а в задачі з'являється ще один параметр α, що відповідає за швидкість спуску – швидкість навчання (learning rate).

3.1.5 Поставлення задачі

Провести регресійний аналіз даних.

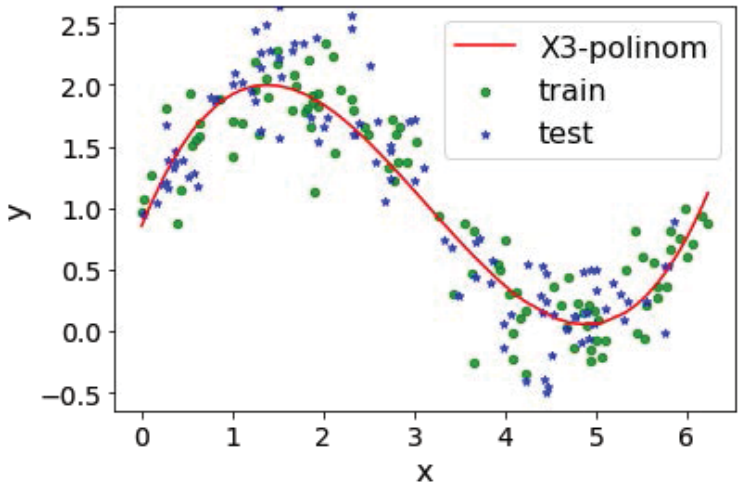
Етапи розв'язування

1. Імпортувати / згенерувати вибірку для задачі регресії.
2. Розділити всю вибірку на дві частини (навчальну й тестову).
3. Підібрати регресійну модель.
4. Провести регресійний аналіз:
   а) визначити модель регресії та за необхідності додати ознаки за допомогою нелінійного перетворення наявних;
   б) для обраної моделі порахувати коефіцієнти w_i;
   в) порахувати загальну помилку з використанням однієї з метрик оцінювання для навчальної й тестової вибірок.
5. Візуалізувати навчальну й тестову вибірки та результуючу регресійну залежність.
6. Порівняти результати з убудованою функцією із sklearn.
7. Оформити результати у вигляді звіту.

3.1.6 Приклад подання результатів

Поліноміальна регресія поліномом третього ступеня:

- поліном: $y = 0,85 + 1,85x - 0,86x^2 + 0,09x^3$;
- помилка на навчальній вибірці: $MSE \simeq 0,075$;
- помилка на тестовій вибірці: $MSE \simeq 0,164$.



Розділ 4

Навчання з учителем: задачі класифікації

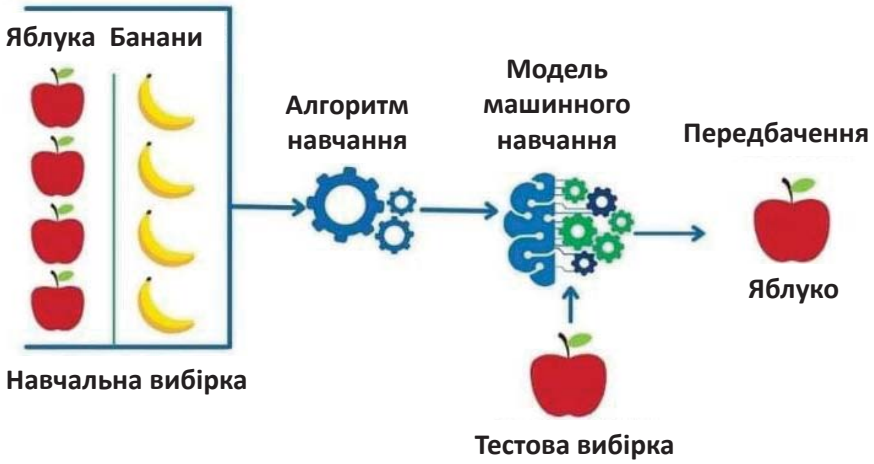
У цьому розділі буде надано основний теоретичний матеріал щодо методів класифікації даних. Алгоритми класифікації дозволяють розділити об'єкти відповідно до зазначених заздалегідь класів. Буде розглянуто чотири основних та найбільш часто використовуваних методи:

- дерево прийняття рішень (Decision Tree);
- k-найближчих сусідів (k-Nearest Neighbors);
- метод опорних векторів (Support Vector Machine);
- метод логістичної регресії (Logistic Regression).

Для кожного методу буде наведено приклад його використання, поставлено завдання для практичної роботи та наведено приклад подання результатів.

4.1 Класифікація

Класифікація – найпопулярніше завдання в усьому машинному навчанні – розділяє об'єкти за заздалегідь відомими ознаками. Для класифікації завжди потрібен **учитель** – розмічені дані з ознаками та категоріями, які машина вчитиметься визначати за цими ознаками.



Використовують для:

- спам-фільтрів;
- аналізування тональності;
- визначення мови;
- пошуку подібних документів;
- розпізнавання прописних літер та цифр;
- визначення підозрілих транзакцій.

Найпопулярніші методи (алгоритми) класифікації:

- простий байєсівський класифікатор (Naive Bayes classifiers);
- дерева пошуку рішень (Decisions Trees);
- k-найближчих сусідів (K-NN);
- метод опорних векторів (SVM, Support Vector Machine);
- логістична регресія (Logit Model);
- ...

4.2 Дерева прийняття рішень (Decision Tree)

Дерева рішень *Decision Tree* є одним із найбільш ефективних інструментів інтелектуального аналізування даних та передбачуваної аналітики, що дозволяють розв'язувати задачі *класифікації* та *регресії*.

Оскільки правила в деревах рішень одержують узагальненням безлічі окремих спостережень (*навчальних прикладів*), що описують *предметну сферу*, то за аналогією з відповідним методом логічного виведення їх називають індуктивними правилами, а процес *навчання* – індукцією дерев рішень.

У навчальній множині для прикладів повинне бути задане цільове значення, оскільки дерева рішень є моделями, які будують на основі *навчання з учителем*. У цьому разі якщо цільова змінна дискретна (*мітка класу*), то модель називають деревом класифікації, а якщо безперервна, то – деревом регресії.

4.2.1 Структура дерева рішень

Уведемо до розгляду основні поняття, використовувані в теорії дерев рішень.

Назва	Опис
Об'єкт.	Приклад, шаблон, спостереження.
Атрибут.	Ознака, незалежна змінна, властивість.
Цільова змінна.	Залежна змінна, мітка класу.
Вузол.	Внутрішній вузол дерева, вузол перевірки.
Кореневий вузол.	Початковий вузол дерева рішень.
Лист.	Кінцевий вузол дерева, вузол рішення.
Вирішальне правило.	Умова у вузлі, перевірка.

Власне, саме дерево рішень — метод подання *вирішальних правил* в ієрархічній структурі, що складається з елементів двох типів – вузлів (node) та листя (leaf). У вузлах містяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила за яким-небудь *атрибутом* навчальної множини.

У найпростішому випадку, в результаті перевірки множину прикладів, що потрапили до вузла, розбивають на дві підмножини,

в одну з яких потрапляють приклади, що задовольняють правило, а в інше, – що не задовольняють. Потім до кожної підмножини знову застосовується правило і процедура рекурсивно повторюється доти, поки буде досягнена деяка умова зупинення алгоритму. Внаслідок цього в останньому вузлі перевірка та розбиття не проводиться і він оголошується листом. Лист визначає рішення для кожного прикладу, що в нього потрапив. Для дерева класифікації це *клас*, що асоціюється з вузлом, а для дерева регресії модальний інтервал цільової змінної, що відповідає листу.

Отже, на відміну від вузла в листі міститься не правило, а підмножина об'єктів, що задовольняють усі правила гілки, яка закінчується цим листом. Очевидно, щоб потрапити в лист, приклад повинен задовольняти всі правила, що лежать на шляху до цього листа. Оскільки шлях у дереві до кожного листа єдиний, то й кожен приклад може потрапити лише в один лист, що забезпечує єдиність рішення.

4.2.2 Процес побудови

Процес побудови дерев рішень полягає в послідовному, рекурсивному розбитті навчальної множини на підмножини із застосуванням вирішальних правил у вузлах. Процес розбиття триває доти, поки всі вузли наприкінці всіх гілок не будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він міститиме єдиний об'єкт, або об'єкти лише одного класу), або після досягнення певної умови зупинення, що задається користувачем (наприклад, мінімально допустима кількість прикладів у вузлі або максимальна глибина дерева).

Алгоритми побудови дерев рішень належать до категорії *жадібних алгоритмів*. Жадібними називаються алгоритми, які допускають, що локально-оптимальні рішення на кожному кроці (розбиття у вузлах), призводять до оптимального підсумкового рішення. У разі дерев рішень це означає, що якщо один раз був обраний атрибут, і за ним було розбито на підмножини, то алгоритм не може повернутися назад і вибрати інший атрибут, який дав би найкраще підсумкове розбиття. Тому на етапі побудови не можна сказати, чи забезпечить обраний атрибут оптимальне розбиття.

В основі більшості популярних алгоритмів навчання дерев рішень лежить принцип «поділяй і володарюй». Алгоритмічно цей принцип реалізується в такий спосіб. Нехай задано навчальну множину X, що містить n прикладів, для кожного з яких задано мітку класу $C_i (i = 1, \ldots, k)$, та m атрибутів $A_j (j = 1, \ldots, m)$ які, як передбачається, визначають належність об'єкта до того чи іншого класу. Тоді можливі три випадки:

1. Усі приклади множини X мають однакову мітку класу C_i (тобто всі навчальні приклади належать лише до одного класу). Очевидно, що навчання в цьому разі не має сенсу, оскільки всі приклади моделі, що подаються, будуть одного класу, який і «навчиться» розпізнавати модель. Саме дерево рішень у цьому разі буде листом, асоційованим із класом C_i. Практичне використання такого дерева безглузде, оскільки будь-який новий об'єкт воно відноситиме лише до цього класу.

2. Множина X взагалі не містить прикладів, тобто є порожньою множиною. У цьому разі для нього теж буде створено лист (застосовувати правило, щоб створити вузол до порожньої множини безглуздо), клас якого буде обраний з іншої множини (наприклад, клас, який найчастіше трапляється в батьківській множині).

3. Множина X містить навчальні приклади всіх класів C_k. У цьому разі потрібно розбити множину X на підмножини, асоційовані з класами. Для цього вибирається один із атрибутів A_j множини X, який містить два і більше унікальні значення $(a_1, a_2, \ldots, a_p)$, де p – кількість унікальних значень ознаки. Потім множина X розбивається на p підмножин $(X_1, X_2, \ldots, X_p)$, кожна з яких містить приклади, що мають відповідне значення атрибута. Потім вибирається наступний атрибут і повторюється розбиття. Ця процедура рекурсивно повторюватиметься до того часу, поки всі приклади в результуючих підмножинах не виявляться одного класу.

Побудова дерева рішень відбувається зверху вниз (від кореневого вузла до листя). Описана вище процедура є основою багатьох сучасних алгоритмів побудови дерев рішень. Найбільш популярні алгоритми:

ID3 (Iterative Dichotomizer 3) – алгоритм дозволяє працювати лише з дискретною цільовою змінною, тому дерева рішень, побудовані за допомогою цього алгоритму, є такими, що класифікують. Число нащадків у вузлі дерева не обмежене. Не може працювати із пропущеними даними.

C4.5 – удосконалена версія алгоритму ID3, до якої додано можливість роботи з пропущеними значеннями атрибутів.

CART (Classification and Regression Tree) – алгоритм навчання дерев рішень, що дозволяє використовувати як дискретну, так і безперервну цільову змінну, тобто вирішувати завдання як класифікації, так і регресії. Алгоритм будує дерева, які в кожному вузлі мають лише два нащадки.

Основні етапи побудови

У разі побудови дерева рішень необхідно вирішити кілька основних проблем, з кожною з яких пов'язаний відповідний крок процесу навчання:

- вибір атрибута, за яким буде проводитися розбиття в даному вузлі (атрибута розбиття);
- вибір критерію зупинення навчання;
- вибір методу відсікання гілок (спрощення);
- оцінювання точності збудованого дерева.

4.2.3 Вибір атрибуту розбиття

Під час формування правила для розбиття в черговому вузлі дерева необхідно вибрати атрибут, яким це буде зроблено. Загальне правило для цього можна сформулювати так: обраний атрибут

повинен розбити безліч спостережень у вузлі так, щоб підмножини, що одержують, містили приклади з однаковими мітками класу, або були максимально наближені до цього, тобто кількість об'єктів з інших класів («домішок») у кожному з цих множин було якнайменше. Для цього використовують різні критерії, найбільш популярними з яких стали теоретико-інформаційний й статистичний.

Теоретико-інформаційний критерій

Цей критерій ґрунтується на поняттях теорії інформації, а саме – *інформаційної ентропії*

$$S = -\sum_{i=1}^{n} \frac{N_i}{N} \log_2 \left(\frac{N_i}{N} \right), \tag{4.1}$$

де n – число класів у вихідній підмножині; N_i – кількість прикладів i-го класу; N – загальна кількість прикладів у підмножині. Отже, ентропія може розглядатися як міра неоднорідності підмножини за поданими у ньому класами. Коли класи подані в однакових частках і невизначеність класифікації найбільша, ентропія також максимальна. Якщо приклади у вузлі належать до одного класу, тобто $N = N_i$, логарифм від одиниці зануляє ентропію.

Отже, найкращим атрибутом розбиття A_j буде той, який забезпечить максимальне зниження ентропії одержаної підмножини щодо батьківської. На практиці кажуть не про ентропію, а про величину, обернену їй, яка називається інформацією. Тоді найкращим атрибутом розбиття буде той, який забезпечить максимальний приріст інформації результуючого вузла щодо вихідного:

$$Gain(A) = S_0 - \sum_{i=1}^{n} \frac{N_i}{N} S_i, \tag{4.2}$$

де S_0 – інформаційна ентропія, пов'язана з підмножиною X до розбиття; S_i – інформаційна ентропія, пов'язана з підмножинами, одержаними після розбиття за атрибутом A. Тому завдання вибору атрибуту розбиття у вузлі полягає в максимізації величини $Gain(A)$, що називається приростом інформації (від англ. Gain –

приріст, збільшення). Тому сам теоретико-інформаційний підхід відомий як *критерій приросту інформації*. Він уперше був застосований в алгоритмі ID3, а потім у C4.5 та інших алгоритмах.

Статистичний підхід

В основі статистичного підходу лежить використання індексу Джині (названий на честь італійського статистика та економіста Коррадо Джині). Статистичний зміст цього показника в тому, що він показує, наскільки часто випадково обраний приклад навчальної множини буде розпізнаний неправильно, за умови, що цільові значення в цій множині були взяті з певного статистичного розподілу. Отже, індекс Джині фактично показує відстань між двома розподілами – розподілом цільових значень і розподілом передбачень моделі. Очевидно, що чим менша ця відстань, тим краще працює модель.

Індекс Джині може бути розрахований за формулою:

$$Gini(Q) = \sum_{i=1}^{n} p_i(1 - p_i), \tag{4.3}$$

де Q – результуюча множина; n – кількість класів у ньому, $p = N_i/N$ – імовірність і-го класу (виражена як відносна частота прикладів відповідного класу). Очевидно, що цей показник змінюється від 0 до 1. Водночас він дорівнює 0, якщо всі приклади Q належать до одного класу, і дорівнює 1, коли класи подані в однакових пропорціях і є рівноймовірними. Тоді найкращим буде те розбиття, для якого значення індексу Джині буде мінімальним.

4.2.4 Критерій зупинення алгоритму

Теоретично алгоритм навчання дерева рішень буде працювати доти, поки внаслідок цього не будуть одержані абсолютно «чисті» підмножини, в кожній з яких будуть приклади одного класу. Хоча водночас буде побудовано дерево, в якому для кожного прикладу буде створено окремий лист. Очевидно, що таке дерево виявиться марним, оскільки воно буде перенавченим — кожному при-

кладу відповідатиме свій унікальний шлях у дереві, а отже, і набір правил, актуальний лише для цього прикладу.

Перенавчання у разі дерева рішень приводить до точного розпізнавання прикладів, що беруть участь у навчанні та повній неспроможності до нових даних. Крім цього, перенавчені дерева мають дуже складну структуру й тому їх складно інтерпретувати. Очевидним вирішенням проблеми є примусове зупинення побудови дерева доти, поки воно не стало перенавченим. Для цього розроблено такі підходи.

Раннє зупинення – алгоритм буде зупинено, як тільки буде досягнуто заданого значення деякого критерію, наприклад, процентної частки правильно розпізнаних прикладів. Єдиною перевагою підходу є зниження часу навчання. Головним недоліком є те, що раннє зупинення завжди приносить шкоду точності дерев.

Обмеження глибини дерева – завдання максимальної кількості розбиття в гілках, після досягнення навчання зупиняється. Цей метод також приводить до зниження точності дерева.

Завдання мінімально допустимого числа прикладів у вузлі – заборонити алгоритму створювати вузли з числом прикладів менше заданого (наприклад, 5). Це дозволить уникнути створення тривіального розбиття і відповідно незначних правил.

Усі перелічені підходи є евристичними, тобто не гарантують кращого результату чи взагалі працюють лише в якихось окремих випадках. Тому до їх використання потрібно підходити з обережністю. Будь-яких обґрунтованих рекомендацій щодо того, який метод краще працює, нині теж не існує. Тому аналітикам доводиться використовувати метод спроб й помилок. У багатьох випадках рекомендують віддавати перевагу відсіканню гілок.

4.2.5 Відсікання гілок

Як було зазначено вище, якщо «зростання» дерева не обмежити, то внаслідок цього буде побудоване складне дерево з великою

кількістю вузлів й листя. Як наслідок воно буде важко інтерпретованим. Водночас вирішальні правила таких дерев, що створюють вузли, в яких потрапляють два-три приклади, виявляються малозначними з практичного погляду.

Набагато краще мати дерево, що складається з малої кількості вузлів, яким відповідала б велика кількість прикладів з навчальної вибірки. Тому цікавий підхід, альтернативний ранньому зупиненню – побудувати всі можливі дерева і вибрати те з них, яке за розумної глибини забезпечує прийнятний рівень помилки розпізнавання, тобто знайти найбільш вигідний баланс між складністю та точністю дерева. Альтернативним підходом є так зване відсікання гілок (pruning), згідно з яким необхідно виконати таке:

1) побудувати повне дерево (щоб усе листя містило приклади одного класу);

2) визначити два показники: відносну точність моделі – відношення числа правильно розпізнаних прикладів до загального числа прикладів, та абсолютну помилку – число неправильно класифікованих прикладів;

3) видалити з дерева листя і вузли, відсікання яких не призведе до значного зменшення точності моделі або збільшення помилки.

Відсікання гілок, очевидно, проводиться в напрямку, протилежному напрямку зростання дерева, тобто знизу вгору, шляхом послідовного перетворення вузлів на листя. Перевагою відсікання гілок порівняно з раннім зупиненням є можливість пошуку оптимального співвідношення між точністю та зрозумілістю дерева. Недоліком є більший час навчання через необхідність спочатку збудувати повне дерево.

4.2.6 Вилучення правил

Іноді навіть спрощене дерево рішень все ще є надто складним для візуального сприйняття та інтерпретації. У цьому разі може виявитися корисним витягти з дерева вирішальні правила та організувати їх у набори, що описують класи.

Для одержання правил необхідно відстежити всі шляхи від кореневого вузла до листя дерева. Кожен такий шлях надає правило, що складається з безлічі умов, що є перевіркою в кожному вузлі шляху.

Візуалізація складних дерев рішень як вирішальних правил замість ієрархічної структури з вузлів і листя може бути зручнішою для візуального сприйняття.

4.2.7 Поставлення задачі

Провести класифікацію даних із використанням методу дерева прийняття рішень.
Етапи розв'язання

1. Імпортувати вибірку для проведення навчання.
2. Розділити всю вибірку на навчальну й тестову.
3. Побудувати алгоритм навчання на навчальній вибірці:
   а) порахувати ентропію за формулою (4.1) або показник Gini (4.3) для вихідної таблиці;
   б) відсортувати таблицю за першою ознакою;
   в) знайти значення ознаки для першого поділу таблиці, коли значення цільового вектора змінюється вперше;
   г) порахувати приріст інформації за такого ділення за формулою (4.2);
   д) повторити процедуру для всіх можливих ділень за першою ознакою та за всіма ознаками та знайти оптимальне перше ділення таблиці, для якого приріст інформації є максимальним;
   е) рекурсивно для кожного ділення повторити процедуру ділення;
   є) вихід з рекурсії здійснити за однієї з умов:
      - перевищення фіксованої глибини дерева;
      - мінімальної (фіксованої) кількості об'єктів у вузлі;
      - мінімального (фіксованого) значення ентропії вузла.
4. Перевірити точність роботи алгоритму на тестовій вибірці.
5. Порівняти результати з Decision tree з sklearn.
6. Оформити результати у вигляді звіту.

4.2.8 Приклад подання результату

Як результат роботи програми отримуємо словник із правилами рухів по дереву (датасет “iris”):
Мітка:

’ ’ – корінь дерева

’l’ – гілка вліво

’r’ – гілка вправо

Елементи словника:

[0] – номер ознаки для ділення;

[1] – номер рядка для ділення;

[2] – значення ознаки для ділення;

[3]:

none – не завершена гілка;

Int(i) – клас, до якого належить об’єкт.

Типовий словник:
’’: [3, 70, 1.75, None], ’l’: [3, 69, 1.65, None], ’lr’: [None, None, None, 2], ’ll’: [3, 58, 1.45, None], ’lll’: [3, 32, 0.8, None], ’llll’: [None, None, None, 0], ’lllr’: [2, 24, 5.2, None], ’lllrl’: [None, None, None, 1], ’lllrr’: [None, None, None, 2], ’llr’: [2, 7, 4.95, None], ’llrl’: [None, None, None, 1], ’llrr’: [3, 1, 1.55, None], ’llrrl’: [None, None, None, 2], ’llrrr’: [None, None, None, 1], ’r’: [0, 3, 5.85, None], ’rl’: [None, None, None, 2], ’rr’: [1, 20, 3.15, None], ’rrl’: [None, None, None, 2], ’rrr’: [3, 0, 1.9, None], ’rrrl’: [None, None, None, 1], ’rrrr’: [None, None, None, 2]
Помилка на навчальній вибірці = 0,0.
Помилка на тестовій вибірці = 0,066.

4.3 Метод k-найближчих сусідів (k-Nearest Neighbors)

Типовим представником методів класифікації є метод k-найближчих сусідів *k-nearest neighbors algorithm* (kNN). Він використовує просту логіку: об'єкт належить до того самого класу, що й більшість його найближчих сусідів.

Метод належить до класу непараметричних, тобто не вимагає припущень про те, з якого статистичного розподілу була сформована навчальна множина. Отже, класифікаційні моделі, побудовані за допомогою методу kNN, також будуть непараметричними. Це означає, що структура моделі не визначається жорстко спочатку, а визначається даними.

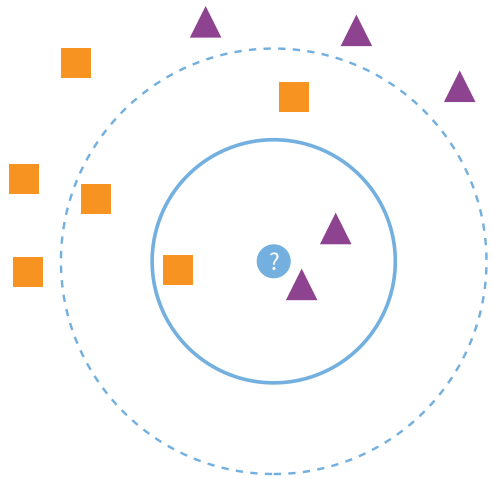
Оскільки ознаки, на основі яких проводиться класифікація, можуть мати різну фізичну природу і відповідно діапазони значень, для покращання результатів класифікації буде корисно виконати нормалізацію навчальних даних.

4.3.1 Алгоритм

Нехай є набір даних, що складається з n спостережень $X_i (i = 1, \ldots, n)$, для кожного з яких заданий клас $C_j (j = 1, \ldots, m)$. Тоді на його основі може бути сформована навчальна множина, всі приклади якої являють собою пари X_i, C_j.

Алгоритм kNN можна розділити на дві прості фази: навчання та класифікації. Під час навчання алгоритм просто запам'ятовує вектори ознак спостережень та його мітки класів (тобто приклади). Також задається параметр алгоритму k, який задає число «сусідів», які будуть використовуватися під час класифікації.

На фазі класифікації подається новий об'єкт, мітка класу якого не задана. Для нього визначаються k найближчих (у сенсі деякої метрики) попередньо класифікованих спостережень. Потім вибирається клас, якому належить більшість з k найближчих прикладів-сусідів, і до цього самого класу відноситься об'єкт, що класифікується. Роботу алгоритму можна легко пояснити за допомогою рисунка.



Робота алгоритму kNN

Кружком поданий об'єкт, який потрібно класифікувати як представника одного з двох класів «трикутники» та «квадрати». Якщо вибрати $k = 3$, то з трьох найближчих об'єктів два виявляться «трикутниками» і один «квадратом». Тому новому об'єкту буде надано клас «трикутник». Якщо поставити $k = 5$, то з п'яти «сусідів» два будуть «трикутники» і три «квадрати», внаслідок чого об'єкт, що класифікується, буде розпізнаний як «квадрат».

4.3.2 Визначення класу нового об'єкта

У найпростішому випадку клас нового об'єкта може бути визначений простим вибором класу, що найчастіше трапляється серед k прикладів. Водночас розраховується відстань від цього об'єкта до кожного об'єкта з навчальної вибірки за формулою

$$D_j = \sqrt{\sum_{i=1}^{n}(x_i - a_i)^2}, \tag{4.4}$$

де x_i – i-та ознака об'єкта, що класифікується; a_i – ознака класифікованих об'єктів.

Проте на практиці це не завжди вдале рішення, наприклад, якщо частота появи для двох або більше класів виявляється однаковою. І тут використовують деяку функцію, за допомогою якої визначається клас, яка називається функцією поєднання (combination function).

У звичайному випадку використовують так зване *просте незважене голосування (simple unweighted voting)*. Водночас передбачається, що всі k прикладів мають однакове право «голосу» незалежно від відстань до об'єкта, що класифікується.

Проте логічно припустити, що чим далі приклад розміщений від об'єкта, що класифікується, в просторі ознак, тим нижча його

значущість для визначення класу. Тому для покращання результатів класифікації вводять зважування прикладів залежно від їхньої віддаленості. І тут використовують *зважене голосування (weighted voting)*.

В основі ідеї зваженого голосування лежить введення «штрафу» для класу залежно від того, наскільки приклади, що відносяться до нього, віддалені від класифікованого об'єкта. Такий «штраф» подається як сума величин, обернених квадрату відстаней від прикладу j-го класу до класифікованого об'єкта (часто це значення називають *показником близькості*):

$$Q_j = \sum_{i=1}^{n_j} \frac{1}{D^2(x, a_{ij})}, \tag{4.5}$$

де D – оператор обчислення відстані за формулою (4.4); x – вектор ознак об'єкта, що класифікується; a_{ij} – i-й приклад j-го класу. Отже «перемагає» той клас, для якого величина Q_j виявиться найбільшою. Водночас також знижується ймовірність того, що класи отримають однакову кількість голосів.

4.3.3 Вибір значення параметра k

Вибір параметра k є важливим для одержання коректних результатів класифікації. Якщо значення параметра мале, виникає ефект перенавчання, коли рішення щодо класифікації ухвалюється з урахуванням малого числа прикладів і має низьку значущість. Це схоже на перенавчання в деревах рішень, коли в них багато правил, що належать до невеликої кількості прикладів. Якщо встановити $k = 1$, то алгоритм буде просто надавати будь-якому новому спостереженню мітку класу найближчого об'єкта.

Крім цього, необхідно враховувати, що використання невеликих значень k збільшує вплив шумів на результати класифікації, коли невеликі зміни даних призводять до великих змін у результатах класифікації. Але водночас межі класів виявляються більш вираженими (клас під час голосування перемагає з великим рахунком).

Навпаки, якщо значення параметра занадто велике, то в процесі класифікації бере участь багато об'єктів, які належать до різних класів. Така класифікація є дуже грубою і погано відбиває локальні особливості набору даних. Отже, вибір параметра k є компромісом між точністю та узагальнювальною здатністю моделі.

За великих значень параметра k зменшується шумування результатів класифікації, але знижується вираженість меж класів.

У задачах бінарної класифікації буває доцільно вибрати k як непарне число, оскільки це дозволяє уникнути рівності «голосів» щодо класу нового спостереження.

4.3.4 Значущість ознак

Для більшості методів класифікації існує проблема пов'язана з різною значущістю ознак із погляду на визначення класу об'єктів. Врахування фактора значущості ознак в алгоритмі може дозволити підвищити точність класифікації. Для цього на основі суб'єктивного або деякого формального оцінювання можна задати рівень значущості ознаки та виразити його за допомогою числового коефіцієнта (позначимо його s від англ significance — значущість), який враховується під час обчислення відстані між прикладами та об'єктом, що класифікується:

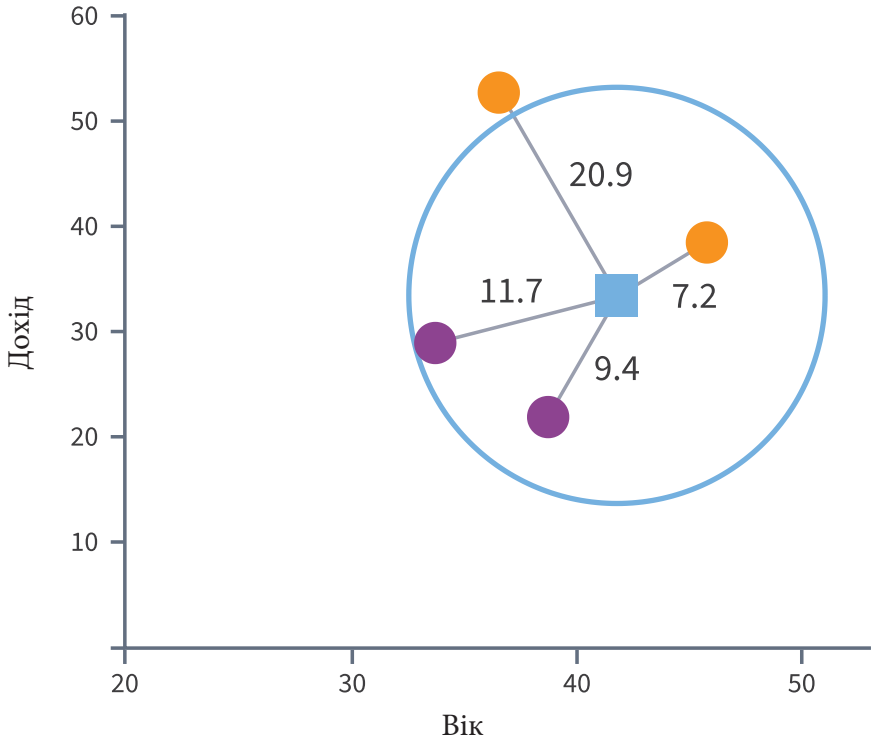
$$D_E = \sqrt{\sum_{i=1}^{n} s_i (x_i - a_i)^2}, \tag{4.6}$$

де $s_i (i = 1, \ldots, p)$ – коефіцієнт значущості для i-ї ознаки; p – кількість ознак вихідного набору даних. Такий прийом називається *розтягуванням осей* і він дозволяє збільшити або зменшити внесок ознаки в обчислення відстані від прикладу до об'єкта, що класифікується. Якщо $s_i > 1$, то завдяки відповідній ознаці відстань між прикладом об'єктом, що класифікується, зростає, і внесок у визначення класу падає, а якщо $0 < s_i < 1$, то навпаки.

4.3.5 Чисельний приклад

Розглянемо простий чисельний приклад роботи алгоритму kNN, проілюстрований на рисунку. Нехай є набір даних про пози-

чальників банку, частина з яких допустили прострочення платежу (таблиця). Ознаками є вік і середньомісячний дохід. Мітками класу в полі «Прострочено» будуть «Так» та «Ні».



X_1 – вік	X_2 – дохід	Прострочено
46	40	Ні
36	54	Ні
34	29	Так
38	23	Так

Робота алгоритму kNN

На рисунку помаранчевими кружками подані об'єкти класу «Ні», а фіолетовими класу «Так». Синім квадратом відображається об'єкт, що класифікується (новий позичальник).

Завдання полягає в тому, щоб виконати класифікацію нового позичальника, для якого $a_1 = 42$ і $a_2 = 34$ з метою оцінити можливість прострочення ним платежів.

1. Задамо значення параметра $k = 3$.

2. Розрахуємо відстань між вектором ознак об'єкта, що класифікується, і векторами навчальних прикладів за формулою (4.4) та встановимо для кожного прикладу його ранг.

X_1	X_2	Відстань	Ранг	Сусід	Прострочено
46	40	7,2	1	Так	Ні
36	54	20,9	4	Ні	Ні
34	29	9,4	2	Так	Так
38	23	11,7	3	Так	Так

3. Приберемо з розгляду приклад, який за $k = 3$ не є сусідом, і розглянемо класи, що залишилися.

X_1	X_2	Відстань	Ранг	Сусід	Прострочено
46	40	7,2	1	Так	Ні
34	29	9,4	2	Так	Так
38	23	11,7	3	Так	Так

Отже, з трьох найближчих сусідів (на рисунку розміщені всередині кола) об'єкта, що класифікується, два мають клас «Так», а один – «Ні». Тому шляхом простого незваженого голосування визначаємо його клас як «Так».

Для проведення зваженого голосування розрахуємо показник близькості за формулою (4.5):

$$Q_{NO} = \frac{1}{65} \approx 0,015; \qquad Q_{YES} = \frac{1}{98} + \frac{1}{137} \approx 0,018.$$

Шляхом зваженого голосування визначаємо клас об'єкта, що класифікується як «Так».

Із роботи класифікатора робимо висновок, що позичальник із заданими характеристиками може допустити прострочення виплати кредиту.

4.3.6 Області застосування алгоритму

Алгоритм kNN може застосовуватися практично в усіх завданнях класифікації, особливо в разі, коли оцінити параметри ймовірнісного розподілу даних складно чи неможливо. Найбільш типовими завданнями алгоритму kNN є:

- класифікація клієнтів (наприклад, за рівнем лояльності);
- медицина – класифікація пацієнтів за медичними показниками;
- маркетинг – класифікація товарів за рівнем популярності тощо.

4.3.7 Переваги та недоліки алгоритму

На завершення відзначимо переваги та недоліки алгоритму kNN. До переваг алгоритму можна віднести:

- стійкість до викидів й аномальних значень, оскільки ймовірність попадання записів, що містять їх, до k-найближчих сусідів мала. Якщо ж це сталося, то вплив на голосування (особливо зважене) швидше за все, буде незначним, а отже, малим буде і вплив на результати класифікації;
- програмна реалізація алгоритму доволі проста;
- результати роботи алгоритму легко піддаються інтерпретації;
- логіка роботи алгоритму зрозуміла експертам у різних галузях.

До недоліків алгоритму kNN можна віднести:

- цей метод не створює будь-яких моделей, які узагальнюють попередній досвід, а інтерес можуть становити й самі правила класифікації;
- під час класифікації об'єкта використовуються всі доступні дані, тому метод kNN є досить витратним, особливо в разі великих обсягів даних;
- висока трудомісткість через необхідність обчислення відстаней до всіх прикладів;
- підвищені вимоги до репрезентативності вихідних даних.

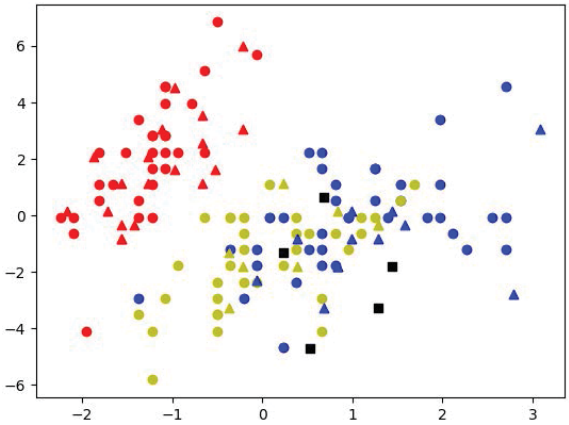
4.3.8 Поставлення задачі

Провести класифікацію даних із використанням методу k-найближчих сусідів.
Етапи розв'язання

1. Імпортувати вибірку для проведення навчання.
2. Розділити всю вибірку на дві частини (навчальну та тестову).
3. Побудувати алгоритм класифікації:
   а) зафіксувати значення параметра k;
   б) встановити значення ваг кожної з ознак таким, що дорівнює 1;
   в) для кожного об'єкта з тестової вибірки визначити, до якого класу з навчальної вибірки він належить.
4. Порахувати кількість помилок класифікації шляхом порівняння результатів класифікації з відомими значеннями класів для кожного об'єкта з тестової вибірки.
5. Провести оптимізацію параметра k та ваг кожної ознаки.
6. Порівняти результати з kNN з sklearn.
7. Оформити результати у вигляді звіту.

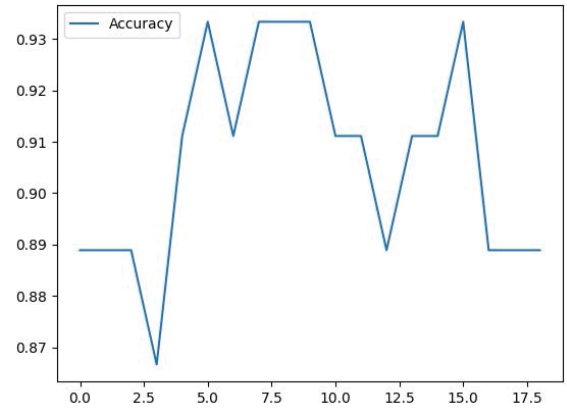
4.3.9 Приклад подання результатів

Імпортуємо вибірку з розміченими даними та ділимо її на: навчальну та тестову. Будуємо алгоритм класифікації для $k = k_0$.



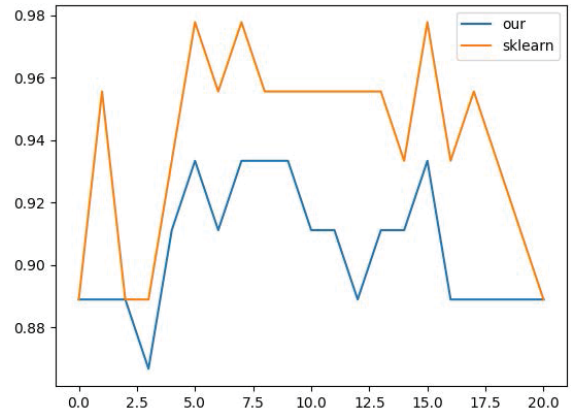
Візуалізуємо результати роботи алгоритму для $k_0 = 3$. Тут кружками різного кольору позначено об'єкти навчальної вибірки, а трикутниками того самого кольору – об'єкти тестової вибірки, що класифіковані правильно; чорні квадрати – помилки класифікації.

Вираховуємо помилку класифікації, яка становить 9 %.
Проводимо оптимізацію роботи алгоритму.



Будуємо залежність точності роботи алгоритму від значення параметра k. Робимо висновок щодо оптимального значення кількості сусідів для цієї вибірки.

Проводимо порівняння одержаних результатів з вбудованим класифікатором kNN бібліотеки sklearn.



Ілюструємо результати щодо залежності точності роботи написаного алгоритму та вбудованого алгоритму від кількості сусідів (значення параметра k) для класифікації для цієї вибірки.

4.4 Метод опорних векторів (Support Vector Machine)

Розглянемо метод опорних векторів (англ. SVM, Support Vector Machine) для задач класифікації. Подамо основну ідею алгоритму, правила виведення налаштування ваг ознак і розберемо просту реалізацію. На прикладі датасету $Iris$ буде продемонстровано роботу алгоритму з лінійно роздільними/нероздільними даними у просторі R^2. Додатково будуть озвучені плюси та мінуси алгоритму, його модифікації.

4.4.1 Завдання

Вирішуватимемо завдання бінарної (якщо класів всього два) класифікації. Спочатку алгоритм тренується на об'єктах із навчальної вибірки, яким заздалегідь відомі мітки класів. Далі вже навчений алгоритм передбачає мітку класу для кожного об'єкта з тестової вибірки. Мітки класів можуть набувати значень $Y = \{-1, +1\}$. Об'єкт – вектор із n ознаками $X = (x_1, x_2, \ldots, x_n)$ в просторі R^n. Під час навчання алгоритм повинен побудувати функцію $F(X) = Y$, яка одержує аргумент X – об'єкт з простору R^n і видає мітку класу Y. Головна мета SVM як класифікатора – знайти рівняння роздільної гіперплощини

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_0 = 0 \tag{4.7}$$

в просторі R^n, яка розділила б два класи якимось оптимальним чином.

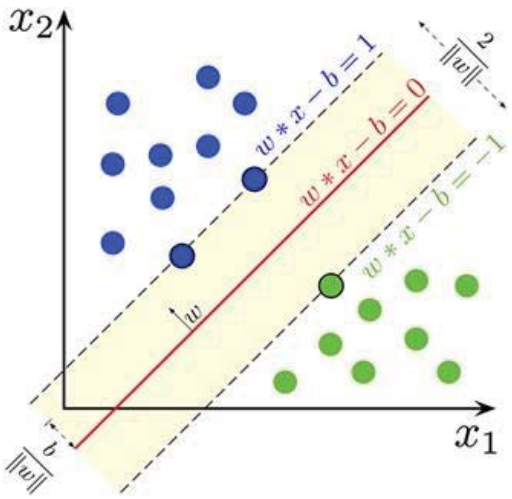
4.4.2 Загальні відомості про алгоритм

Для простоти розглянемо вибірку об'єктів із двома ознаками x_1 та x_2, які належать до двох різних класів. Загальний вид перетворення F об'єкта X на мітку класу Y:

$$F(X) = sign(w^T X - b), \tag{4.8}$$

де $w = (w_1, w_2, \ldots, w_n), b = -w_0$. Після налаштування ваг алгоритму w та b (навчання), всі об'єкти, що потрапляють по одну

сторону від побудованої гіперплощини, передбачатимуться як перший клас, а об'єкти, що потрапляють по інший бік, – другий клас.



Робота алгоритму SVM

Сині кружки – об'єкти першого класу; зелені – об'єкти другого класу; кружки з чорним контуром – опорні об'єкти; пряма $w \cdot x - b = 0$ оптимально розділяє два класи; прямі $w \cdot x - b = \pm 1$, які проходять через опорні вектори x_+ та x_- визначають ширину розділювальної смуги, як проекцію вектора $(x_+ - x_-)$ на вектор нормалі до розділювальної прямої (гіперплощини) w.

Усередині функції $sign()$ стоїть лінійна комбінація ознак об'єкта з вагами алгоритму, саме тому SVM належить до лінійних алгоритмів. Розділювальну гіперплощину можна побудувати різними способами, але в SVM ваги w і b налаштовуються так, щоб об'єкти класів лежали якнайдалі від роздільної гіперплощини. Іншими словами, алгоритм максимізує зазор (англ. margin) між гіперплощиною та об'єктами класів, які розміщені найближче до неї – опорні вектори x_+ та x_- (див. рисунок).

4.4.3 Правила налаштування ваг

Відступ (Margin)

Відступ (Margin) – характеристика, яка оцінює, наскільки об'єкт «занурений» у свій клас, наскільки типовим представником свого класу він є. Чим менше значення відступу для даного об'єкта M_i, тим ближче об'єкт X_i підходить до границі класів і тим вище є ймовірність помилки. Відступ об'єкта X_i від межі класів визначається за формулою

$$M = Y(w^T X - b). \tag{4.9}$$

Тут і далі позначатимемо скалярний добуток двох векторів як $\langle a, b \rangle$ або $a^T b$. Алгоритм припускається помилки на об'єкті тоді і лише тоді, коли відступ M негативний (коли Y і $(w^T X - b)$ різних знаків).

Якщо $M \in (0,1)$, то об'єкт потрапляє всередину розділювальної смуги. Якщо $M > 1$, то об'єкт X класифікується правильно, і розміщений на деякому віддаленні від смуги, що розділяє.

Задача – побудувати таку розділювальну гіперплощину, щоб об'єкти навчальної вибірки розміщувались на найбільшій відстані від неї. Зручно використати процедуру нормування: під час множення w та b на константу $C \neq 0$ рівняння $\langle x_i, Cw\rangle - Cb = 0$ визначає ту саму гіперплощину, що й рівняння $\langle x_i, w\rangle - b = 0$. У такому разі для зручності проводиться нормування шляхом підбору константи C такою, щоб мінімальний відступ об'єкта (відступ опорного вектора) дорівнював 1:

$$\min M_i(w,b) = 1.$$

У кожному з двох класів знайдеться хоча б один «граничний» об'єкт навчальної вибірки, відступ якого дорівнює цьому мінімуму: в іншому разі можна було б змістити гіперплощину в бік класу з більшим відступом, тим самим збільшити мінімальну відстань від гіперплощини до об'єктів навчальної вибірки.

Задача з жорстким зазором (Hard margin)

Щоб розділювальна гіперплощина розташовувалась якнайдалі від точок вибірки, ширина смуги повинна бути максимальною. Знайдемо проєкцію вектора, кінцями якого є опорні вектори різних класів, на вектор w. Ця проєкція і буде показувати ширину смуги, що розділяє два класи:

$$\frac{\langle (x_+ - x_-), w\rangle}{||w||} = \frac{(\langle x_+, w\rangle - \langle x_-, w\rangle)}{||w||} = \frac{(b+1)-(b-1)}{||w||} = \frac{2}{||w||}.$$

Ширина смуги буде максимальною за умови

$$\frac{2}{||w||} \to max \quad \Rightarrow \quad ||w|| \to min \quad \Rightarrow \quad \frac{(w^T w)}{2} \to min.$$

Тобто алгоритм правильно класифікуватиме об'єкти, якщо виконується умова:

$$Y(w^T X - b) \geq 1.$$

Якщо об'єднати два виведені вирази, то одержимо налаштування SVM із жорстким зазором (hard-margin SVM), коли жодному об'єкту не дозволяється потрапляти ні в середину смуги поділу, ні в область іншого (не свого) класу. Для класів, які лінійно розділяються, задача вирішується аналітично через теорему Куна-Такера. Задача, яку одержуємо, еквівалентна двоїстій задачі пошуку сідлової точки функції Лагранжа:

$$\begin{cases} \dfrac{w^T w}{2} \to min \\ y(w^T X - b) \geq 1 \end{cases} . \tag{4.10}$$

Для цієї задачі для ℓ об'єктів із навчальної вибірки маємо ℓ множників Лагранжа $\lambda_1, \lambda_2, \ldots, \lambda_\ell$. У такому разі функція Лагранжа відповідно до теореми Куна-Такера визначається в такий спосіб:

$$\mathcal{L}(w, b, \lambda_1, \lambda_2, \ldots, \lambda_\ell) = \frac{1}{2}||w|| - \sum_{i=1}^{\ell} \lambda_i \left\{ y_i(\langle x_i, w\rangle - b) - 1 \right\}, \tag{4.11}$$

де $||w|| \equiv \langle w, w\rangle$. Тоді оптимальні значення параметрів w і b можуть бути знайдені під час виконання таких умов:

$$\begin{cases} \dfrac{\partial \mathcal{L}(w, b, \lambda_1, \lambda_2, \ldots, \lambda_\ell)}{\partial w} = 0 \\ \dfrac{\partial \mathcal{L}(w, b, \lambda_1, \lambda_2, \ldots, \lambda_\ell)}{\partial b} = 0 \\ \lambda_i > 0 \\ \lambda_i = 0 \quad \text{або} \quad y_i(\langle x_i, w\rangle - b) - 1 = 0 \end{cases} . \tag{4.12}$$

Із першого рівняння системи (4.12) маємо:

$$\begin{aligned}
&\frac{\partial \mathcal{L}(w, b, \lambda_1, \lambda_2, \ldots, \lambda_\ell)}{\partial w} = \\
=&\frac{\partial}{\partial w}\left(\frac{1}{2}||w|| - \sum_{i=1}^{\ell} \lambda_i \left\{y_i(w^T x_i - b) - 1\right\}\right) = \\
=&\frac{1}{2}\frac{\partial}{\partial w}||w|| - \sum_{i=1}^{\ell} \lambda_i \frac{\partial}{\partial w}\left\{y_i(w^T x_i - b) - 1\right\} = \\
=&w - \sum_{i=1}^{\ell} \lambda_i y_i x_i = 0.
\end{aligned} \tag{4.13}$$

Звідки отримуємо

$$w = \sum_{i=1}^{\ell} \lambda_i y_i x_i. \tag{4.14}$$

Із другого рівняння системи (4.12) маємо:

$$\begin{aligned}
&\frac{\partial \mathcal{L}(w, b, \lambda_1, \lambda_2, \ldots, \lambda_\ell)}{\partial b} = \\
=&\frac{\partial}{\partial b}\left(\frac{1}{2}||w|| - \sum_{i=1}^{\ell} \lambda_i \left\{y_i(w^T x_i - b) - 1\right\}\right) = \\
=&-\sum_{i=1}^{\ell} \lambda_i y_i = 0.
\end{aligned} \tag{4.15}$$

Звідки отримуємо

$$\sum_{i=1}^{\ell} \lambda_i y_i = 0. \tag{4.16}$$

Підставляючи одержані вирази (4.14) і (4.16) до Лагранжіану (4.11),

одержуємо вираз

$$\begin{aligned}\mathcal{L}(w,b,\lambda_1,\lambda_2,\ldots,\lambda_\ell) &= \frac{1}{2}||w|| - \sum_{i=1}^{\ell}\lambda_i\left\{y_i(\langle x_i,w\rangle - b) - 1\right\} = \\ &= \frac{1}{2}||w|| - \underbrace{\sum_{i=1}^{\ell}\lambda_i y_i x_i^T}_{=w^T} w - \underbrace{\sum_{i=1}^{\ell}\lambda_i y_i}_{=0} b + \sum_{i=1}^{\ell}\lambda_i = \\ &= \frac{1}{2}||w|| - ||w|| + \sum_{i=1}^{\ell}\lambda_i = \\ &= \sum_{i=1}^{\ell}\lambda_i - \frac{1}{2}||w|| = \\ &= \sum_{i=1}^{\ell}\lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\lambda_i\lambda_j y_i y_j x_i^T x_j = \\ &= \tilde{\mathcal{L}}(\lambda_1,\lambda_2,\ldots,\lambda_\ell).\end{aligned} \tag{4.17}$$

Отже, ми отримуємо Лагранжіан, який визначається лише коефіцієнтами $\lambda_1,\lambda_2,\ldots,\lambda_\ell$ і замість задачі мінімізації $\frac{w^T w}{2} \to min$ переходимо до задачі максимізації Лагранжіана $\tilde{\mathcal{L}}(\lambda)$:

$$\begin{cases}\mathcal{L}(\lambda) = \sum\limits_{i=1}^{\ell}\lambda_i - \frac{1}{2}\sum\limits_{i=1}^{\ell}\sum\limits_{j=1}^{\ell}\lambda_i\lambda_j y_i y_j x_i^T x_j \to \max\limits_{\lambda} \\ \lambda_i \geq 0, \quad i = 1,\ldots,\ell \\ \sum\limits_{i=1}^{\ell}\lambda_i y_i = 0\end{cases}. \tag{4.18}$$

Далі зручно увести в розгляд матрицю

$$\underset{[\ell\times\ell]}{\mathrm{H}} \equiv \underset{[\ell\times 1]}{\mathrm{y}}\,\underset{[1\times\ell]}{\mathrm{y}^{\mathrm{T}}} \odot \underset{[\ell\times n]}{\mathrm{X}}\,\underset{[n\times\ell]}{\mathrm{X}^{\mathrm{T}}}, \tag{4.19}$$

де оператор $\odot$ позначає добуток Адамара[1]. Компоненти цієї матриці визначаються так:

$$(H)_{ij} = y_i y_j x_i^T x_j. \tag{4.20}$$

Тоді Лагранжіан $\tilde{\mathcal{L}}(\lambda)$ можна подати у більш компактному вигляді:

$$\begin{aligned}\tilde{\mathcal{L}}(\lambda) &\equiv \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i\lambda_j y_i y_j x_i^T x_j = \\ &= \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i\lambda_j \left(H\right)_{ij} = \\ &= ||\lambda|| - \frac{1}{2}\lambda^T H\lambda.\end{aligned} \tag{4.21}$$

Для знаходження оптимального рішення λ використовується метод градієнтного спуску. Відповідно до цього методу початкові значення параметрів $\lambda[0]$ встановлюються випадково, а параметри оновлюються «вгору», у напрямку вектора градієнта, оскільки це проблема максимізації:

$$\lambda^{[t+1]} = \lambda^{[t]} + \eta\frac{\partial\tilde{\mathcal{L}}(\lambda)}{\partial\lambda}, \tag{4.22}$$

де параметр t нумерує епоху навчання; η – крок навчання. В ітеративній процедурі (4.22) вектор градієнту (похідної) $\frac{\partial\tilde{\mathcal{L}}(\lambda)}{\partial\lambda}$ одержуємо у такому вигляді:

$$\frac{\partial\tilde{\mathcal{L}}(\lambda)}{\partial\lambda} = \frac{\partial}{\partial\lambda}||\lambda|| - \frac{1}{2}\frac{\partial}{\partial\lambda}\lambda^T H\lambda = \mathbf{1} - H\lambda, \tag{4.23}$$

де $\mathbf{1}$ позначає ℓ-розмірний вектор-стовпчик $\mathbf{1} = (1, 1, \ldots, 1)^T$, де всі компоненти дорівнюють 1. Отже, основна формула для одержання оптимальних значень параметрів Лагранжа (4.22) набуває вигляду:

$$\lambda^{[t+1]} = \lambda^{[t]} + \eta\left(\mathbf{1} - H\lambda^{[t]}\right). \tag{4.24}$$

[1]У математиці добуток Адамара (також відомий як добуток Шура або покомпонентний добуток) це бінарна операція над двома матрицями однакової розмірності, у результаті котрої створюється нова матриця де кожен елемент ij це добуток елементів ij початкових матриць.

Зазвичай у методі градієнтного спуску крок навчання η беруть малим, щоб задовольнити умову збіжності. У разі великих значень кроку навчання глобальний максимум за λ може бути не досягнутий через те, що з великим кроком ми будемо перестрибувати максимум функції Лагранжа $\tilde{\mathcal{L}}$ і процес навчання може відбуватися досить довго. Ітеративну процедуру пошуку оптимальних λ зупиняють або шляхом фіксації кількості ітерацій (епох навчання), або шляхом порівняння попереднього значення $\lambda^{[t]}$ та $\lambda^{[t+1]}$ з умовою мінімальної помилки, тобто коли компоненти цих двох векторів майже не відрізняються.

Останнім кроком у процедурі побудови розділювальної прямої (гіперплощини) $\langle wx\rangle - b = 0$ є знаходження оптимальних значень w та b за одержаними λ. З першого рівняння умов (4.12) ми одержали вираз для знаходження оптимального значення w за формулою (4.14)

$$w = \sum_{i=1}^{\ell} \lambda_i y_i x_i \tag{4.25}$$

Відповідно до четвертого рівняння умов (4.12) маємо

$$\lambda_i = 0 \quad \text{або} \quad y_i(\langle x_i, w\rangle - b) - 1 = 0 \tag{4.26}$$

Отже, для об'єктів навчальної вибірки маємо

$$\begin{cases} y_i(\langle x_i, w\rangle - b) - 1 = 0, & x_i \text{ – опорний вектор,} \\ y_i(\langle x_i, w\rangle - b) - 1 > 0 - - & \text{в іншому випадку.} \end{cases} \tag{4.27}$$

Для показників Лагранжа одержуємо:

$$\begin{cases} \lambda_i \neq 0 & \Leftrightarrow \quad x_i \text{ – опорний вектор,} \\ \lambda_i = 0 & \Leftrightarrow \quad \text{в іншому випадку.} \end{cases} \tag{4.28}$$

Отже, у сумі, що береться в рівнянні (4.29) ті об'єкти, які не є опорними векторами (для яких $\lambda_i = 0$), не дають жодного вкладу в суму і відповідно не впливають на визначення оптимальних значень w. Отже, сумування в (4.29) має проводитися лише за опорними векторами для збереження часу навчання:

$$w = \sum_{x_i \in S} \lambda_i y_i x_i, \tag{4.29}$$

де S задає набір опорних векторів.

За визначеними w можемо знайти значення для параметра b із умови $y_i(\langle x_i, w\rangle - b) - 1 = 0$, коли об'єкт x_i є опорним вектором:

$$b = \frac{1}{y_i} - \langle x_i, w\rangle = y_i - \langle x_i, w\rangle. \tag{4.30}$$

Тут нами було враховано, що $y_i = \{1, -1\}$. Отже, для визначення значення для параметра b достатньо лише одного опорного вектора. Проте, на практиці зазвичай використовують усі опорні вектори з множини S і знаходять середнє арифметичне

$$b = \frac{1}{S}\sum_{x_i \in S}(y_i - \langle x_i, w\rangle) \tag{4.31}$$

або медіану. Внаслідок цього, встановлені w і b дозволяють провести пряму (гіперплощину), яка оптимальним чином розділить дані на два класи.

Задача з м'яким зазором (Soft margin)

У даних можуть бути певні викиди, що призводить до нечітких меж між класами. У такому разі щоб провести класифікацію методом опорних векторів необхідно послабити обмеження, дозволяючи де-яким об'єктам попадати в середину розділювальної смуги та на «територію» іншого класу.

Отже, щоб алгоритм зміг працювати і з лінійно нероздільними даними, необхідно дозволити алгоритму припускатися помилок на навчальних об'єктах, але водночас цих помилок повинно бути якнайменше. Уведемо набір додаткових змінних $\xi_i > 0$, що характеризують величину помилки на кожному об'єкті x_i. Тоді задача оптимізації зводиться до задачі мінімізації одержаного штрафу за сумарну помилку:

$$\begin{cases} \dfrac{1}{2}(w^T w) + \alpha \displaystyle\sum_{i=1}^{\ell} \xi_i \to min, \\ y(w^T x_i - b) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{cases} \tag{4.32}$$

Еквівалентна задача безумовної мінімізації набуває вигляду:

$$\frac{1}{2}(w^T w) + \alpha \sum (1 - M_i(w,b))_+ \to min. \tag{4.33}$$

Для цієї задачі функція Лагранжа має вигляд

$$\begin{aligned}\mathcal{L}(w,b,\xi;\lambda,\theta) = \frac{1}{2}||w|| - \sum_{i=1}^{\ell} \lambda_i \left(y_i(w^T x_i - b) - 1\right) - \\ - \sum_{i=1}^{\ell} \xi_i(\lambda_i + \theta_i - \alpha),\end{aligned} \tag{4.34}$$

де $\theta = (\theta_1, \ldots, \theta_\ell)$ – вектор змінних двійкових до змінних $\xi = (\xi_1, \ldots, \xi_\ell)$. Як і в попередньому випадку жорсткого відступу, умови Куна-Такера зводять задачу до пошуку сідлової точки функції Лагранжа:

$$\begin{cases} \mathcal{L}(w,b,\xi;\lambda,\theta) \to \min\limits_{w,b,\xi} \max\limits_{\lambda,\theta}; \\ \xi_i \geq 0, \quad \lambda_i \geq 0, \quad \theta_i \geq 0, \quad i = 1, \ldots, \ell; \\ \lambda_i = 0 \quad \text{або} \quad y_i(\langle x_i, w\rangle - b) = 1 - \xi_i, \quad i = 1, \ldots, \ell; \\ \theta_i = 0 \quad \text{або} \quad \xi_i = 0, \quad i = 1, \ldots, \ell. \end{cases} \tag{4.35}$$

Останні два рядки системи (4.35) виражають умови, що доповнюють не жорсткість. Необхідними умовами існування сідлової точки функції Лагранжа (4.34) є рівність нулю похідних функції Лагранжа за змінними w, b й ξ. Взявши відповідні похідні та прирівнюючи їх до нуля, отримуємо три дуже корисних співвідношення:

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w} = w - \sum\limits_{i=1}^{\ell} \lambda_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum\limits_{i=1}^{\ell} \lambda_i y_i x_i, \\ \dfrac{\partial \mathcal{L}}{\partial b} = -\sum\limits_{i=1}^{\ell} \lambda_i y_i = 0 \quad \Rightarrow \quad \sum\limits_{i=1}^{\ell} \lambda_i y_i = 0, \\ \dfrac{\partial \mathcal{L}}{\partial \xi} = -\lambda_i - \theta_i + \alpha = 0 \quad \Rightarrow \quad \theta_i + \lambda_i = \alpha. \end{cases} \tag{4.36}$$

Необхідно зауважити, що $\theta_i > 0$, $\lambda_i > 0$, $\alpha > 0$, тому з останнього обмеження одержуємо $0 \leq \theta_i \leq \alpha$, $0 \leq \lambda_i \leq \alpha$. Отже, комбінуючи

всі одержані обмеження з систем (4.35) й (4.36) і враховуючи визначення відступу $M_i = y_i(\langle x_i, w\rangle - b)$ одержуємо такі співвідношення:

$$\begin{cases} w = \sum\limits_{i=1}^{\ell} \lambda_i y_i x_i;\ \sum\limits_{i=1}^{\ell} \lambda_i y_i = 0;\ M_i(w,b) \geq 1 - \xi_i, \\ \forall i : i = 1, \ldots, \ell : \\ \quad \xi_i \geq 0, \quad \lambda_i \geq 0, \quad \theta_i \geq 0, \quad \theta_i + \lambda_i = \alpha; \\ \quad \lambda_i M_i(w,b) = 0 \Rightarrow \lambda_i = 0 \text{ або } M_i(w,b) = 1 - \xi_i, \\ \quad \theta_i \xi_i = 0 \Rightarrow \theta_i = 0 \text{ або } \xi_i = 0. \end{cases} \tag{4.37}$$

Діапазон значень λ_i, які відповідають обмеженням на величину відступа дозволяють поділити об'єкти навчальної вибірки x_i, $i = 1, \ldots, \ell$ на три типи:

1. $\lambda_i = 0$, $\theta_i = \alpha$, $\xi_i = 0$, $M_i > 1$:
периферійні (не інформативні) об'єкти – вони розміщені в своєму класі, класифікуються правильно та не впливають на вибір розділяючої гіперплощини;

2. $0 < \lambda_i < \alpha$, $0 < \theta_i < \alpha$, $\xi_i = 0$, $M_i = 1$:
опорні граничні об'єкти – розташовані чітко на межі розділювальної смуги на стороні свого класу;

3. $\lambda_i = \alpha$, $\theta_i = 0$, $\xi_i > 0$, $M_i < 1$:
об'єкти-порушники – розміщені всередині розділювальної смуги за умови $0 < \xi_i < 1$, $0 < M_i < 1$; або не правильно класифіковані об'єкти, які розміщені на боці не свого класу за $\xi_i > 1$, $M_i < 0$. В обох цих випадках об'єкти x_i є порушниками.

Через співвідношення в третьому рядку в системі (4.36) в Лагранжіані обнуляються всі члени, які містять змінні ξ_i та θ_i, і він набуває того самого вигляду, що й у випадку лінійної роздільності для підходу з жорстким відступом. Параметри розділювальної поверхні w і b, згідно з формулами (4.36), також виражаються лише через двійкові змінні λ_i. Отже, задача знову зводиться до квадратичного програмування щодо двійкових змінних λ_i. Єдина відмінність

від лінійно розділеного випадку (з жорстким відступом) полягає в появі обмеження зверху $\lambda_i \leq \alpha$:

$$\begin{cases} \mathcal{L}(\lambda) = \sum\limits_{i=1}^{\ell} \lambda_i - \dfrac{1}{2} \sum\limits_{i=1}^{\ell} \sum\limits_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j x_i^T x_j \to \max\limits_{\lambda}, \\ 0 \leq \lambda_i \leq \alpha, \quad i = 1, \ldots, \ell, \\ \sum\limits_{i=1}^{\ell} \lambda_i y_i = 0. \end{cases} \tag{4.38}$$

Насправді для побудови SVM вирішують саме цю задачу, а не (4.18), оскільки гарантувати лінійну роздільність вибірки в загальному випадку не є можливим.

Отже, різниця між двома підходами полягає в тому, що в разі м'якого відступу ненульовими значеннями λ_i характеризуються не лише опорні об'єкти, а й об'єкти-порушники. У певному сенсі це недолік SVM, оскільки порушниками часто є шумові викиди, і побудоване на них вирішальне правило насправді спирається на шум.

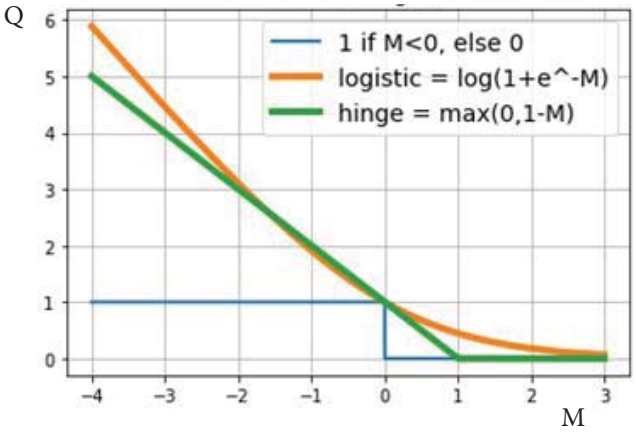
Функція помилок. Для розв'язання задачі (4.38) зручно увести функцію помилок Q (loss function). Кількість помилок алгоритму визначається з умови $M < 0$. Тоді штраф для всіх об'єктів дорівнюватиме сумі штрафів для кожного об'єкта x_i: $Q = \sum\limits_{i=1}^{\ell} Q(M_i)$. Одним із варіантів функції втрат може бути кусково-лінійна функція

$$Q = \begin{cases} 1, & \text{якщо } M_i < 0, \\ 0, & \text{якщо } M_i \geq 0. \end{cases}$$

У такому разі всі помилки на всіх об'єктах-порушниках будуть однакові не залежно від величини відступу.

На практиці зазвичай ураховують чутливість штрафу до величини помилки (чим сильніше M «йде в мінус» – тим більше штраф) і уводять штраф за наближення об'єкта до межі класів. У SVM зазвичай використовують функцію, яка обмежує граничну функцію помилки:

$$Q \leq \alpha \sum_{i=1}^{\ell} (1 - M_i)_+ = \alpha \sum_{i=1}^{\ell} max(0, 1 - M_i).$$



Функція втрат (Loss function)

Оскільки для n-розмірного простору ознак існує $n+1$ параметрів, що визначають роздільну площину: n параметрів w та один параметр b, то для подальшого розгляду зручно увести ще одну ознаку $x_i^{[n+1]}$ і прирівняти її значення до 1, поклавши при цьому $w_{n+1} = -b$. Це дозволить зробити такий перехід:

$$\begin{aligned} &(\langle x_i, w\rangle - b)_{i=1,\ldots,\ell} \quad \to \quad (\langle x_i, w\rangle)_{i=1,\ldots,\ell+1}, \\ &M_i(w,b) = y_i(\langle x_i, w\rangle - b) \to M_i(w) = y_i(\langle x_i, w\rangle). \end{aligned} \tag{4.39}$$

Під час додавання до виразу штрафу доданка $(w^T w)/2$ одержуємо класичний вигляд функціоналу якості для методу опорних векторів із м'яким відступом:

$$\mathcal{F} = \frac{1}{2}(w^T w) + \alpha \frac{1}{\ell} \sum_{i=1}^{\ell} (1 - M_i(w))_+ \to \min_w . \tag{4.40}$$

Мінімізація функціоналу якості $\mathcal{F}$ проводиться за допомогою градієнтного спуску. Правила зміни ваг набуває вигляду:

$$w = w - \eta \nabla Q,$$

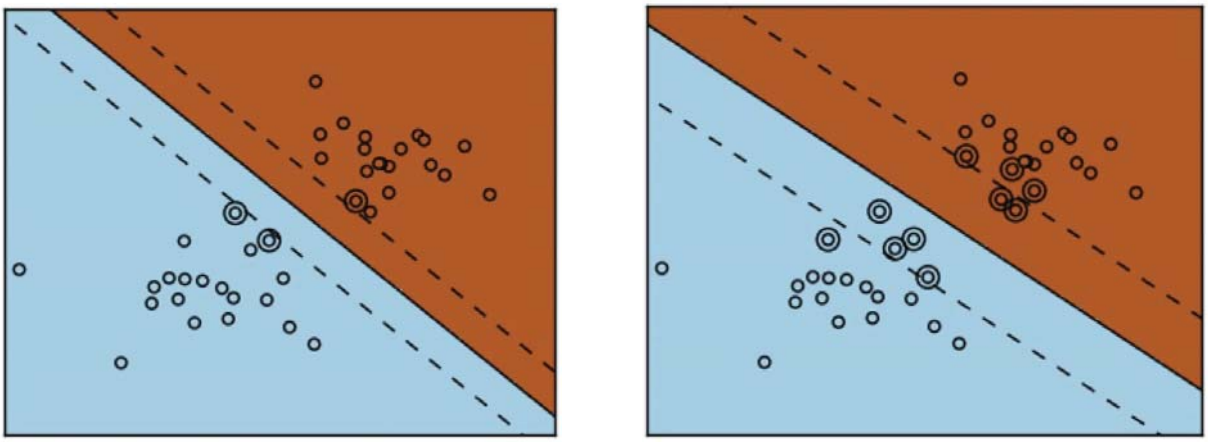
де η – крок спуску (крок навчання). Похідна від функції втрат дає:

$$\nabla Q(w) = \frac{dQ}{dw} = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} w, \text{ якщо } max(1 - y_i \langle w_i, x_i\rangle)_+ = 0, \\ w - \alpha y_i x_i - - \text{ в іншому разі.} \end{cases}$$

Цей алгоритм дозволить провести класифікацію об'єктів для лінійно не роздільної вибірки.

Порівнюючи задачі оптимізації для жорсткого відступу (4.18) та м'якого відступу (4.38) бачимо, що відмінність полягає в тому, що в останньому випадку параметри λ обмежені зверху параметром регуляризації – константою α. Тому, логічно припустити, що під час спрямування параметра α в бік нескінченності задача з м'яким відступом переходить у задачу з жорстким відступом.

Отже, під час збільшення параметра регуляризації α кількість порушників – об'єктів, які попадають усередину розділювальної смуги та в область не свого класу, буде зменшуватися за рахунок звуження самої смуги. Типові приклади для великого значення параметра α (зліва) та малого значення параметра α (справа) наведено на рисунку.



Вплив параметра регуляризації: велике α (зліва) та мале α (справа)

Константу α зазвичай вибирають за критерієм ковзного контролю. Це трудоємний спосіб, оскільки завдання доводиться вирішувати заново за кожного значення α.

Якщо є підстави вважати, що вибірка майже лінійно розділена, і лише об'єкти-викиди класифікуються неправильно, то можна застосувати фільтрацію викидків. Спочатку завдання вирішується за деякого α, і з вибірки видаляється невелика частка об'єктів, що мають найбільшу величину помилки ξ_i. Після цього завдання вирішується заново за усіченою вибіркою. Можливо, доведеться зробити кілька таких ітерацій, поки об'єкти, що залишилися, не виявляться лінійно роздільними.

4.4.4 Плюси, мінуси та модифікації

Плюси:

- добре працює з простором ознак великого розміру;
- добре працює з даними невеликого обсягу;
- алгоритм максимізує смугу, що розділяє класи, яка дозволяє зменшити кількість помилок класифікації;
- оскільки алгоритм зводиться до розв'язання задачі квадратичного програмування в опуклій області, то таке завдання завжди має єдине рішення (роздільна гіперплощина з певними гіперпараметрами алгоритму завжди одна).

Мінуси:

- довгий час навчання (для великих наборів даних);
- нестійкість до шуму: викиди в навчальних даних стають опорними об'єктами-порушниками і безпосередньо впливають на побудову роздільної гіперплощини;
- не описані загальні методи побудови ядер і спрямовуючих просторів, що найбільш підходять для конкретного завдання в разі лінійної нероздільності класів.

Модифікації алгоритму:

- Метод релевантних векторів (Revance Vector Machine, RVM);
- 1-norm SVM (LASSO SVM);
- Doubly Regularized SVM (ElasticNet SVM);
- Support Features Machine (SFM);
- Relevance Features Machine (RFM);

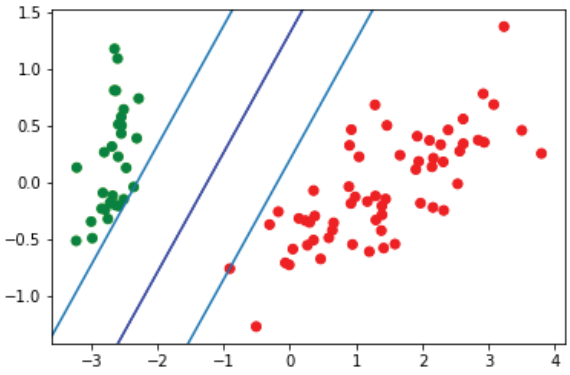
4.4.5 Постановлення задачі

Провести класифікацію даних із використанням методу опорних векторів.
Етапи розв'язання

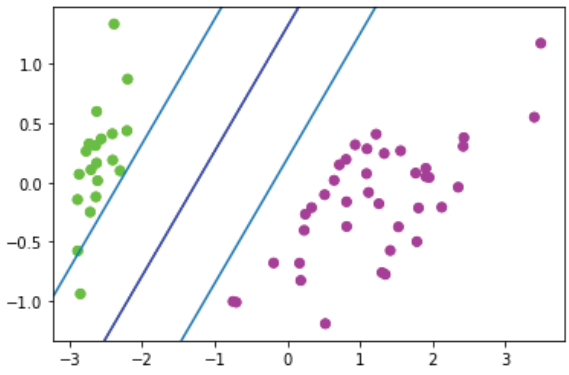
1. Імпортувати вибірку для проведення навчання.
2. Розділити всю вибірку на навчальну та тестову.
3. Провести підготовку даних до класифікації:
   а) визначити дві головних ознаки, за якими буде проводитися класифікація;
   б) додати ще один стовпчик до матриці ознак і встановити всі значення такими, що дорівнюють 1;
   в) встановити значення цільового вектора $+1$ та -1.
4. Побудувати алгоритм навчання на навчальній вибірці:
   а) встановити значення кроку навчання η, коефіцієнта регуляризації α;
   б) визначити умову завершення навчання (встановити кількість епох навчання або мінімально допустиму точність алгоритму);
   в) залежно від величини зазору змінювати ваги за допомогою градієнта функції втрат;
   г) рахувати кількість помилок (неправильно класифікованих об'єктів) у процесі навчання.
5. Графічно подати результат класифікації для навчальної вибірки та залежність помилок від номеру епохи навчання.
6. Перевірити точність роботи алгоритму на тестовій вибірці.
7. Графічно подати результат класифікації для тестової вибірки.
8. Порівняти результати з SVM з sklearn.
9. Оформити результати у вигляді звіту.

4.4.6 Приклад подання результатів

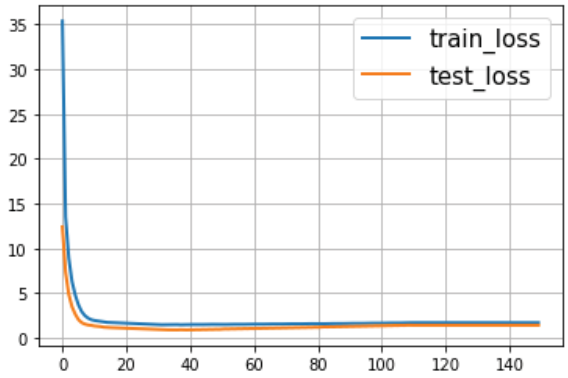
Класифікація лінійно роздільної вибірки (Hard margin)



Навчальна вибірка

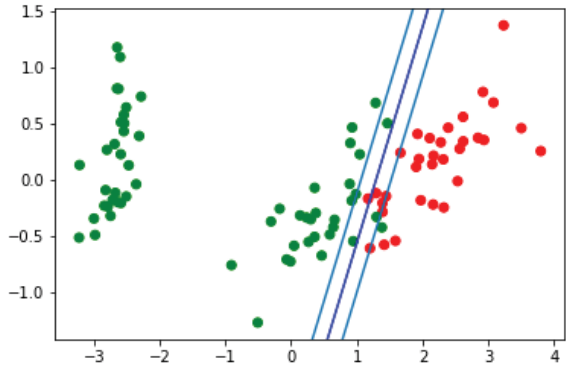


Тестова вибірка

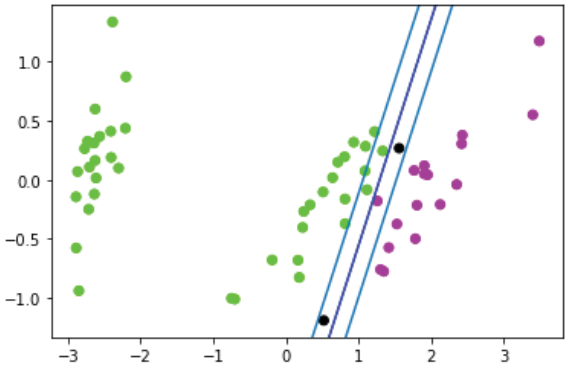


Помилки класифікації

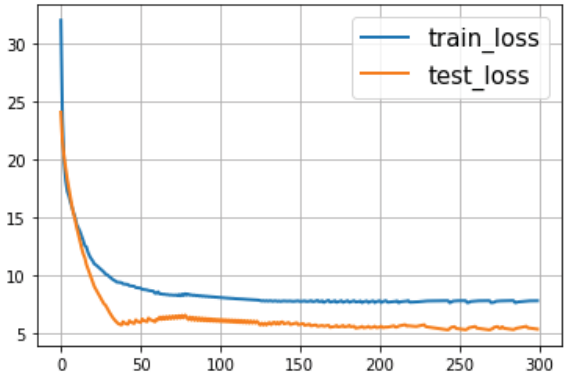
Класифікація лінійно не роздільної вибірки (Soft margin)



Навчальна вибірка



Тестова вибірка



Помилки класифікації

4.5 Логістична регресія

Логістична регресія – корисний класичний інструмент для розв'язання задачі регресії та класифікації. Вона набула поширення у скорингу для розрахунку рейтингу позичальників та управління кредитними ризиками.

Логістична регресія – це різновид множинної регресії, загальне призначення якої полягає в аналізі зв'язку між декількома незалежними змінними (названими також регресорами або предикторами) та залежною змінною. Бінарна логістична регресія застосовується в разі, коли залежна змінна є бінарною (тобто може набувати лише двох значень). За допомогою логістичної регресії можна оцінювати ймовірність того, що подія настане для конкретного випробуваного (хворий/здоровий, повернення кредиту/дефолт тощо).

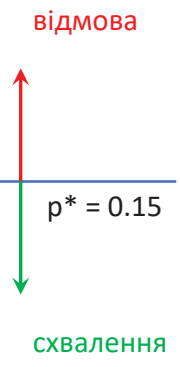
Розглянемо задачу бінарної класифікації, причому мітки цільового класу позначимо «+1» (позитивні приклади) та «-1» (негативні приклади). Логістична регресія є окремим випадком лінійного класифікатора, але вона має гарне «вміння» – прогнозувати ймовірність p_+ віднесення об'єкта $\vec{x_i}$ до класу «+»:

$$p_+ = P\left(y_i = 1 \mid \vec{x_i}, \vec{w}\right).$$

Прогнозування не просто відповіді («+1» або «-1»), а саме ймовірності віднесення до класу «+1» у багатьох завданнях є дуже важливою бізнес-вимогою.

Наприклад, завдання кредитного скорингу, де традиційно застосовується логістична регресія, часто прогнозують імовірність неповернення кредиту (p_+). Клієнтів, які звернулися за кредитом, сортують за цією передбачуваною ймовірністю (за спаданням), і виходить скорингова карта, рейтинг клієнтів від «поганих» до «хороших». Банк вибирає собі поріг $p*$ передбаченої ймо-

Клієнт	Імовірність не повернення кредиту
Клієнт 1	0.78
Клієнт 2	0.56
Клієнт 3	0.43
Клієнт 4	0.13
Клієнт 5	0.09
Клієнт 6	0.05
Клієнт 7	0.02



вірності неповернення кредиту (на рисунку $0,15$) і з цього значення не видає кредит. Більше того, можна помножити передбачену ймовірність на видану суму і одержати математичне очкування втрат із клієнта, що теж буде гарною бізнес-метрикою.

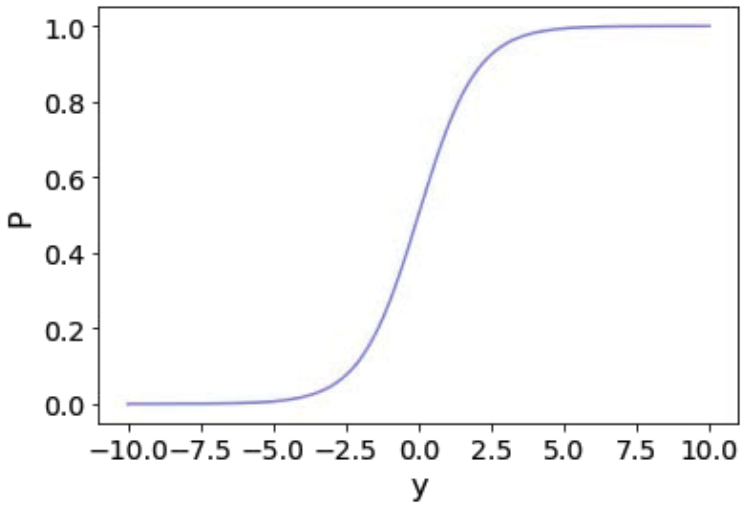
Отже, є задача прогнозувати ймовірність $p_+ \in [0,1]$. У множинній лінійній регресії передбачається, що залежна змінна є лінійною функцією незалежних змінних, тобто:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n, \tag{4.41}$$

для якої лінійний прогноз одержується за допомогою методу найменших квадратів: $\hat{y}(\vec{x}) = \vec{w}^T \vec{x} \in \mathbb{R}$. Для задачі класифікації нам необхідно одержати як відповіді ймовірність, межі якої $[0,1]$ за будь-яких значень незалежних змінних. Це досягається застосуванням деякої функції $f : \mathbb{R} \to [0,1]$, а саме:

$$P(y) = \frac{1}{1 + \ exp^{-\hat{y}}}, \tag{4.42}$$

де P – імовірність того, що відбудеться певна подія.



Позначимо $P(X)$ імовірністю події, що відбувається, X. Тоді відношення ймовірностей $OR(X)$ визначається з $\frac{P(X)}{1-P(X)}$, а це відношення ймовірностей того, чи відбудеться подія чи ні. Очевидно, що ймовірність і відношення шансів містять однакову інформацію. Але в той час, як $P(X)$ приймає значення в інтервалі від 0 до 1, $OR(X)$ приймає значення в інтервалі від 0 до ∞. Якщо обчислити логарифм $OR(X)$ (логарифм шансів, або логарифм відношення

ймовірностей), то легко помітити, що $\log OR(X) \in \mathbb{R}$. Кроки прогнозу логістичної регресії $p_+ = P\left(y_i = 1 \mid \vec{x_i}, \vec{w}\right)$ (за умови, що модель вже навчена і ваги $\vec{w}$ відомі).

Крок 1. Обчислити значення $w_0 + w_1x_1 + w_2x_2 + \ldots = \vec{w}^T\vec{x}$. Рівняння $\vec{w}^T\vec{x} = 0$ задає гіперплощину, що розділяє приклади на 2 класи.

Крок 2. Обчислити логарифм відношення шансів: $\log(OR_+) = \vec{w}^T\vec{x}$.

Крок 3. Маючи прогноз шансів на віднесення до класу «+» OR_+, обчислити p_+ за допомогою простої залежності:

$$p_+ = \frac{OR_+}{1 + OR_+} = \frac{\exp^{\vec{w}^T\vec{x}}}{1 + \ exp^{\vec{w}^T\vec{x}}} = \frac{1}{1 + \exp^{-\vec{w}^T\vec{x}}} = P(\vec{w}^T\vec{x}).$$

Отже, логістична регресія прогнозує ймовірність віднесення об'єкта до класу «+» (за умови, що ми знаємо його ознаки та ваги моделі) як сигмоїд-перетворення лінійної комбінації вектора ваг моделі та вектора ознак об'єкта:

$$p_+(x_i) = P\left(y_i = 1 \mid \vec{x_i}, \vec{w}\right) = \sigma(\vec{w}^T\vec{x_i}), \tag{4.43}$$

для класу «-» аналогічна ймовірність:

$$p_-(\vec{x_i}) = P\left(y_i = -1 \mid \vec{x_i}, \vec{w}\right) = 1 - \sigma(\vec{w}^T\vec{x_i}) = \sigma(-\vec{w}^T\vec{x_i}). \tag{4.44}$$

Для розв'язання задачі оптимізації – мінімізації помилок класифікації – використовується принцип максимальної правдоподібності.

4.5.1 Принцип максимальної правдоподібності

Для загальної ймовірності належності об'єкта до якогось із двох класів маємо вираз

$$P\left(y = y_i \mid \vec{x_i}, \vec{w}\right) = \sigma(y_i\vec{w}^T\vec{x_i}). \tag{4.45}$$

Вираз $M(\vec{x_i}) = y_i\vec{w}^T\vec{x_i}$ називають відступом (margin) класифікації на об'єкті $\vec{x_i}$. Якщо він позитивний, модель не помиляється на об'єкті $\vec{x_i}$, якщо ж негативний – отже, клас для $\vec{x_i}$ спрогнозований

неправильно. Зауважимо, що відступ визначений для об'єктів саме навчальної вибірки, яким відомі реальні мітки цільового класу y_i.

Тепер розпишемо правдоподібність вибірки, а саме, можливість спостерігати цей вектор $\vec{y}$ у вибірці X. Робимо сильне припущення: об'єкти приходять незалежно з одного розподілу. Основу методу максимальної правдоподібності становить функція правдоподібності (likehood function) $\mathcal{L}(X, \vec{y}, \vec{w})$, що виражає густину ймовірності (імовірність) спільної появи результатів вибірки

$$\mathcal{L}(X, \vec{y}, \vec{w}) = P\left(\vec{y} \mid X, \vec{w}\right) = \prod_{i=1}^{\ell} P\left(y = y_i \mid \vec{x_i}, \vec{w}\right), \tag{4.46}$$

де ℓ – довжина вибірки X. Згідно з методом максимальної правдоподібності за оцінку невідомого параметра беруть таке значення $\vec{w}$, яке максимізує функцію $\mathcal{L}$.

Знаходження оцінки спрощується, якщо максимізувати не саму функцію $\mathcal{L}$, а логарифм $\ln(\mathcal{L})$ (суму оптимізувати набагато простіше, ніж добуток), оскільки максимум обох функцій досягається за того самого значення $\vec{w}$:

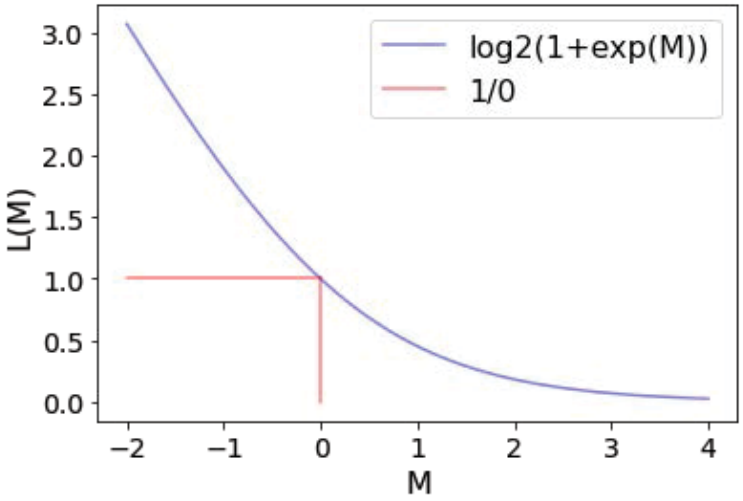
$$\begin{aligned}
\log P\left(\vec{y} \mid X, \vec{w}\right) &= \log \prod_{i=1}^{\ell} P\left(y = y_i \mid \vec{x_i}, \vec{w}\right) = \\
&= \log \prod_{i=1}^{\ell} \sigma(y_i \vec{w}^T \vec{x_i}) = \\
&= \sum_{i=1}^{\ell} \log \sigma(y_i \vec{w}^T \vec{x_i}) = \\
&= \sum_{i=1}^{\ell} \log \frac{1}{1 + \exp^{-y_i \vec{w}^T \vec{x_i}}} = \\
&= -\sum_{i=1}^{\ell} \log(1 + \exp^{-y_i \vec{w}^T \vec{x_i}}).
\end{aligned} \tag{4.47}$$

Отже, у цьому разі принцип максимізації правдоподібності приво-

дить до мінімізації виразу

$$\mathcal{L}_{log}(X, \vec{y}, \vec{w}) = \sum_{i=1}^{\ell} \log(1 + \exp^{-y_i \vec{w}^T \vec{x_i}}). \tag{4.48}$$

Це логістична функція втрат, підсумована за всіма об'єктами навчальної вибірки: $L(M) = \log(1 + \exp^{-M})$.



На графіку зображено її залежність, а також графік 1/0 функції втрат (zero-one loss), яка просто штрафує модель на 1 за помилку на кожному об'єкті (відступ негативний): $L_{1/0}(M) = [M < 0]$. Рисунок відбиває загальну ідею, що у завдання класифікації, не вміючи безпосередньо мінімізувати кількість помилок (принаймні, градієнтними методами це не зробити – похідна 1/0 функцій втрат у нулі прямує до нескінченності) ми мінімізуємо деяку її верхню оцінку. У цьому разі – це логістична функція втрат (де логарифм двійковий, але це не є принциповим), і справедливо

$$\begin{aligned} \mathcal{L}_{1/0}(X, \vec{y}, \vec{w}) = &\sum_{i=1}^{\ell} [M(\vec{x_i}) < 0] \leq \\ \leq &\sum_{i=1}^{\ell} \log(1 + \exp^{-y_i \vec{w}^T \vec{x_i}}) = \\ = &\mathcal{L}_{log}(X, \vec{y}, \vec{w}), \end{aligned} \tag{4.49}$$

де $\mathcal{L}_{1/0}(X, \vec{y}, \vec{w})$ — просто число помилок логістичної регресії з вагами $\vec{w}$ на вибірці $(X, \vec{y})$. Тобто зменшуючи верхню оцінку $\mathcal{L}_{log}$ на

кількість помилок класифікації, ми таким чином сподіваємося зменшити й кількість помилок.

4.5.2 L_2-регуляризація логістичних втрат

L_2-регуляризація логістичної регресії влаштована майже так само, як і у разі з гребеневою (Ridge регресією). Замість функціоналу $\mathcal{L}_{log}(X, \vec{y}, \vec{w})$ мінімізується такий:

$$J(X, \vec{y}, \vec{w}) = \mathcal{L}_{log}(X, \vec{y}, \vec{w}) + \lambda |\vec{w}|^2.$$

У разі логістичної регресії прийнято запровадження зворотного коефіцієнта регуляризації $C = \frac{1}{\lambda}$. І тоді вирішенням завдання буде

$$\hat{w} = \arg\min_{\vec{w}} J(X, \vec{y}, \vec{w}) = \arg\min_{\vec{w}} \left(C \sum_{i=1}^{\ell} \log(1 + \exp^{-y_i \vec{w}^T \vec{x_i}}) + |\vec{w}|^2 \right).$$

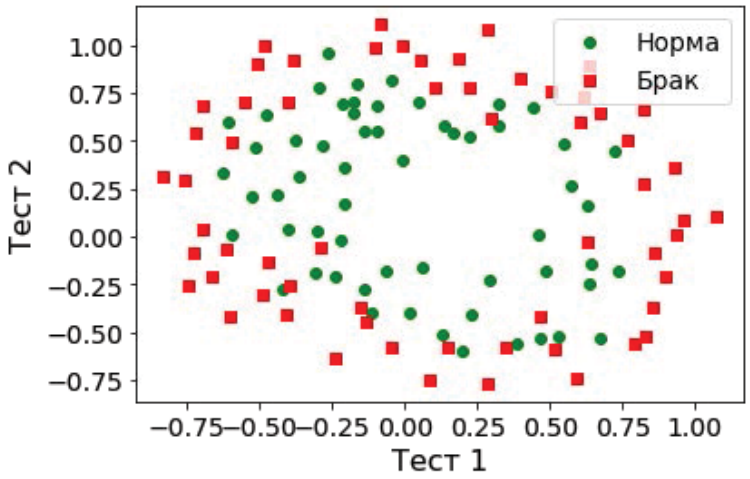
Далі розглянемо приклад, що дозволяє інтуїтивно зрозуміти один із сенсів регуляризації.

Наочний приклад регуляризації логістичної регресії

Як приклад розглянемо задачу бінарної класифікації вибірки з двома ознаками та подивимося, як регуляризація впливає на якість класифікації. Будемо використовувати логістичну регресію з поліноміальними ознаками та варіюватимемо параметр регуляризації C. Розглянемо набір даних щодо тестування мікрочипів, у якому для 118 мікрочипів (об'єкти) зазначено результати двох тестів з контролю якості (дві числові ознаки) і сказано, чи мікрочип пустили у виробництво («Норма»), чи він бракований («Брак»); ознаки вже центровані.

- Спочатку подивимося, як регуляризація впливає на роздільну межу класифікатора, інтуїтивно розпізнаємо перенавчання та недонавчання.
- Потім чисельно встановимо близький до оптимального параметра регуляризації за допомогою крос-валідації.

Вибірка має такий вигляд: зелені кружечки – чипи випущені у виробництво; червоні квадрати – брак.



Кроки реалізації

1. Визначаємо функцію для відображення роздільної кривої класифікатора. Поліноміальними ознаками до d для двох змінних x_1 і x_2 будемо вважати такі:

$$\{x_1^d, x_1^{d-1}x_2, \dots x_2^d\} = \{x_1^i x_2^j\}_{i+j\leq d, i,j\in\mathbb{N}}.$$

   Наприклад, для $d = 3$ це будуть такі ознаки:

$$1,\ x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2,\ x_1^3,\ x_1^2x_2,\ x_1x_2^2,\ x_2^3.$$

2. Додамо до матриці ознак X поліноміальні ознаки до ступеня 7.

3. Зафіксуємо параметр регуляризації C.

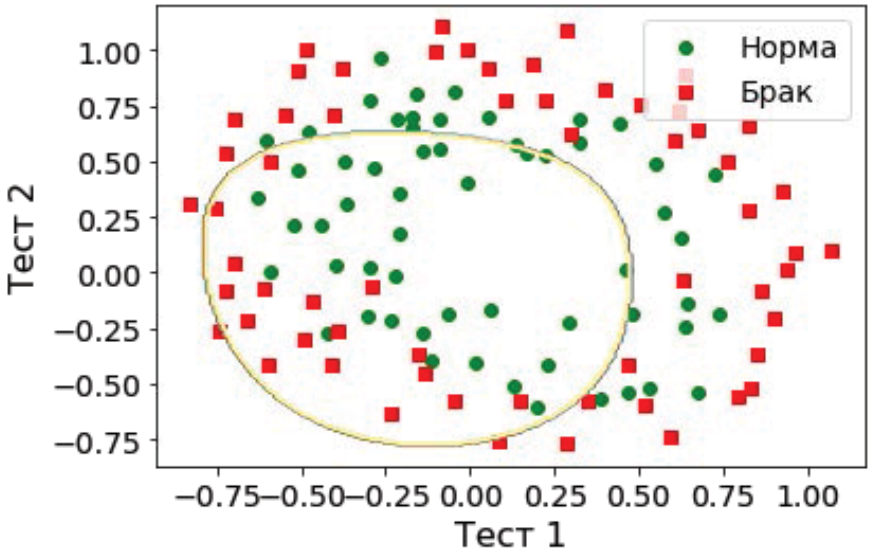
4. Навчимо логістичну регресію.

5. Зобразимо межу, що розділяє класи.

6. Перевіримо частку правильних відповідей класифікатора на навчальній вибірці.

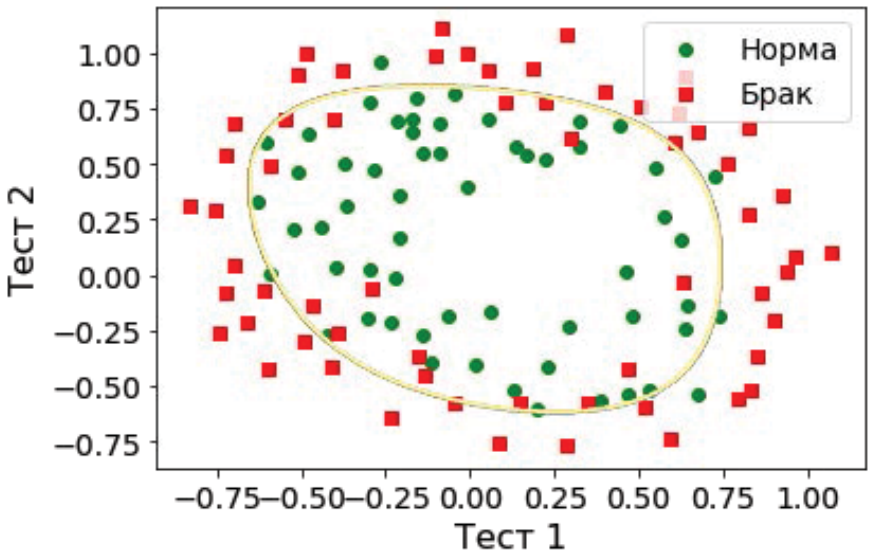
Проміжні результати

- Сильна регуляризація з $C = 10^{-2}$.



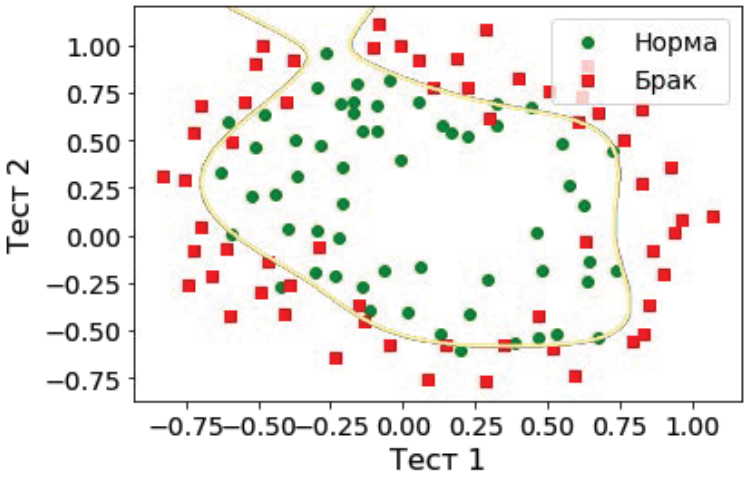
Частка правильних відповідей класифікатора на навчальній вибірці дорівнює 0,627.

- Середня регуляризація з $C = 1$.



Частка правильних відповідей класифікатора на навчальній вибірці дорівнює 0,831.

- Слабка регуляризація з $C = 10^4$.



Частка правильних відповідей класифікатора на навчальній вибірці дорівнює 0,873.

Бачимо, що у разі сильної регуляризації ($C = 10^{-2} \to \lambda = 10^2$) модель «недонавчилась», про що свідчить доволі не висока якість класифікації (62,7%). Під час зменшення регуляризації ($C = 1 \to \lambda = 1$) у вирішенні значення ваги логістичної регресії можуть виявитися більше (за модулем), ніж у випадку сильної регуляризації, що приводить до покращання якості класифікації (83,1%). Під час збільшення параметра C до 10 тисяч – істотне зменшення регуляризації – її явно недостатньо, і ми спостерігаємо перенавчання. Водночас якість класифікації збільшилася лише на 4,2 %, але розділювальна крива сильно прив'язана до навчальної вибірки і може погано класифікувати об'єкти тестової вибірки.

Щоб обговорити результати, перепишемо формулу для функціонала, який оптимізується в логістичній регресії, у такому вигляді:

$$J(X, y, w) = \mathcal{L} + \frac{1}{C}||w||^2,$$

де $\mathcal{L}$ – логістична функція втрат, підсумована за всією вибіркою; C – зворотний коефіцієнт регуляризації.

Проміжні висновки

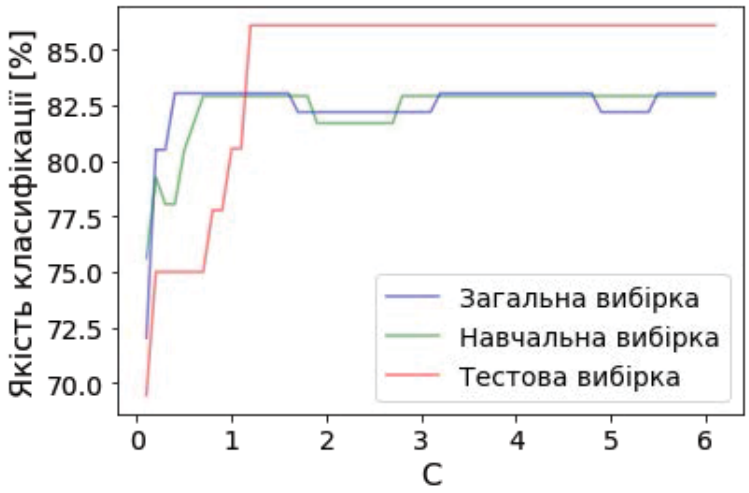
- Якщо регуляризація занадто сильна (малі значення C), то розв'язанням задачі мінімізації логістичної функції втрат мо-

же виявитися таким, що багато вагових коефіцієнтів занулилися або стали занадто малими. У такому разі модель недостатньо «штрафується» за помилки (тобто у функціоналі J «переважує» сума квадратів вагових коефіцієнтів, а помилка $\mathcal{L}$ може бути відносно великою). У цьому разі модель виявиться недонавченою (випадок за $C = 10^{-2}$).

- Навпаки, якщо регуляризація дуже слабка (великі значення C), то розв'язанням задачі оптимізації може стати вектор w із великими за модулем компонентами. У такому разі більший внесок у функціонал, що оптимізується, J має $\mathcal{L}$. Така модель занадто «боїться» помилитися на об'єктах навчальної вибірки, тому виявиться перенавченою (випадок за $C = 10^4$).

Налаштування параметра регуляризації

Параметр C – гіперпараметр моделі, що налаштовується під час проведення навчання на конкретній вибірці шляхом крос-валідації. Для знаходження оптимального значення параметра регуляризації C порівняємо результати класифікації на навчальній та тестовій вибірках для різних значень C. Для цього розділимо вибірку на навчальну та тестову у відношенні 70 %/30 %. Одержуємо результати для трьох випадків: загальної, навчальної та тестової вибірок.



З одержаних результатів робимо висновок про те, що за значення оберненого параметра регуляризації $C \simeq 1,5$ якість класифікації є максимальною для всіх трьох вибірок: загальної, навчальної та тестової.

4.5.3 Поставлення задачі

Провести класифікацію даних із використанням методу логістичної регресії.
Етапи розв'язання

I Випадок лінійно-роздільної вибірки

1. Імпортувати дані, що можуть бути лінійно розділені.
2. Поділити дані на навчальну та тестову вибірки.
3. Навчити модель логістичної регресії на навчальній вибірці:
   - встановити значення кроку навчання η;
   - визначити умову завершення навчання (встановити кількість епох навчання або мінімально допустиму точність алгоритму);
   - змінювати ваги за допомогою методу градієнтного спуску;
   - рахувати кількість помилок (неправильно класифікованих об'єктів) у процесі навчання.
4. Перевірити якість класифікації на тестовій вибірці.
5. Графічно подати результат класифікації для навчальної вибірки та залежність помилок від номеру епохи навчання
6. Перевірити точність роботи алгоритму на тестовій вибірці.
7. Графічно подати результат класифікації для тестової вибірки.
8. Порівняти результати з вбудованим алгоритмом з sklearn.

II Випадок лінійно не роздільної вибірки

1. Імпортуємо дані, що не можуть бути лінійно розділені.
2. Поділити дані на навчальну та тестову вибірки.
3. Навчити модель логістичної регресії з регуляризацією L_2 на навчальній вибірці:
   - встановити значення кроку навчання η та значення параметра регуляризації $C = 1/\lambda$;
   - визначити умову завершення навчання (встановити кількість епох навчання або мінімально допустиму точність алгоритму);
   - змінювати ваги за допомогою методу градієнтного спуску;
   - рахувати кількість помилок (неправильно класифікованих об'єктів) у процесі навчання.
4. Провести оптимізацію параметра регуляризації.
5. Для оптимального значення C побудувати роздільну криву на навчальній вибірці.
6. Перевірити якість класифікації на тестовій вибірці.
7. Графічно подати результат класифікації для навчальної вибірки та залежність помилок від номеру епохи навчання.
8. Перевірити точність роботи алгоритму на тестовій вибірці.
9. Графічно подати результат класифікації для тестової вибірки.
10. Порівняти результати з вбудованим алгоритмом з sklearn.

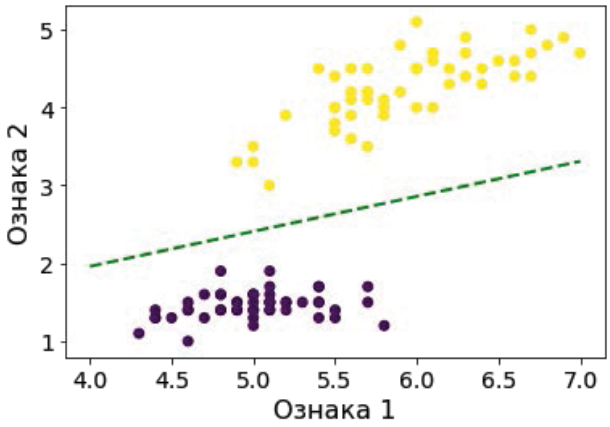
III Оформити результати для обох випадків у вигляді звіту.

4.5.4 Приклад подання результатів

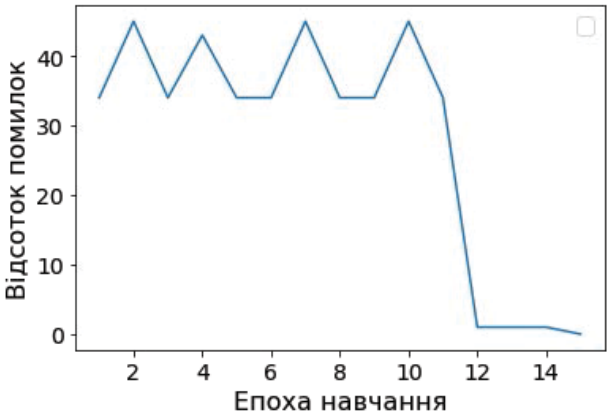
Класифікація лінійно роздільної вибірки

Класи що не перекриваються

Вибірка та розділювальна пряма

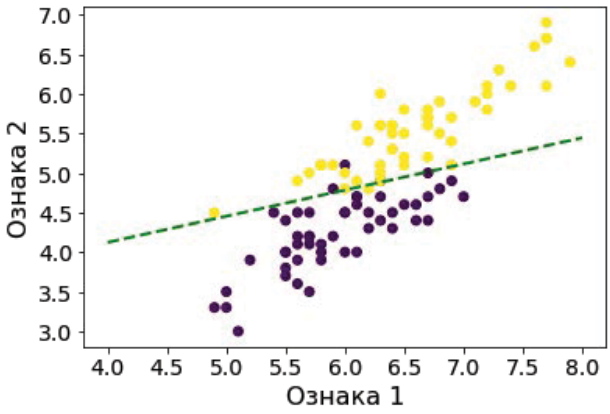


Помилка від епохи навчання

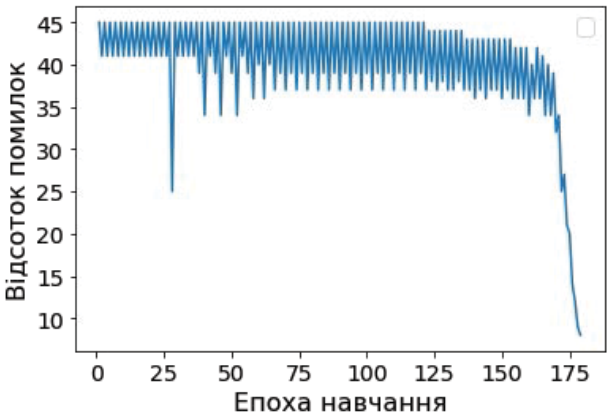


Класи що перекриваються

Вибірка та розділювальна пряма



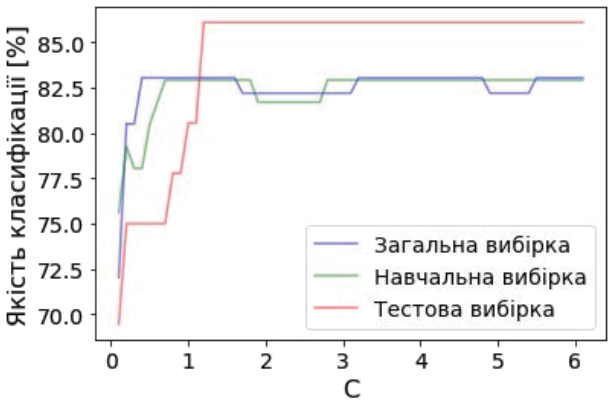
Помилка від епохи навчання



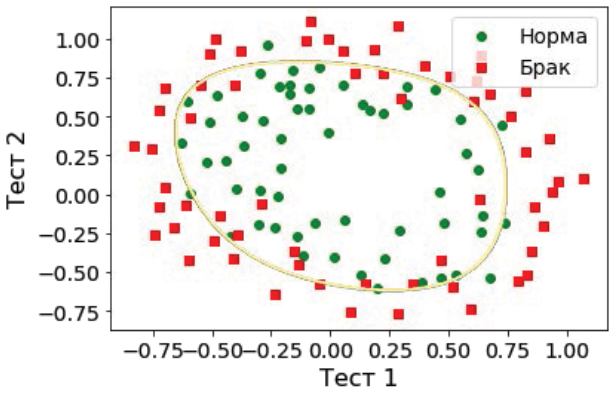
- якість класифікації на навчальній вибірці 89,5%
- якість класифікації на тестовій вибірці 85,7%

Класифікація лінійно не роздільної вибірки

Оптимізація параметра регуляризації



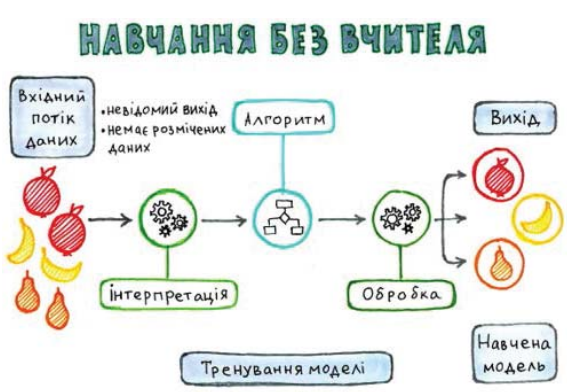
Вибірка та межа



- якість класифікації на навчальній вибірці 82,5%
- якість класифікації на тестовій вибірці 86,2%

Розділ 5

Навчання без учителя: зменшення розмірності



У цьому розділі буде надано основний теоретичний матеріал щодо методів зменшення розмірності даних. Ці алгоритми дозволяють звести велику кількість ознак до меншої для зручності їх подальшого використання. Буде розглянуто два основні і найбільш часто використовувані методи:

- метод головних компонент (Principal component analysis – PCA);
- метод сингулярного розкладання (Singular vector decomposition – SVD).

Для кожного методу буде наведено приклад його використання, поставлено завдання для практичної роботи та наведено приклад подання результатів.

5.1 Метод головних компонент (Principal Component Analysis – PCA)

В аналізі даних, як і в будь-якому іншому аналізі, часом буває незайвим створити спрощену модель, що максимально точно описує реальний стан справ. Часто буває так, що ознаки досить сильно залежать одна від одної та їх одночасна наявність є надмірною. Якщо одна ознака строго залежить від іншої, то знаючи одну, ми завжди знаємо й іншу. Але набагато частіше буває так, що ознаки залежать одна від одної не так строго і (що важливо!) не так очевидно. Але як з'ясувати, який саме набір параметрів добре описує наш набір даних, але водночас має невелику надмірність? Іншими словами, як зменшити розмірність простору, в якому «живуть» дані, втративши водночас мінімум інформації?

Способи вирішення цього завдання називаються методами зменшення розмірності (dimensionality reduction). Метод головних компонент (principal components analysis, PCA) – один із них. Мета PCA – вилучення з цих даних потрібної інформації. Що є інформацією, залежить від суті завдання, що вирішується. Дані можуть містити потрібну нам інформацію, вони можуть бути надлишковими. Проте в деяких випадках, інформації в даних може не бути зовсім. Розмірність даних – кількість зразків і змінних – має велике значення для успішного знаходження інформації. Зайвих даних немає. Краще, коли їх багато, ніж мало.

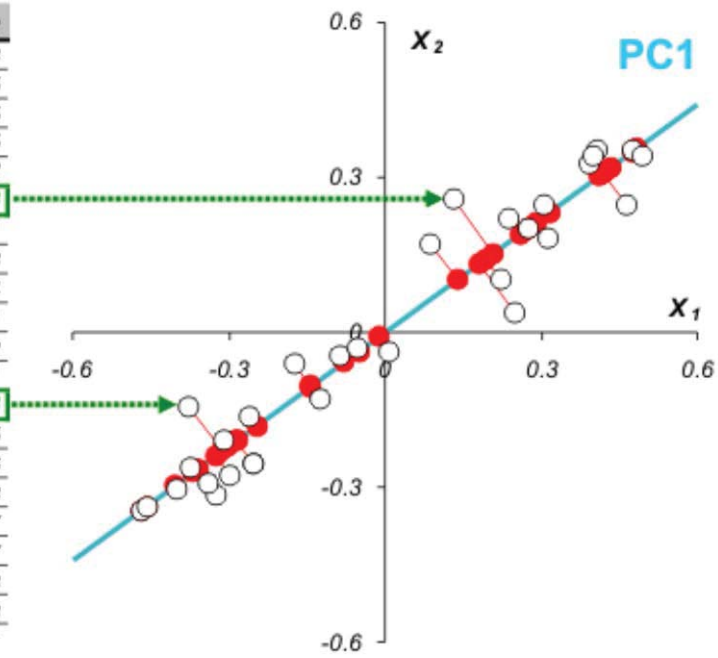
Дані завжди (або майже завжди) містять у собі небажану складову, яка називається шумом. Природа цього шуму може бути різною, але в багатьох випадках, шум – це та частина даних, яка не містить інформації, що шукається. Що вважати шумом, а що інформацією, завжди вирішується з урахуванням поставлених цілей і методів, які використовуються для її досягнення.

Шум і надмірність у даних обов'язково виявляють себе через кореляційні зв'язки між змінними. Похибки даних можуть призвести до появи не систематичних, а випадкових зв'язків між змінними. Поняття ефективного рангу та прихованих, латентних змінних, кількість яких дорівнює цьому рангу, є найважливішим поняттям у PCA.

5.1.1 Інтуїтивний підхід

Розглянемо простий випадок, коли об'єкти описуються лише двома ознаками: x_1 і x_2. Такі дані легко зобразити на площині (див. рисунок).

	X_1	X_2
1	0.407	0.353
2	0.475	0.355
3	-0.088	-0.045
4	0.394	0.325
5	0.274	0.202
6	0.131	0.258
7	-0.053	-0.031
8	-0.124	-0.128
9	-0.469	-0.344
10	0.088	0.171
11	-0.261	-0.162
12	0.401	0.341
13	-0.376	-0.143
14	-0.251	-0.255
15	-0.325	-0.316
16	0.464	0.248
17	-0.310	-0.207
18	0.307	0.247
19	-0.399	-0.303
20	-0.253	-0.253
21	-0.341	-0.291



Кожному рядку вихідної таблиці (тобто зразку) відповідає точка на площині з відповідними координатами. Вони позначені порожніми кружками. Проведемо через них пряму так, щоб уздовж неї відбувалася максимальна зміна даних. На рисунку ця пряма виділена блакитним кольором; вона називається першою головною компонентою PC_1. Потім проєктуємо всі вихідні точки на цю вісь (червоні кружки). Якщо дані описані не повністю (шум великий), то вибирається ще один напрямок (PC_2) – перпендикулярний до першого, так щоб описати зміну, що залишилася в даних і т.д. Отже, знаючи залежності та їх силу, ми можемо виразити кілька ознак через одну, злити докупи, так би мовити, і працювати вже з більш простою моделлю. Звичайно уникнути втрат інформації, швидше за все, не вдасться, але мінімізувати її нам допоможе якраз метод PCA.

Отже, цей метод апроксимує n-розмірну хмару спостережень до еліпсоїда (теж n-вимірного), півосі якого і будуть майбутніми головними компонентами. За проєкції даних на такі осі (зниження розмірності) зберігається найбільше інформації.

5.1.2 Зменшення розмірності даних

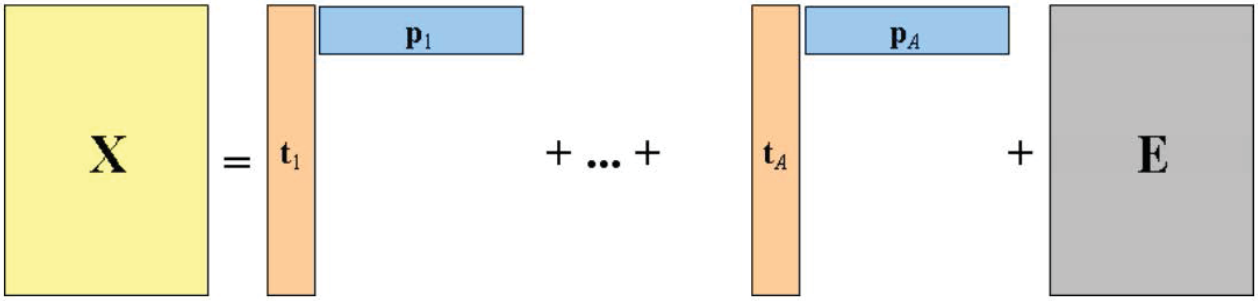
Метод основних компонентів застосовується до даних, записаних у вигляді матриці X – прямокутної таблиці чисел розмірністю I рядків і J стовпців. Традиційно рядки цієї матриці називаються об'єктами. Вони нумеруються індексом $i = 1, \ldots, I$. Стовпці називаються ознаками і вони нумеруються індексом $j = 1, \ldots, J$. У методі головних компонент використовуються нові, формальні змінні t_a $(a = 1, \ldots, A)$, що є лінійною комбінацією вихідних змінних x_j $(j = 1, \ldots, J)$:

$$t_a = p_{a1}x_1 + p_{a2}x_2 + \cdots + p_{aJ}x_J.$$

За допомогою цих нових змінних матриця X розкладається у добуток двох матриць T та P:

$$X = TP^T + E = \sum_{a=1}^{A} t_a p_a^T + E$$

Матриця T називається матрицею рахунків (scores). Її розмірність $(I \times A)$. Матриця P називається матрицею навантажень (loadings). Її розмірність $(J \times A)$. E – матриця залишків, розмірністю $(I \times J)$.



Нові змінні t_a називаються головними компонентами (Principal Components), тому сам метод називається методом головних компонент (PCA). Число стовпців t_a в матриці T, і p_a в матриці P дорівнює A, яке називається числом головних компонент (PC). Ця величина свідомо менша від числа змінних J і числа зразків I.

Важливою властивістю PCA є ортогональність (незалежність) основних компонент. Тому матриця рахунків T не перебудовується зі збільшенням числа компонент, а до неї просто додається ще один стовпець – відповідний новому напрямку. Те саме відбувається і з матрицею навантажень P.

5.1.3 Статистичні основи

Для опису випадкової величини використовуються моменти. Потрібні нам – математичне очікування та дисперсія. Можна сміливо сказати, що математичне очікування – це «центр тяжіння» величини, а дисперсія – її «розмір» (розкид). Сам процес проєктування на вектор ніяк не впливає на значення середніх, тому що для мінімізації втрат інформації наш вектор повинен проходити через центр нашої вибірки. Тому зазвичай проводять центрування вибірки — лінійно зміщуючи її так, щоб середні значення ознак дорівнювали 0. Оператор, зворотний зсуву дорівнюватиме вектору початкових середніх значень — він потрібен для відновлення вибірки у вихідній розмірності. При цьому, дисперсія залежить від порядків значень випадкової величини, тобто є чутливою до масштабування. Тому якщо одиниці вимірювання ознак дуже відрізняються своїми порядками, рекомендується стандартизувати їх за формулою

$$x_i = \frac{x_i - \langle x_i \rangle}{\sigma_i},$$

де $\langle x_i \rangle$ середнє значення ознаки x_i; σ_i – її дисперсія.

Для опису форми випадкового вектора необхідна матриця коваріації. Це матриця, в якої елемент (i, j) є кореляцією ознак (x_i, x_j). Формула коваріації має вигляд

$$\begin{aligned} Cov(x_i, x_j) =& E\left[(x_i - E(x_i)) \cdot (x_j - E(x_j))\right] \\ & E(x_i, x_j) - E(x_i) - E(x_j), \end{aligned} \tag{5.1}$$

де $E(\dots)$ – середнє від вектора. У нашому разі вона спрощується, тому що $E(x_i) = E(x_j) = 0$. За умови проведеного нормування:

$$Cov(x_i, x_j) = E(x_i, x_j). \tag{5.2}$$

Зауважимо, що коли $x_i = x_j$:

$$Cov(x_i, x_i) = Var(x_i) \tag{5.3}$$

і це справедливо для будь-яких випадкових величин.

Отже, у коваріаційній матриці по діагоналі будуть дисперсії ознак (оскільки $i = j$), а в інших комірках — коваріації відповідних

пар ознак. Оскільки коваріація характеризується симетричністю, то й матриця теж буде симетрична. Коваріаційна матриця є узагальненням дисперсії на випадок багатовимірних випадкових величин. Вона так само описує форму (розкид) випадкової величини, як і дисперсія, яка для одновимірної випадкової величини має вигляд матриці розміру 1×1, в якій її єдиний член заданий формулою $Cov(x, x) = Var(x)$. Коваріаційна матриця описує форму нашої випадкової величини (ознаки). Узагальнення дисперсії на вищі розмірності – матриця коваріації, і ці два поняття еквівалентні. Під час проєкції на вектор максимізується дисперсія проєкції, під час проекції на простір великих порядків – вся її коваріаційна матриця.

5.1.4 Власні значення та власні вектори

Тепер необхідно знайти такі вектори, за яких максимізувався б розкид (дисперсія) проєкції вибірки на них. Візьмемо одиничний вектор на який проєктуватимемо наш випадковий вектор X. Тоді проєкція на нього дорівнюватиме $v^T X$ (v^T – транспонування вектора v). Дисперсія проєкції на вектор відповідно дорівнює $Var(v^T X)$. Загалом у векторній формі (для центрованих величин) дисперсія виражається так:

$$Var(X) = \Sigma = E(X \cdot X^T).$$

Відповідно дисперсія проєкції:

$$\begin{aligned} Var(X^*) = \Sigma^* = & E(X^* \cdot X^{*T}) = E\left((v^T X) \cdot (V^T X)^T\right) = \\ & E\left((v^T X \cdot X^T v\right) = v^T E\left(X \cdot X^T\right) v = v^T \Sigma v. \end{aligned} \tag{5.4}$$

Отже, дисперсія максимізується за умови максимального значення $v^T \Sigma v$. Далі зручно використати співвідношення Релея, які для коваріаційних матриць M мають спеціальний випадок:

$$R(M, x) = \frac{X^T M X}{X^T X} = \lambda \frac{X^T X}{X^T X} \lambda \tag{5.5}$$

і відповідно маємо

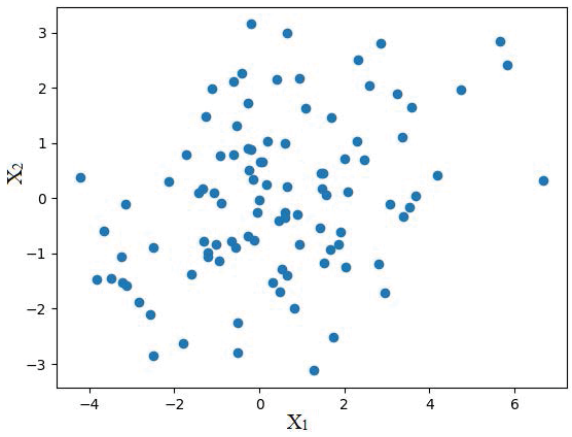
$$MX = \lambda X. \tag{5.6}$$

Остання формула є розкладанням матриці на власні вектори та значення: X є власним вектором, а λ – власним значенням. Кількість власних векторів і значень дорівнюють розміру матриці (і значення можуть повторюватися).

Отже, робимо висновок, що напрямок максимальної дисперсії у проекції завжди збігається з власним вектором, що має максимальне власне значення, яке дорівнює величині цієї дисперсії. Це справедливо також для проєкцій на більшу кількість вимірювань – дисперсія (коваріаційна матриця) проєкції на m-вимірний простір буде максимальною в напрямку m власних векторів, що мають максимальні власні значення.

5.1.5 Зменшення розмірності двовимірної вибірки

Розглянемо простий приклад зменшення розмірності двовимірної вибірки, яку подано на рисунку.



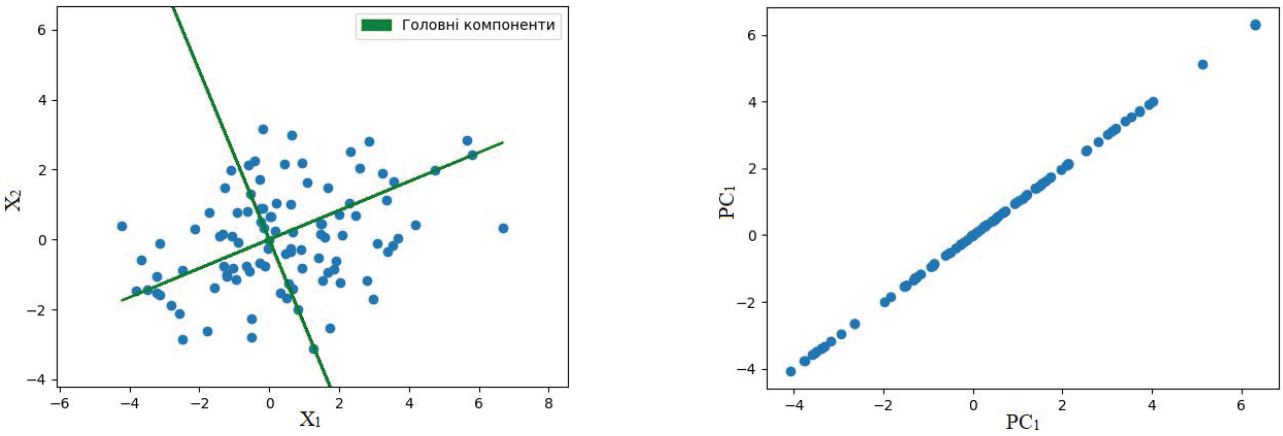
Бачимо, що дисперсія є максимальною в напрямку зростання (спадання) обох величин X_1 і X_2. Це і буде перша головна компонента PC_1, яка буде подаватися лінійною комбінацією X_1 і X_2. Відповідно друга головна компонента буде мати напрямок перпендикулярний по першої.

Для встановлення головних компонент розраховуємо матрицю коваріацій і знаходимо її власні значення та власні вектори. Власні вектори будуть зазначати напрямок зміни дисперсій нашої вибірки – новий базис. Для нашої вибірки маємо:

$$\lambda = \begin{pmatrix} 4,41 \\ 1,58 \end{pmatrix} \qquad v = \begin{pmatrix} 0,92 & -0,38 \\ 0,38 & 0,92 \end{pmatrix} \tag{5.7}$$

Отже, перший вектор із максимальним власним значенням визначає напрямок першої головної компоненти, другий – другої. Вибірку з власними векторами коваріаційної матриці наведено на рисунку

зліва. Для проведення проєкції, потрібно провести операцію $v^T X$ (вектор повинен бути довжини 1). Або якщо у нас не один вектор, а гіперплощина, то замість вектора v^T беремо матрицю базисних векторів V^T. Одержаний вектор (або матриця) буде масивом проєкцій спостережень. Для нашої вибірки проєкція на перший власний вектор із максимальним власним значенням наведено на рисунку справа.



Часто виникає потреба оцінити обсяг втраченої і збереженої інформації. Найзручніше це подавати у відсотках. Для цього використовуємо дисперсії по кожній осі та ділимо на загальну суму дисперсій по осях (тобто суму всіх власних чисел коваріаційної матриці). Отже, перший вектор описує $(4,41/5,99) \times 100\ \% \simeq 74\ \%$, а менший відповідно приблизно 26 % всієї інформації. Отже, спроєктувавши дані на перший вектор ми втратимо приблизно 26 % інформації, проте зменшимо кількість даних вдвічі для подальшого аналізу.

5.1.6 Постановлення задачі

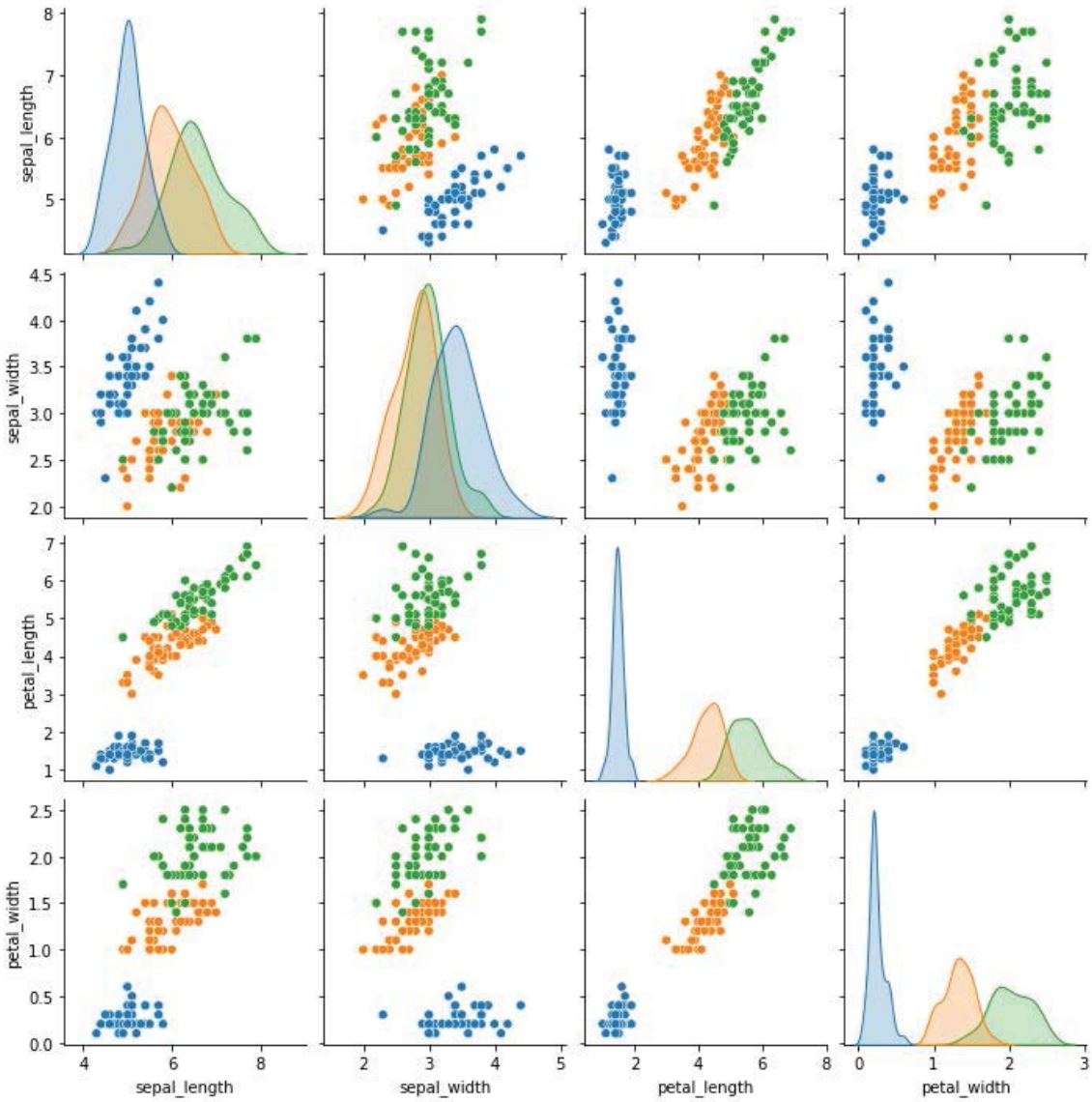
Провести реалізацію алгоритму PCA для зменшення розмірності вибірки до двох.
Етапи розв'язання

1. Імпортувати дані з кількістю ознак більше трьох.
2. Побудувати матрицю коваріацій.
3. Розрахувати власні вектори та власні значення матриці.
4. Відсортувати власні вектори в порядку спадання власних значень.
5. Візуалізувати власні значення.
6. Спроєктувати дані на два перші власні вектори.
7. Візуалізувати одержану двовимірну вибірку.
8. Розрахувати принципові компоненти за допомогою вбудованої функції PCA з кількістю компонентів 2 з бібліотеки sklearn.
9. Порівняти результати роботи власного алгоритму з результатами функції PCA з бібліотеки sklearn.
10. Візуалізувати різницю в одержаних вибірках.
11. Оформити результати у вигляді звіту.

5.1.7 Приклад подання результатів

Розглянемо вибірку *iris* з бази даних sklearn.

1. Побудуємо парні залежності ознак, що подано на рисунку.

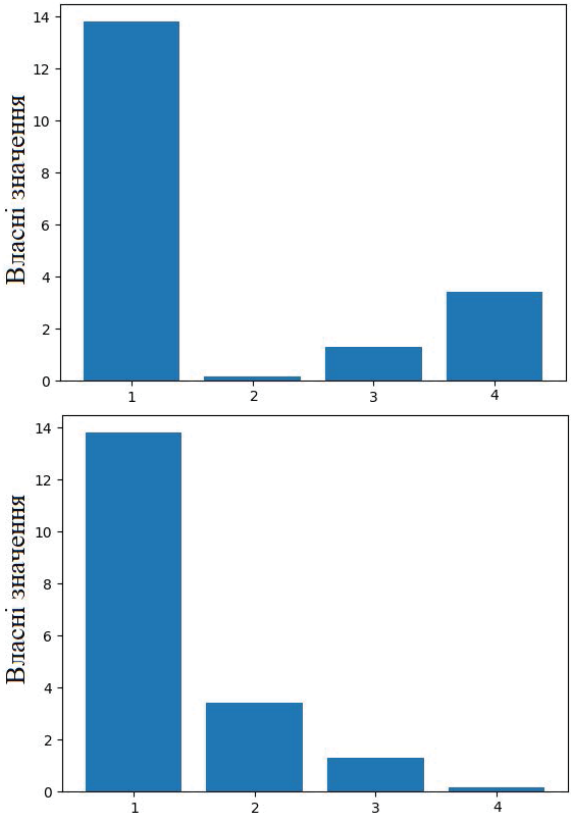


Оскільки перший клас добре виділяється від інших двох, залишимо у вибірці лише дані для другого та третього класів. Одержуємо вибірку $X = [\ell \times n]$, де ℓ – кількість об'єктів; n – кількість ознак (для цього прикладу $n = 4$).

2. Проводимо нормалізацію даних: центрування та зведення до однакового розкиду (дисперсії):

$$X_i = (X_i - \langle X_i \rangle)/\sigma.$$

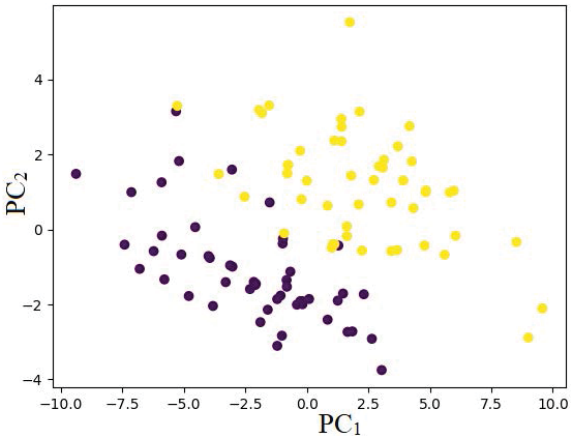
3. Розрахунок матриці коваріацій та її власних векторів і власних значень.



Розраховуємо матрицю коваріацій; розраховуємо власні вектори та власні значення. Візуалізуємо одержані власні значення.

Оскільки власні вектори не відсортовані за спаданням, проводимо їх сортування й відповідно сортуємо власні вектори. Візуалізуємо відсортовані власні значення.

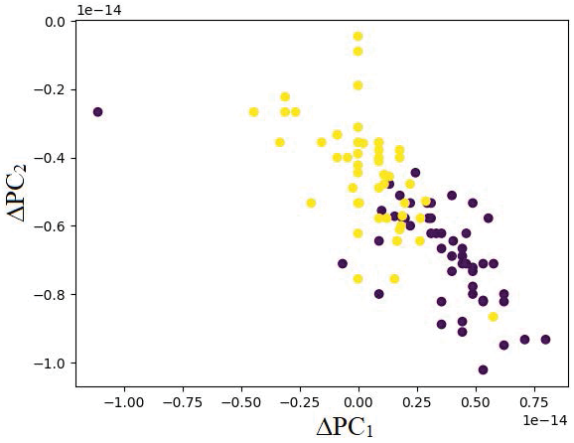
4. Встановимо кількість головних компонент $n_{PC} = 2$. Проєктуємо дані на перші два власні вектори матриці коваріації, що характеризуються найбільшими власними значеннями.



Множимо вихідну матрицю даних розміру $[\ell \times n]$ на транспоновану матрицю двох власних векторів розміру $[n \times n_{PC}]$ для одержання нової матриці даних розміром $[\ell \times n_{PC}]$. Візуалізуємо одержані дані.

5. Використовуючи вбудовану функцію PCA з пакету sclearn, проводимо зменшення розмірності вибірки X до n_{PC} компонент.

6. Порівняння результатів.



Розраховуємо для кожної з головних компонент PC_i, $i = 1, 2$ різницю між одержаними даними ΔPC_i за власним алгоритмом і вбудованою функцією PCA. Візуалізуємо одержані результати.

7. Розраховуємо кількість втраченої інформації, використовуючи власні значення матриці коваріацій під час зменшення розмірності з чотирьох ознак до двох:

$$\Delta I = 1 - \frac{\lambda_1 + \lambda_2}{\sum\limits_{i=1}^{n} \lambda_i} \cdot 100\ \% \simeq 7,9\ \%$$

Отже, проведена процедура зменшення розмірності дозволила зменшити кількість даних у два рази, водночас втрати інформації становили менше 8 %.

5.2 Сингулярне розкладання матриць (Singular Value Decomposition – SVD)

Сингулярне розкладання (Singular Value Decomposition, SVD) – декомпозиція матриці з метою її приведення до канонічного вигляду. Сингулярне розкладання є зручним методом під час роботи з матрицями. Воно показує геометричну структуру матриці і дозволяє наочно уявити дані. Сингулярне розкладання використовується під час вирішення найрізноманітніших завдань – від наближення методом найменших квадратів і розв'язання систем рівнянь до стискання зображень. Водночас використовуються різні властивості сингулярного розкладання, наприклад, здатність показувати ранг матриці, наближати матриці цього рангу. SVD дозволяє обчислювати зворотні та псевдозворотні матриці великого розміру, що робить його корисним інструментом під час вирішення завдань регресійного аналізу.

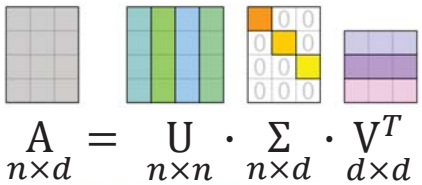
Матрична факторизація або матрична декомпозиція широко використовується під час побудови рекомендаційних систем. Такі системи були визначені як програмні засоби та методи, що пропонують пропозиції щодо елементів, які можуть бути корисними для користувача. Два основні поняття тут – це елементи та користувачі. Як простий приклад можна розглянути елементи як книги на Amazon, а користувачів як читачів. Рекомендаційна системи зазвичай починається з побудови матриці рейтингів. Матриця оцінок створюється шляхом агрегування даних про інтереси користувача в окрему оцінку елемента користувача для користувача та пари елементів. Рядки подають користувачів, а стовпці – елементи. Тоді рейтинг залежить від спостережуваного інтересу, виміряного неявно чи явно. Ідея полягає в тому, щоб факторизувати або розкласти матрицю на добуток інших матриць, які забезпечують більш стисле, цілеспрямоване подання інформації в матриці рейтингів. Інакше кажучи, факторизація – це спосіб наближення матриці, коли вона схильна до зменшення розмірності через кореляції між стовпцями або рядками. У своїй найпростішій формі ми розкладаємо рейтингову матрицю на матрицю характеристик користувача та матрицю характеристик товару.

За допомогою сингулярного розкладання матрицю можна розбити та записати як скалярний добуток трьох матриць, які мають деякі приємні властивості. Наприклад, одна з цих матриць є стислим поданням вихідної матриці, яка зазвичай корисна в багатьох контекстах. У задачах машинного навчання сингулярним розкладанням матриці «об'єкти-ознаки» A розміру $[n \times d]$: n об'єктів $(X^n = \{x_1, \ldots, x_n\})$ з d ознаками $(f^d = \{f_1, \ldots, f_d\})$ може бути подано у такому вигляді

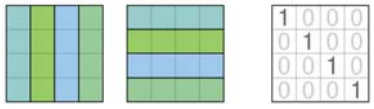
$$A_{[n\times d]} = U_{[n\times n]} \cdot \Sigma_{[n\times d]} \cdot V^T_{[d\times d]}.$$

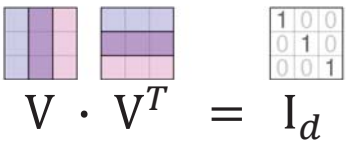
5.2.1 Властивості та обмеження на матриці U, Σ, V

Властивості та обмеження на матриці U, Σ, V:



- матриця U має розміри $n \times n$;
- матриця Σ має розмір $n \times d$;
- матриця V має розмір $d \times d$;
- U і V – ортогональні матриці, тобто $U^TU = I$ і $V^TV = I$;





- $\Sigma = \Sigma^T$ є діагональною матрицею: $\Sigma = \mathrm{diag}(\sigma_1, \ldots \sigma_d)$, тобто всі елементи в матриці Σ дорівнюють нулю, якщо вони не лежать на діагоналі;
- Діагональні елементи в Σ розташовані від найбільшого до найменшого: $\sigma_1 > \sigma_2 > \ldots > \sigma_d$;
- Σ – сингулярні значення матриці A;
- стовпці матриці U – ліві сингулярні вектори;
- стовпці матриці V – праві сингулярні вектори.

5.2.2 Математичний сенс трьох матриць U, Σ, V

Сингулярне розкладання володіє властивістю, яка пов'язує задачу відшукання сингулярного розкладання й завдання відшукання власних векторів. Власний вектор x матриці A – такий вектор, за якого виконується умова $Ax = \lambda x$, число λ називається власним числом. Так як матриці U і V ортогональні, то

$$AA^T = U\Sigma V^T V\Sigma U^T = [V^T V = I] = U\Sigma^2 U^T,$$
$$A^T A = V\Sigma U^T U\Sigma V^T = [U^T U = I] = V\Sigma^2 V^T.$$

Домножуючи обидва вирази справа відповідно на U і V, одержуємо

$$AA^T U = U\Sigma^2,$$
$$A^T AV = V\Sigma^2.$$

Із цього випливає, що стовпці матриці U є власними векторами матриці AA^T, а квадрати сингулярних чисел $\Sigma = \mathrm{diag}(\sigma_1, \ldots \sigma_d)$ – її власними числами. Водночас, стовпці матриці V є власними векторами матриці коваріації $S = A^T A$, а квадрати сингулярних чисел є її власними числами.

Отже,

- стовпці $v_1, \ldots, v_d$ матриці V являють собою власний базис матриці коваріацій; S – власні вектори;
- $\sigma_1, \ldots, \sigma_d$ – власні значення матриці коваріацій S;
- вектори $v_1, \ldots, v_d$ – головні (основні) компоненти (principal components) для матриці даних S.

Для будь-якого вектора $\tilde{x}_i = Av_i$ варіація

$$Var(\tilde{x}_i) = Var(Av_i) = E(v_i^T V\Sigma U^T \cdot U\Sigma V^T v_i) =$$
$$= \Sigma^2(v_i^T VU^T \cdot UV^T v_i) = \Sigma^2(v_i^T v_i) = \sigma_i^2.$$

Отже, перша головна компонента v_i характеризується тією властивістю, що $(\tilde{x}_i)$ має максимальну дисперсію серед усіх нормованих лінійних комбінацій стовпців матриці A; $(\tilde{x}_d)$ має мінімальну дисперсію.

5.2.3 Стискання даних

Використовуючи значення σ_i матриці Σ, можна оцінити кількість збереженої інформації I чи втраченої інформації $100\ \% - I$ у відсотках під час використанні лише певної кількості m перших факторів розкладання ($m < d$), d – загальна кількість ознак):

$$I_m = \frac{\sum_{i=1}^{m} \sigma_i}{\sum_{j=1}^{d} \sigma_j} \cdot 100\ \%.$$

Водночас, використання лише перших m факторів розкладання: m перших стовпців матриці U, m перших чисел матриці Σ та m перших рядків матриці V^T можна зменшити кількість пам'яті для зберігання даних. Дійсно, вся матриця містить $n \times d$ чисел, тоді як розкладання з використанням m факторів потребує зберігання

$$n \times m + m + d \times m$$

чисел. Відповідно відсоток стиснення даних можна порахувати за формулою

$$S = \frac{m \times (n + d + 1)}{n \times d} \cdot 100\ \%.$$

Для визначення у скільки разів редукована таблиця менша (за кількістю даних для зберігання) від вихідної використовують коефіцієнт стиснення:

$$K = \frac{n \times d}{m \times (n + d + 1)}.$$

5.2.4 Приклад використання SVD

Як приклад, розглянемо результати гри в гольф. Гравці 1, 2 і 3 грають разом на дев'яти лунках. Їх карта результатів, яку можна розглядати як матрицю (лунка/гравець) має такий вигляд.

Hole	Par	Player1	Player2	Player3
1	4	4	4	4
2	5	5	5	5
3	3	3	3	3
4	4	4	4	4
5	4	4	4	4
6	4	4	4	4
7	4	4	4	4
8	3	3	3	3
9	5	5	5	5

Par – нормативна кількість ударів за які гравець повинен пройти цю лунку. Розглянемо проблему, пов'язану зі спробою передбачити, який рахунок набере кожен гравець на певній лунці. Одна з ідей полягає в тому, щоб надати кожній лунці фактор складності лунки (*HoleDifficulty*), а кожному гравцеві – коефіцієнт майстерності гравця (*PlayerAbility*). Фактична оцінка буде результатом множення цих двох факторів:

$$PredictedScore = HoleDifficulty \cdot PlayerAbility.$$

Для першої спроби давайте зробимо *HoleDifficulty* таким, що дорівнює нормативній кількості ударів *Par*, а коефіцієнт майстерності гравця *PlayerAbility* таким, що дорівнює 1. Отже, на першій лунці з $HoleDifficulty = 4$, ми очікуємо, що гравець із $PlayerAbility = 1$ одержить кількість балів

$$PredictedScore = HoleDifficulty \cdot PlayerAbility = 4 \cdot 1 = 4.$$

Для всієї нашої картки показників або матриці все, що нам потрібно зробити, це помножити *PlayerAbility* (припускаємо, що він дорівнює 1 для всіх гравців) на *HoleDifficulty* (діапазон від 3 до 5), і ми зможемо точно передбачити всі бали в нашому прикладі.

Фактично це одновимірна (1-D) SVD факторизація системи показників. Ми можемо репрезентувати нашу систему показників або матрицю як добуток двох векторів: вектора *HoleDifficulty* і вектора *PlayerAbility*. Передбачення будь-якої кількості балів із таблиці досягається множенням відповідного коефіцієнта *HoleDifficulty* на відповідний коефіцієнт *PlayerAbility*.

Player1	Player2	Player3
4	4	4
5	5	5
3	3	3
4	4	4
4	4	4
4	4	4
4	4	4
3	3	3
5	5	5

=

HoleDifficulty
4
5
3
4
4
4
4
3
5

x

PlayerAbility		
Player1	Player2	Player3
1	1	1

Відмасштабуємо вектори так, щоб їх довжина становила 1. Наприклад, вектор *PlayerAbility* змінимо так, щоб сума квадратів його елементів дорівнювала 1, а не поточним $1^2 + 1^2 + 1^2 = 3$. Щоб зробити це, ми повинні розділити кожен елемент на квадратний корінь із 3. Так само розділимо кожен елемент *HoleDifficulty* на квадратний корінь зі 148. Квадратний корінь із 3, помножений на квадратний корінь зі 148, є нашим коефіцієнтом масштабування $ScaleFactor = 21,07$. У такому разі повна факторизація 1-D SVD для цього прикладу набуває вигляду

Player1	Player2	Player3
4	4	4
5	5	5
3	3	3
4	4	4
4	4	4
4	4	4
4	4	4
3	3	3
5	5	5

=

HoleDifficulty
0.33
0.41
0.25
0.33
0.33
0.33
0.33
0.25
0.41

x

ScaleFactor
21.07

x

PlayerAbility		
Player1	Player2	Player3
0.58	0.58	0.58

Вектор *HoleDifficulty* є лівим сингулярним вектором; *ScaleFactor* – це сингулярне значення, а вектор *PlayerAbility* – правий сингулярний вектор. Якщо репрезентувати ці три частини й помножити їх разом, ми одержимо точні початкові оцінки. Це означає, що наша матриця є матрицею рангу 1, інакше можна сказати, що вона має простий і передбачуваний шаблон.

Розширення SVD за допомогою додаткових факторів

Більш складні матриці не можна повністю передбачити, використовуючи лише один набір факторів, як це було в попередньому прикладі. У такому разі ми повинні ввести другий набір факторів, щоб уточнити наші прогнози. Для цього ми віднімаємо прогнозовані бали від фактичних, одержуючи залишкові бали. Потім знаходимо другий набір чисел *HoleDifficulty2* і *PlayerAbility2*, що найкраще передбачають залишкові бали.

Замість того, щоб вгадувати фактори *HoleDifficulty* і *PlayerAbility* та віднімати передбачувані бали, є потужні алгоритми, що можуть обчислити факторизацію SVD. Розглянемо більш складну ситуацію – фактичні результати перших 9 лунок *Players Championship 2007*. 1-D SVD факторизація оцінок показана нижче. Щоб полегшити розуміння цього прикладу, *ScaleFactor* включено до векторів *PlayerAbility* та *HoleDifficulty*.

Hole	Par	Player1	Player2	Player3
1	4	4	4	5
2	5	4	5	5
3	3	3	3	2
4	4	4	5	4
5	4	4	4	4
6	4	3	5	4
7	4	4	4	3
8	3	2	4	4
9	5	5	5	5

Player1	Player2	Player3
3.95	4.64	4.34
4.27	5.02	4.69
2.42	2.85	2.66
3.97	4.67	4.36
3.64	4.28	4.00
3.69	4.33	4.05
3.33	3.92	3.66
3.08	3.63	3.39
4.55	5.35	5.00

=

HoleDifficulty1
4.34
4.69
2.66
4.36
4.00
4.05
3.66
3.39
5.00

x

PlayerAbility1		
Player1	Player2	Player3
0.91	1.07	1.00

Наголосимо, що коефіцієнт складності лунки є майже середнім для цієї лунки для трьох гравців. Наприклад, лунка 5, у якій кожен одержав 4 бали, дійсно має коефіцієнт 4,00. Проте лунка 6,

у якій середня оцінка також дорівнює 4, має коефіцієнт 4,05 замість 4,00. Подібно до цього, *PlayerAbility* – це майже відсоток від номіналу, якого досяг гравець. Наприклад, гравець *Player2* зробив 39 ударів, тоді як показник *Par* для нього – 36 і $39/36 = 1{,}08$, що майже дорівнює його коефіцієнту *PlayerAbility* (для цих 9 лунок) 1,07.

Чому середні значення лунок і номінальні відсотки точно не збігаються з факторами 1-D SVD? Відповідь полягає в тому, що SVD додатково уточнює ці числа в циклі. Наприклад, ми можемо почати з припущення, що *HoleDifficulty* є середнім значенням лунки, а потім запитати, яка *PlayerAbility* найкраще відповідає оцінкам, ураховуючи ці числа *HoleDifficulty*. Одержавши цю відповідь, ми можемо повернутися назад і запитати: яка *HoleDifficulty* найкраще відповідає оцінкам, заданим цими числами *PlayerAbility*? Ми повторюємо цей шлях доти, доки не досягнемо набору факторів, що найкраще прогнозують оцінку. SVD скорочує цей процес та одразу дає нам фактори, до яких ми зійшлися б, якби виконали процес.

Однією дуже корисною властивістю SVD є те, що він завжди знаходить оптимальний набір факторів, які найкраще передбачають бали, відповідно до стандартної міри подібності матриці. Тобто, якщо ми використовуємо SVD, щоб знайти множники матриці, це найкращі можливі множники. Ця властивість оптимальності означає, що нам не потрібно ставити запитання: чи може інший набір чисел краще передбачати результати?

Тепер давайте проаналізуємо різницю між фактичними оцінками й нашим одновимірним наближенням. Плюсова різниця означає, що фактична оцінка вища за прогнозовану, мінусова – навпаки. Наприклад, на першій лунці гравець *Player2* одержав 4 бали, а передбачуваний бал був 4,64, тому $4 - 4,64 = -0,64$. Іншими словами, ми повинні додати $-0,64$ до нашого прогнозу, щоб визначити фактичну оцінку. Коли ці відмінності буде знайдено, ми можемо зробити те саме ще раз і передбачити ці відмінності за допомогою формули it HoleDifficulty2 $\cdot$ *PlayerAbility2*. Оскільки ці фактори намагаються передбачити відмінності, вони є наступними факторами, і ми поставили 2 після їх назв (наприклад, *HoleDifficulty2*), щоб показати, що це другий набір факторів.

Player1	Player2	Player3
0.05	-0.64	0.66
-0.28	-0.02	0.31
0.58	0.15	-0.66
0.03	0.33	-0.36
0.36	-0.28	0.00
-0.69	0.67	-0.05
0.67	0.08	-0.66
-1.08	0.37	0.61
0.45	-0.35	0.00

=

HoleDifficulty2
-0.18
-0.38
0.80
0.15
0.35
-0.67
0.89
-1.29
0.44

x

PlayerAbility2		
Player1	Player2	Player3
0.82	-0.20	-0.53

Є кілька цікавих спостережень, що ми можемо зробити щодо цих факторів. Зверніть увагу, що лунка 8 має найзначніший коефіцієнт *HoleDifficulty2* $(-1,29)$. Це означає, що це найскладніша лунка для передбачення. Дійсно, це була єдина лунка, на якій жоден із трьох гравців не зробив показника *Par*. Це було особливо проблематично передбачити, тому що це була найскладніша лунка відносно номіналу $HoleDifficulty - Par = 3,39 - 3 = 0,39$, проте гравець *Player1* забив її, зробивши свій рахунок більше ніж на удар нижчим за прогнозований рахунок (він набрав 2 проти його прогнозованого балу 3,08). Іншими, складними для передбачення лунками, були лунки 3 (0,80) і 7 (0,89). Гравець *Player3* переміг гравця *Player1* на цих лунках, хоча загалом гравець *Player1* грав краще.

Повна факторізація SVD

Повне розкладання SVD для цього прикладу матриці (9 лунок на 3 гравці) містить 3 набори факторів. Загалом матриця $n \times d$, де $n \geq d$, може мати щонайбільше d факторів, тому наша матриця 9×3 не може містити більше ніж 3 набори факторів. Ось повна розкладка SVD (з точністю до двох знаків після коми).

Player 1-3		
4	4	5
4	5	5
3	3	2
4	5	4
4	4	4
3	5	4
4	4	3
2	4	4
5	5	5

=

HoleDifficulty 1-3		
4.34	-0.18	-0.90
4.69	-0.38	-0.15
2.66	0.80	0.40
4.36	0.15	0.47
4.00	0.35	-0.29
4.05	-0.67	0.68
3.66	0.89	0.33
3.39	-1.29	0.14
5.00	0.44	-0.36

X

PlayerAbility 1-3		
0.91	1.07	1.00
0.82	-0.20	-0.53
-0.21	0.76	-0.62

Згідно з умовою SVD усі вектори *HoleDifficulty* і *PlayerAbility* повинні мати довжину 1, тому стандартна факторизація SVD

Player 1-3		
4	4	5
4	5	5
3	3	2
4	5	4
4	4	4
3	5	4
4	4	3
2	4	4
5	5	5

=

HoleDifficulty 1-3		
0.35	0.09	-0.64
0.38	0.19	-0.10
0.22	-0.40	0.28
0.36	-0.08	0.33
0.33	-0.18	-0.20
0.33	0.33	0.48
0.30	-0.44	0.23
0.28	0.64	0.10
0.41	-0.22	-0.25

X

ScaleFactor 1-3		
21.07		
	2.01	
		1.42

X

PlayerAbility 1-3		
0.53	0.62	0.58
-0.82	0.20	0.53
-0.21	0.76	-0.62

Використовуючи значення σ_i матриці *ScaleFactor*, можна оцінити кількість збереженої інформації. Для прикладу порахуємо кількість інформації, що зберігається в разі використання лише перших факторів, $m = 1$; $I_1 = 21,07^2/(21,07 + 2,01 + 1,42) \times 100\ \% = 86\ \%$. Отже, в разі використання лише перших факторів буде втрачено не більше 14 % інформації. Водночас використання лише перших m факторів розкладання: m перших стовпців матриці *HoleDifficulty* (m перших чисел таблиці *ScaleFactor* та m перших рядків таблиці *PlayerAbility*) може зменшити кількість пам'яті для зберігання даних. Для одномірного розкладання з $m = 1$

$$S = \frac{1 \times (9 + 3 + 1)}{9 \times 3} \times 100\ \% \simeq 48\ \%.$$

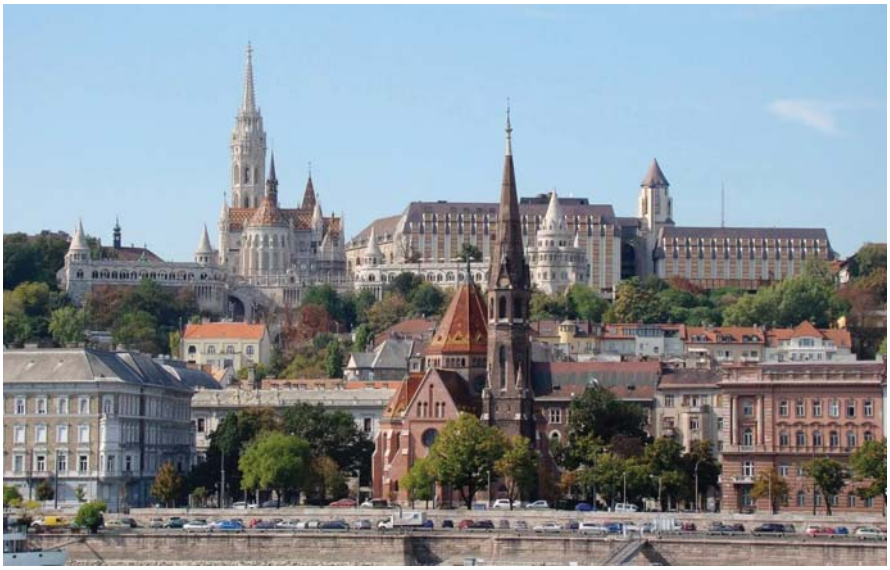
Отже, для цього прикладу використання одномірного SVD розкладання дає змогу вдвічі зменшити кількість даних для зберігання, втрачаючи не більше ніж 14 % інформації.

5.2.5 Використання SVD для стиснення зображень

Сингулярне розкладання матриць призначене для зменшення розмірності будь-яких даних. Одним із його застосувань є стиснення зображень.

Чорно-біле зображення є матрицею розміром $n \times m$, у якій кожен елемент відповідає значенню градацій сірого для цього пікселя. Для кольорових зображень ми маємо тривимірний масив розміром $n \times m \times 3$, де n і m – кількість пікселів по вертикалі й горизонталі відповідно. Для кожного пікселя ми зберігаємо інтенсивність червоного, зеленого та синього кольорів (RGB-подання).

Як приклад розглянемо зображення Замку Буда, або Будайської фортеці – резиденції угорських королів у Будапешті.



Відцифруємо картинку й репрезентуємо її у вигляді матриці A розміром $n \times d \times 3$, $n = 1600$, $d = 1016$. Після цього стиснемо матрицю A, обчисливши SVD-наближення до неї, що займає лише частину пам'яті для зберігання. Оскільки дані в матрицях U, Σ і V відсортовані за внеском у матрицю A в добутку, це дає змогу одержати досить гарне наближення, просто використовуючи

лише найважливіші частини матриць. Шляхом вибору m найбільших (а отже, найважливіших) сингулярних значень із матриці Σ, що ми збираємося використовувати для апроксимації, та відповідно m стовпців матриць U і V, відновлюємо матрицю A', яка є апроксимацією вихідної матриці A. Чим більша кількість m, тим краща якість апроксимації, але також тим більше даних потрібно для її кодування.

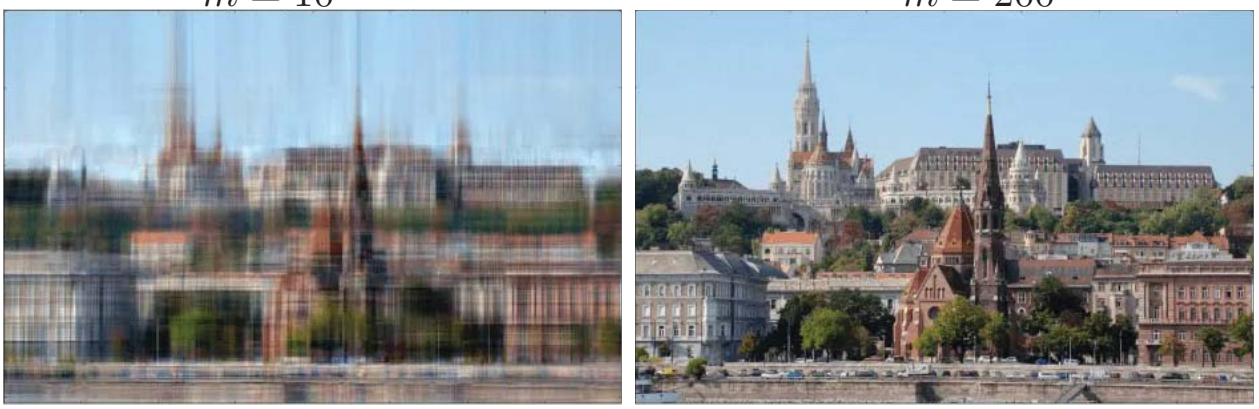
Перевіримо, як цей вибір впливає на якість апроксимації та ступінь стиснення. Спершу розділимо значення градацій сірого для кожного кольору, тобто будемо працювати з трьома незалежними двовимірними матрицями й виконаємо SVD-розкладання для кожного кольору окремо. Покладемо $m = 50$, тобто побудуємо найкраще наближення рангу 50 матриці інтенсивності для кожного кольору.

Ми можемо підрахувати, скільки байтів потрібно для зберігання фрагментів наближення рангу 50 порівняно з розміром вихідного зображення. Насправді ми досягли рівня стиснення $S = 8\ \%$ (зменшивши кількість даних у $K = 12,5$ разів), що означає, що лише 8 % оригінального простору для зберігання використовується для тих матриць, із яких ми можемо відновити матрицю, близьку до вихідної. Проте гарний ступінь стиснення зазвичай означає досить погану якість. Відновлена матриця може бути близькою до оригінального зображення, але чи задоволені ми результатом? Перевіримо, чого ми досягли.

$m = 50$

Будинки впізнавані, але приблизне зображення досить спотворене й шумне. Проведемо розкладання для меншої ($m = 10$) та більшої ($m = 200$) кількості сингулярних значень. Порівняємо: для зберігання матриць для $m = 10$ потрібно 62 080 байт, що призводить до вражаючого рівня стиснення 1.6 % (стиснення більше ніж у 60 разів) (хоча відновлене зображення майже невпізнане), тоді як для $m = 200$ простір для зберігання становить 12 561 600 байт, а ступінь стиснення – 32.2 % (зменшилася кількість даних для зберігання зображення приблизно втричі).

$m = 10$ $m = 200$



Отже, сингулярне розкладання матриць є дуже потужним математичним інструментом із застосуванням у багатьох галузях. SVD-розкладання зазвичай є раціональним вибором, коли потрібно стиснути великий набір даних (тобто зменшити їх розміри) так, щоб певним чином зберегти внутрішню структуру й зв'язки відповідності між точками даних. SVD-розкладання матриць використовують як інструмент для аналізу основних компонент при проведення PCA-аналізу.

5.2.6 Поставлення задачі

Реалізувати алгоритм SVD для стиснення зображення.

Етапи розв'язання

1. Відцифрувати чорно-білий (за бажанням кольоровий) малюнок.

2. Розкласти матрицю за допомогою вбудованої функції SVD.

3. Проаналізувати сингулярні значення, побудувати гістограми перших 20 з них.

4. Порахувати кількість втраченої інформації у відсотках за умови використання лише $1, 2, 3, \ldots, N$ сингулярних значень. Графічно побудувати цю залежність.

5. Порахувати показник стиснення у відсотках за умови використання лише $1, 2, 3, \ldots, N$ сингулярних значень. Графічно побудувати цю залежність.

6. Порахувати коефіцієнт стиснення за умови використання лише $1, 2, 3, \ldots, N$ сингулярних значень. Графічно побудувати цю залежність.

7. Відтворити малюнок для $1, 5, 15, \ldots$ сингулярних значень.

8. Оформити результати у вигляді звіту.

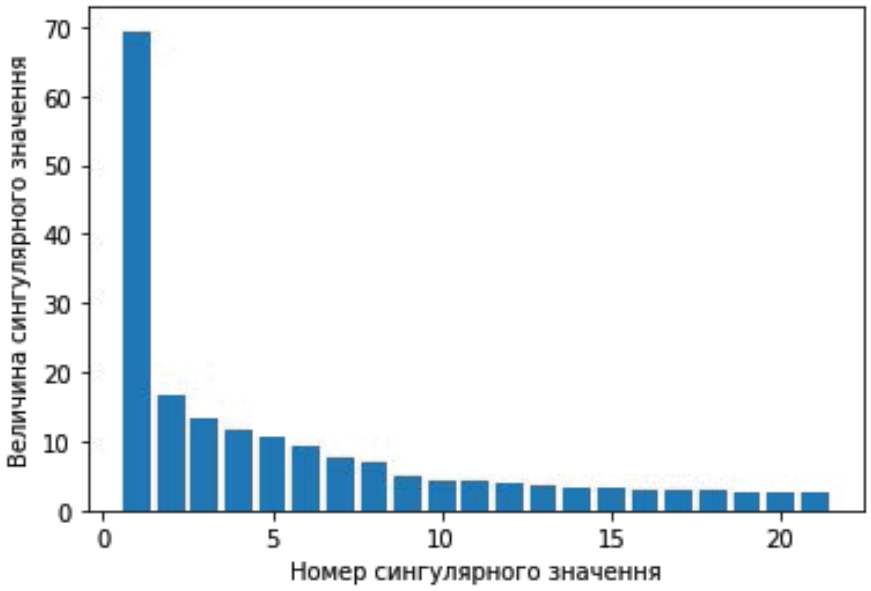
5.2.7 Приклад подання результатів

1. Вихідне зображення розміром 198×255 пікселів.

Відцифровуємо зображення – створюємо матрицю $A_{n \times d}$.

2. Проводимо SVD-розкладання матриці $A = U\Sigma V^T$.

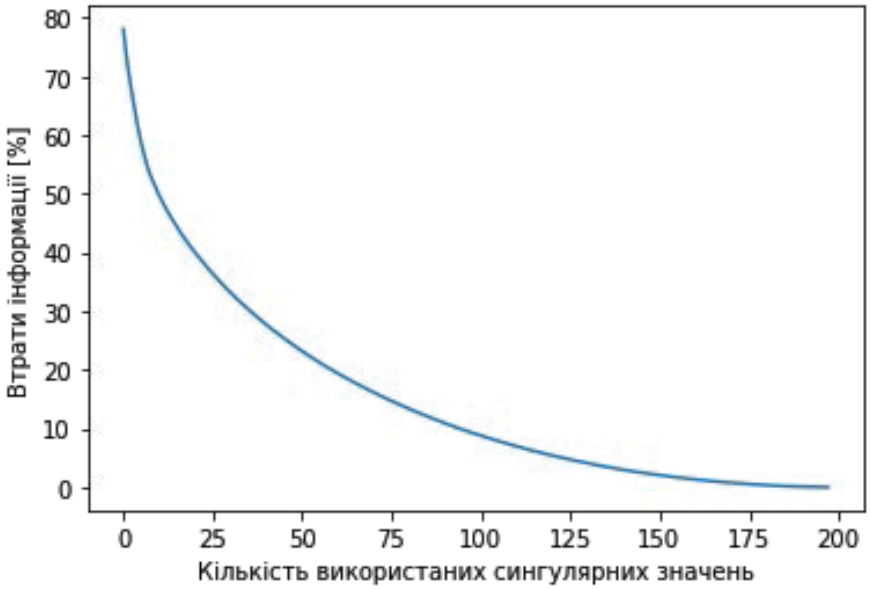
3. Візуалізуємо одержані сингулярні значення.



4. Рахуємо кількість втраченої інформації у відсотках за

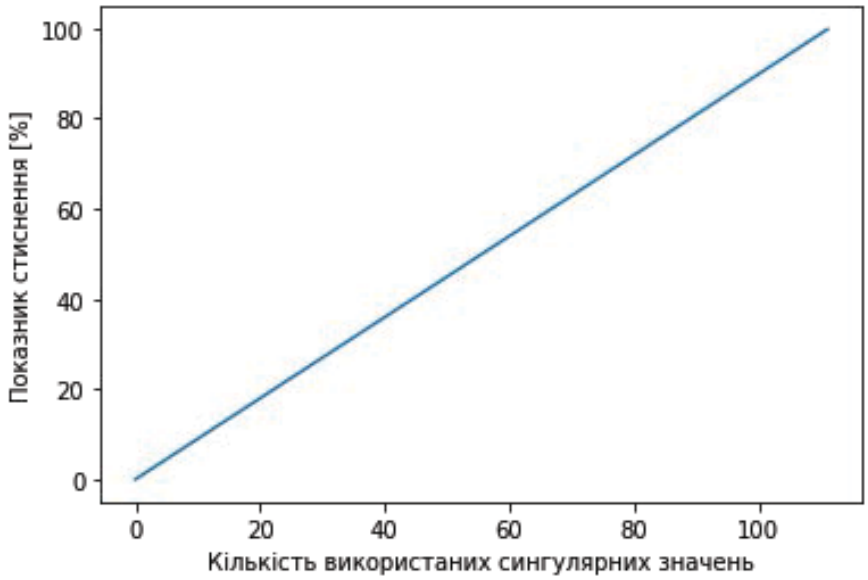
умови використання лише $m = 1, 2, 3, \ldots, N$ сингулярних значень.

$$I_m = \left(1 - \frac{\sum\limits_{i=1}^{m} \sigma_i}{\sum\limits_{j=1}^{d} \sigma_j} \right) \times 100\ \%.$$



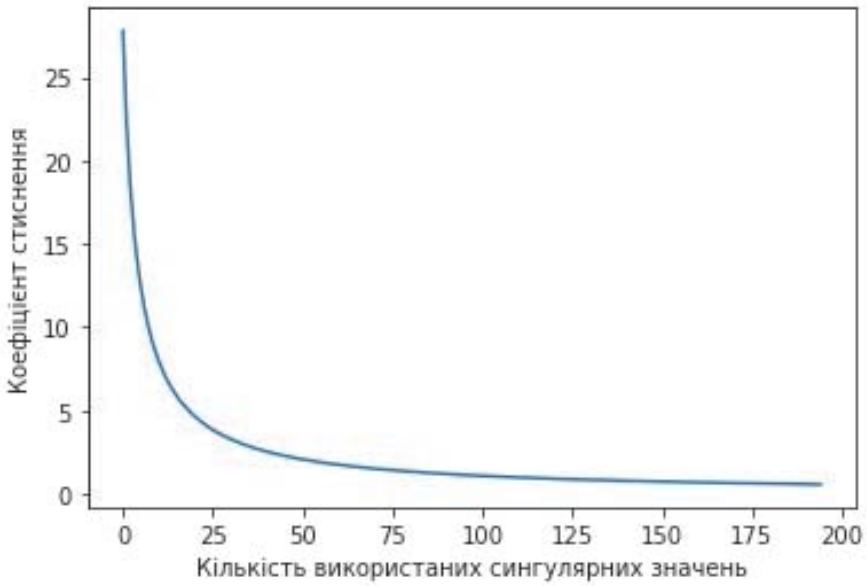
5. Рахуємо показник стиснення у відсотках за умови використання лише $m = 1, 2, 3, \ldots, N$ сингулярних значень.

$$S = \frac{m \times (n + d + 1)}{n \times d} \times 100\ \%.$$

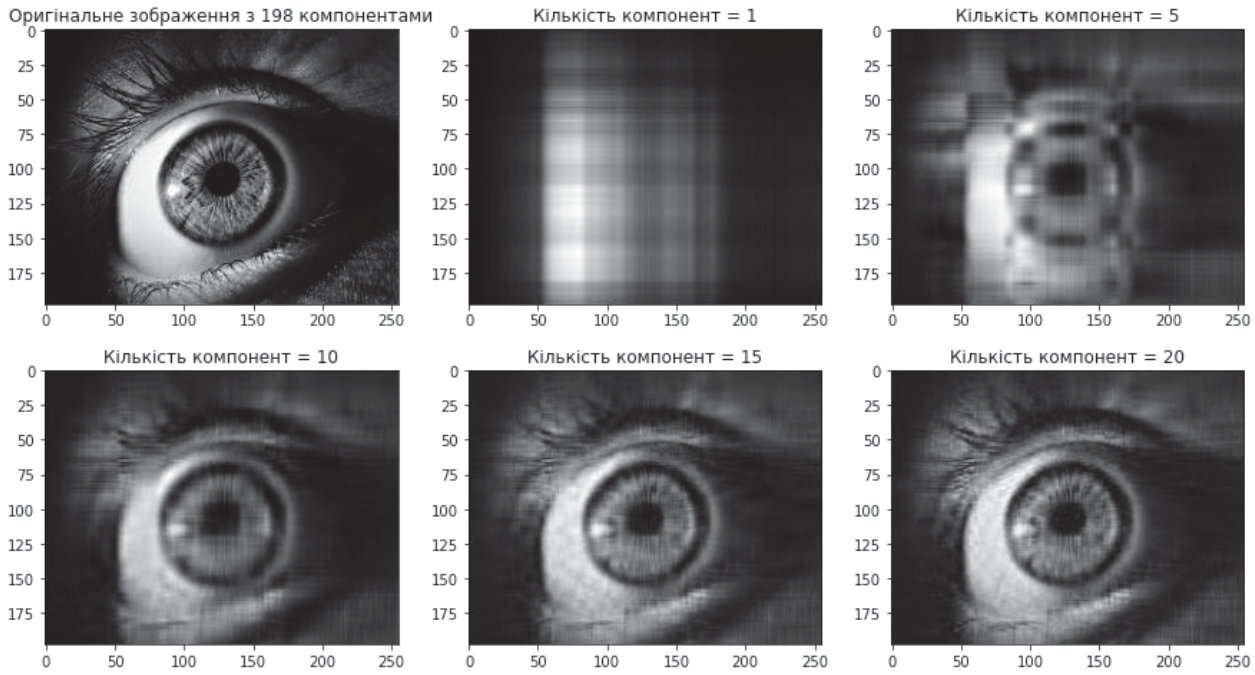


6. Рахуємо коефіцієнт стиснення за умови використання лише $m = 1, 2, 3, \ldots, N$ сингулярних значень.

$$K = \frac{n \times d}{m \times (n + d + 1)}.$$



7. Відтворюємо малюнок для $m = 1, 5, 10, 15, 20$ сингулярних значень.



Розділ 6

Навчання без учителя: задачі кластеризації

У цьому розділі буде надано основний теоретичний матеріал щодо методів кластеризації даних. Алгоритми кластеризації дають змогу розділити об'єкти якщо класи заздалегідь не зазначені, а кластери повинні бути сформовані за подібністю елементів. Буде розглянуто два основні й найбільш часто застосовувані методи:

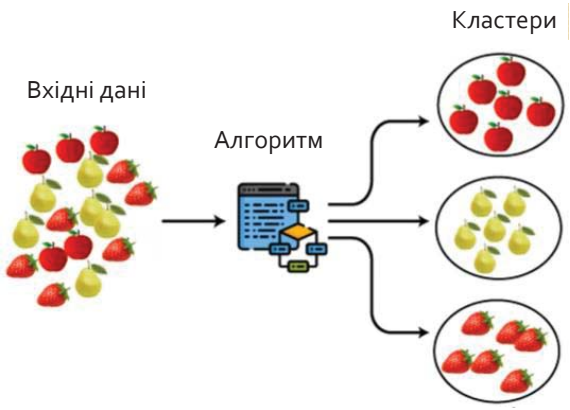
- метод k-середніх (k-Means);
- метод DBscan.

Для кожного методу буде наведено приклад його застосування, поставлено завдання для практичної роботи й наведено приклад подання результатів.

6.1 Кластеризація

Кластеризація – це класифікація, але без заздалегідь відомих класів.

Машина сама шукає подібні об'єкти та об'єднує їх у кластери. Кількість кластерів можна задати заздалегідь або довірити машині. Подібність об'єктів машина визначає за тими ознаками, що ми їй задали – у кого багато подібних характеристик, тих вона об'єднує в один кластер.



- Відмінний приклад кластеризації — маркери на картах в Інтернеті. Коли ви шукаєте всі ресторани азіатської кухні у великому місті, машині доводиться групувати їх у кружечки з цифрою, інакше браузер зависне, намагаючись намалювати мільйон маркерів.

- Більш складні приклади кластеризації можна згадати в програмах «iPhoto» або «Google Photos», що знаходять обличчя людей на фотографіях і групують їх в альбоми. Програма не знає, як звуть ваших друзів, але може відрізнити їх за характерними рисами обличчя.

Кластеризацію сьогодні застосовують для:

- сегментації ринку (типів покупців);
- об'єднання близьких точок на карті;
- стиснення зображень;
- аналізу й розмічення нових даних;
- детектування аномальної поведінки.

Популярні алгоритми: метод k-середніх (k-means), Mean-Shift, DBSCAN.

6.1.1 Постановлення задачі

Дано:

- X – простір об'єктів;
- $X^\ell = \{x_1, \dots x_\ell\}$ – навчальна вибірка;
- $\rho: \ X \to [0, \infty)$ – функція відстані між об'єктами.

Знайти:

- Y – множину кластерів;
- $a: \ X \to Y$ – алгоритм кластеризації.

Властивості кластерів:

- кожен кластер складається з близьких за ознаками об'єктів;
- об'єкти різних кластерів суттєво різні.

6.1.2 Некоректність задачі кластеризації

Розв'язок задачі кластеризації принципово неоднозначний:

- точної постановки задачі кластеризації здебільшого немає;
- є багато критеріїв якості кластеризації: визначення оптимальної кількості кластерів;
- є багато евристичних методів кластеризації;
- зазвичай кількість кластерів заздалегідь не відома;
- результат кластеризації значно залежить від метрики ρ для розрахунку відстані між об'єктами.

Приклад: скільки тут кластерів?

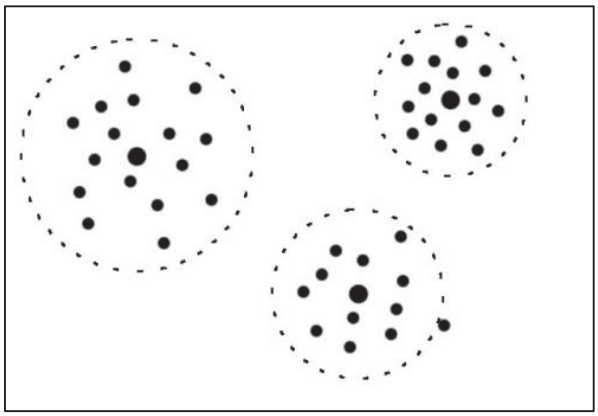
6.1.3 Завдання кластеризації

- Спростити подальше оброблення даних: поділити множину X^ℓ на групи подібних об'єктів для подальшого аналізу кожної групи окремо (завдання класифікації, регресії, прогнозування).
- Скоротити об'єм даних, що зберігається: залишити по одному з типових представників кожного кластеру (центр мас) – завдання стиснення даних.
- Виділити нетипові об'єкти (викиди), що не підходять до жодного з кластерів (завдання одно класової класифікації).
- Побудувати ієрархію множини об'єктів (класифікація рослин і тварин).

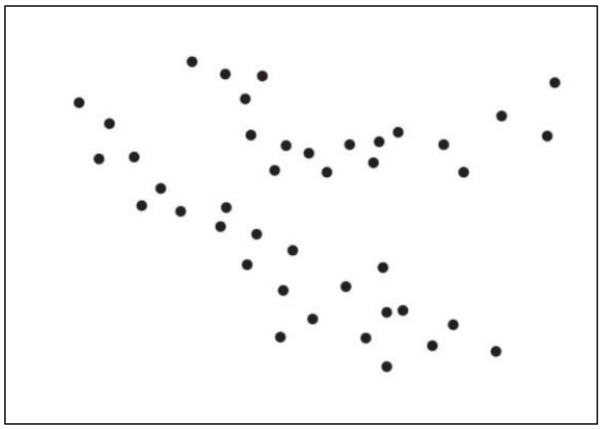
6.1.4 Типи структур кластерів

- Кожен метод кластеризації має свої обмеження й виділяє кластери лише декількох типів.
- Поняття «тип кластерної структури» залежить від методу і також не має формального визначення.

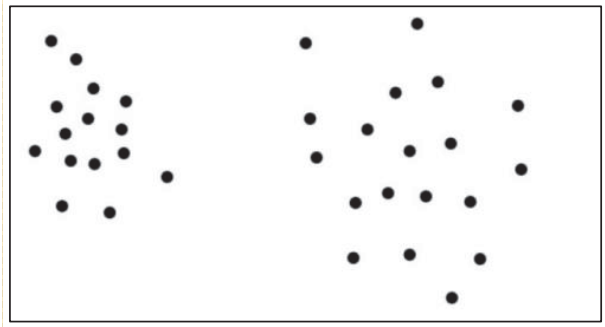
Кластери з центрами



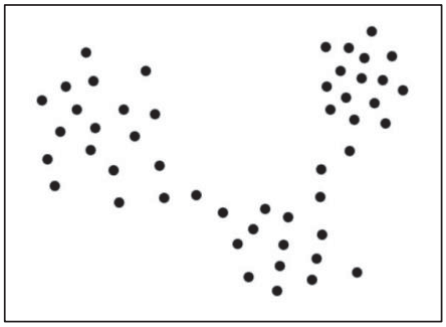
Стрічкові кластери



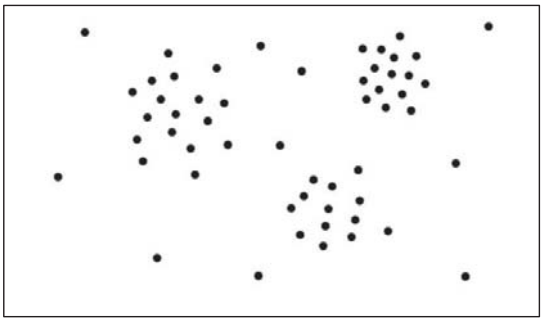
Внутрішньокластерні відстані менші за міжкластерні



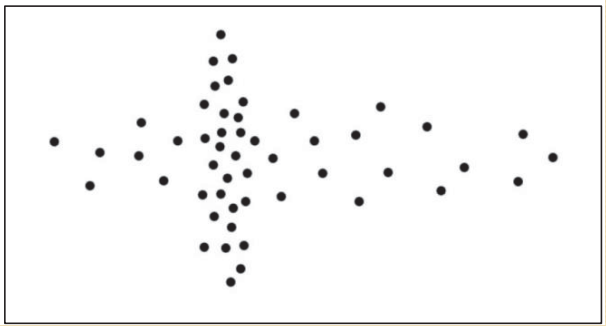
Існування перемичок між кластерами



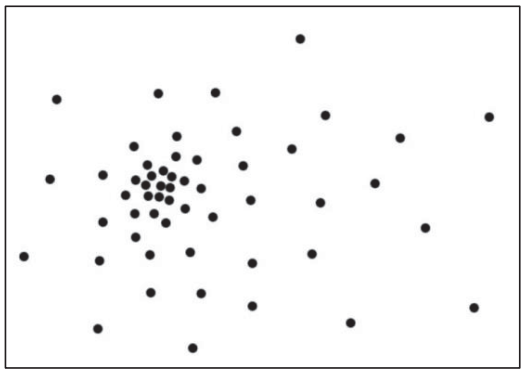
Розріджений набір об'єктів

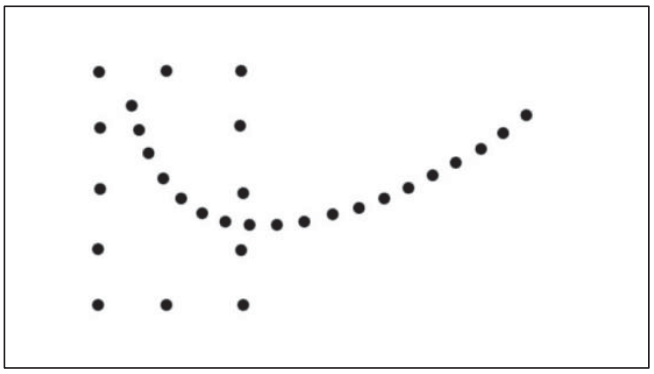


Кластери, що перекриваються



Кластерів узагалі може не бути





6.1.5 Методи кластеризації

Кластеризацією зазвичай вважають такий набір кластерів, що містить усі об'єкти набору даних. Додатково можна розглянути відношення між кластерами. Наприклад, ієрархію вкладеності кластерів один в одного. Узагальнено можна виділити такі кластеризації:

- *жорстку*: кожен об'єкт або належить кластеру, або ні;
- *м'яку* (нечітку): кожен об'єкт належить кожному кластеру до певної міри. Наприклад, це ймовірність належності до кластеру.

Серед них виділяють декілька додаткових.

- *Жорстке розбиття на кластери*: кожен об'єкт належить лише одному кластеру.
- *Жорстке розбиття на кластери з викидами*: об'єкт може не належати жодному кластеру й розглядається як викид.
- *Кластери з перетином*: об'єкт може належати більше ніж одному кластеру.
- *Ієрархічна кластеризація*: якщо об'єкт належить нащадку, то він також належить предку.
- *Підпросторова кластеризація*: хоча кластери можуть перетинатися, проте в межах визначеного підпростору вони не перетинаються.

Типові кластерні моделі

- *Моделі зв'язності*: наприклад, ієрархічна кластеризація, або таксономія, будується на основі відстані між вузлами.
- *Центроїдні моделі*: наприклад, метод k-середніх (k-means) презентує кожен кластер єдиним усередненим вектором.
- *Статистичні моделі*: кластери будуються, ґрунтуючись на статистичних розподілах.
- *Моделі, що базуються на щільності*: наприклад, у DBSCAN та OPTICS кластери визначаються як зв'язані області відповідної щільності в просторі даних.
- *Групові моделі*: деякі алгоритми не забезпечують удосконаленої моделі для своїх результатів, а просто описують групування об'єктів.
- *Графові моделі*: поняття клітинки (така підмножина вершин, у якій кожна пара вершин з'єднана ребром) у графі слугує прототипом кластера.
- *Нейронні моделі*: найбільш відомою моделлю нейронної мережі з навчанням без учителя є нейронна мережа Кохонена.

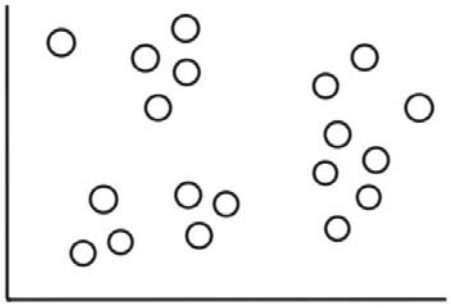
6.2 Метод k-середніх (k-means Method)

Цей метод належить до алгоритмів неієрахічного поділу (Partitioning algorithms), які декомпонують набір даних, що складається з n спостережень, на k груп (кластерів) із заздалегідь невідомими параметрами. Під час цього виконується пошук центроїдів – максимально віддалених один від одного центрів згущень точок із мінімальним розкидом усередині кожного кластера.

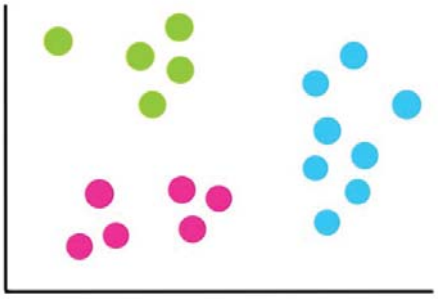
6.2.1 Покрокова реалізація методу

Постановка завдання й попередня обробка даних

Нехай є набір даних із $n = 19$ значень, що виглядають приблизно так:



Тепер припустимо, що ми знаємо, що ці дані можна поділити на три порівняно очевидні категорії. Виглядатиме це так:



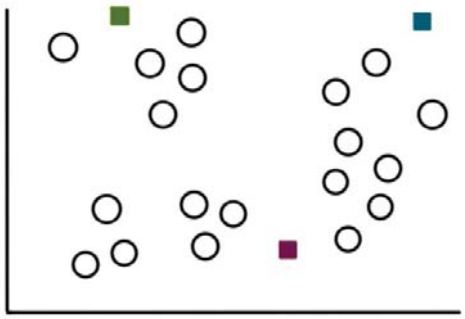
Завдання – застосувати алгоритм кластеризації k-means, щоб зробити цю категоризацію.

Крок 1: вибираємо кількість кластерів, k

Кількість кластерів, які ми хочемо розпізнати, і є k у k-means. У нашому разі, тому що ми припустили, що всього три кластери, $k = 3$.

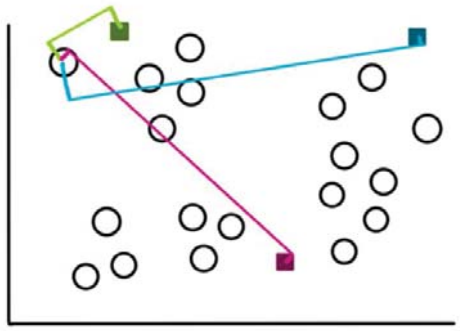
Крок 2: вибираємо k випадкових значень

Процес виявлення кластерів починаємо з вибору трьох випадкових значень (не обов'язково, щоб вони були нашими даними). Ці точки зараз працюватимуть як центроїди, або центри кластерів, що ми збираємося згрупувати.



Крок 3: створення k кластерів

Щоб створити кластери, ми почнемо вимірювати відстань між кожним значенням наших даних до кожного з трьох центроїдів, а потім додамо його до найближчого кластера. Для значення, взятого як приклад, відстані виглядатимуть приблизно так.



Для розрахунку відстані між об'єктами використовуємо стандартне визначення декартової відстані:

$$d = \sqrt{\sum_{i=1}^{m}(x_i - c_{ij})^2},$$

де x_i – координата в m-вимірному просторі (значення кожної ознаки); c_{ij} – координата кожного центроїда. У 2-мірному просторі для знаходження відстані між двома точками маємо

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Використовуючи цю формулу, ми повторюємо процес зі значеннями, що залишилися, після цього кластери будуть виглядати так.

Крок 4: обчислюємо новий центроїд кожного кластера

Тепер, маючи всі три кластери, ми знаходимо нові центроїди для кожного з них за формулою

$$(c_{1j}, c_{2j}, \ldots, c_{mj}) = \left(\frac{1}{n_j} \sum_{i=1}^{n_j} x_{1i}, \frac{1}{n_j} \sum_{i=1}^{n_j} x_{2i}, \ldots, \frac{1}{n_j} \sum_{i=1}^{n_j} x_{mi} \right),$$

де c_{kj} – k-та координата j-го кластера;
x_{ki} – k-та координата i-го об'єкта, що належить до j-го кластера;
n_j – кількість об'єктів у j-му кластері.
З огляду на це, нові центроїди матимуть такий вигляд.

Крок 5: знову класифікуємо об'єкти до кожного з центроїдів

Деякі об'єкти належатимуть до іншого центроїда (кластера). Розраховуємо суму квадратів внутрішньокластерних відстаней до центру кластера (within-cluster sum of squares, WCSS):

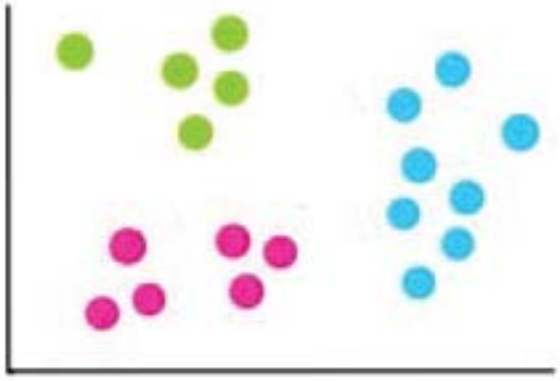
$$Q = \sum_{j=1}^{k} \sum_{i=1}^{n} \min \left(||x_i^j - c_j|| \right)^2. \tag{6.1}$$

Крок 6: повторюємо кроки 4–5

Кроки 4 і 5 повторюють доти, поки алгоритм не стабілізується, тобто поки об'єкти не перестануть переходити від одного центроїда (кластера) до іншого. Говорячи формально, мета алгоритму – мінімізувати функцію втрат (6.1).
Давайте припустимо, що подальші чотири ітерації будуть такими:

Ітерація 1

Ітерація 3



Ітерація 2

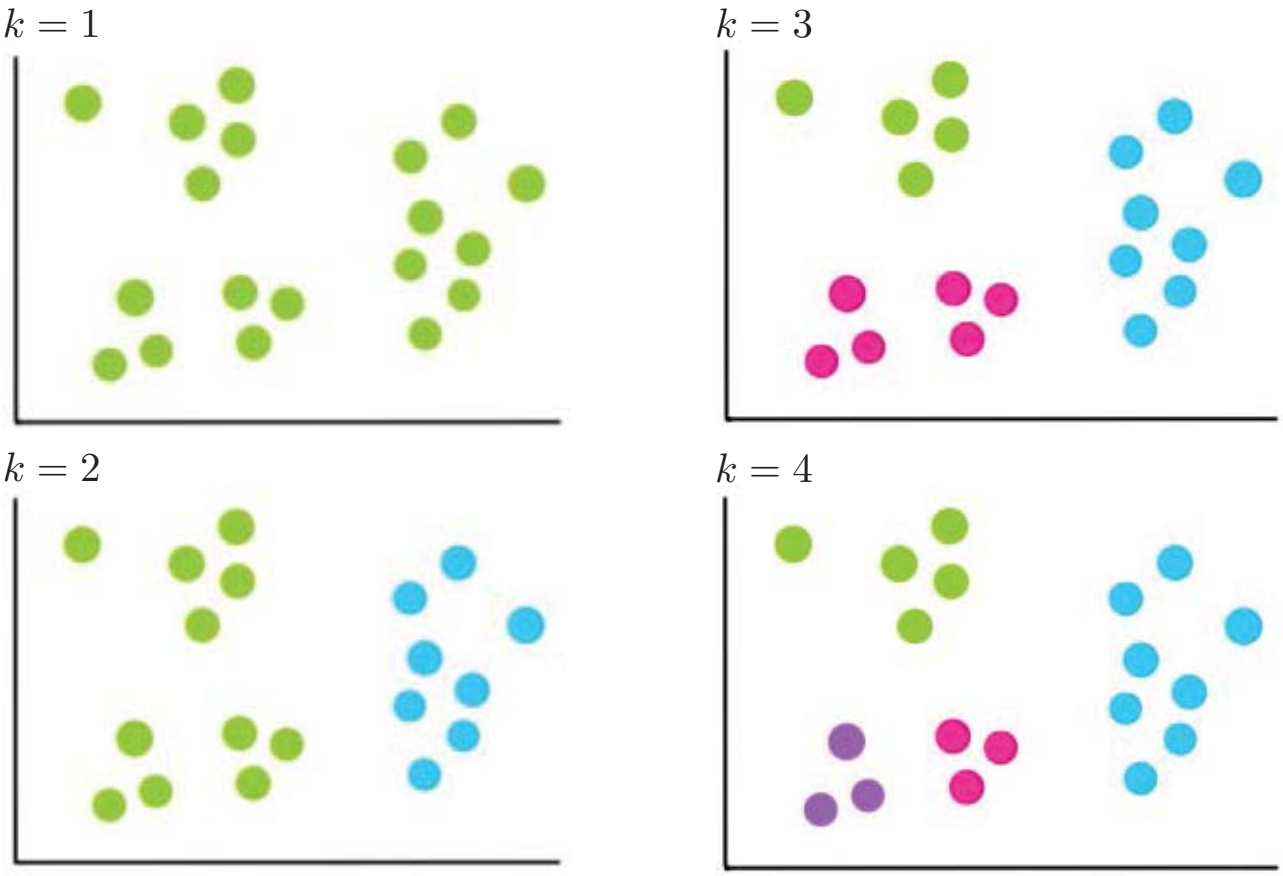
Ітерація 4

На двох останніх ітераціях можемо помітити, що кластери не змінюються. Це означає, що алгоритм зійшовся, тому ми зупиняємо процес. Потім вибираємо кластери з найменшим WCSS. Це будуть кластери на останніх двох ітераціях, тому вони стають нашими фінальними кластерами.

6.2.2 Вибір кількості кластерів

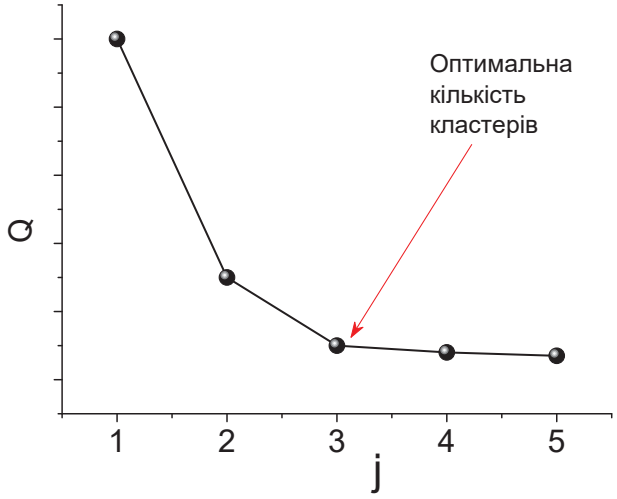
Один із способів вибору кількості кластерів є *експертний метод* – вибір кількості кластерів залежно від знання про предметну сферу (domain knowledge).

У разі кластеризації методом k-середніх кількість кластерів найчастіше оцінюють за допомогою «методу ліктя» (*elbow method*). Він передбачає багаторазове циклічне виконання алгоритму зі збільшенням кількості кластерів, що вибираються.



Для кожного вибору одержуємо мінімальне значення суми квадратів внутрішньокластерних відстаней за формулою (6.1). За результатами досліджень будуємо графік залежності $Q(j)$.

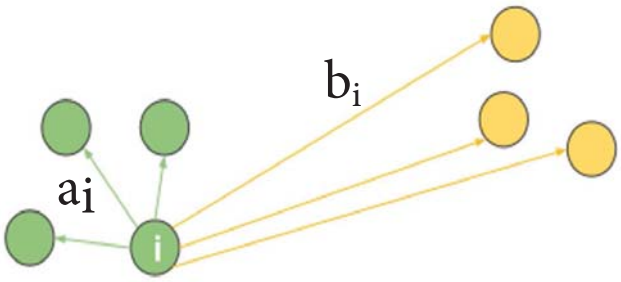
На рисунку наведено типову залежність $Q(j)$. Як можемо помітити, після того, як кількість кластерів досягає трьох, сума квадратів внутрішньокластерних відстаней перестає істотно зменшуватися. Це значення обумовлює оптимальну кількість кластерів.



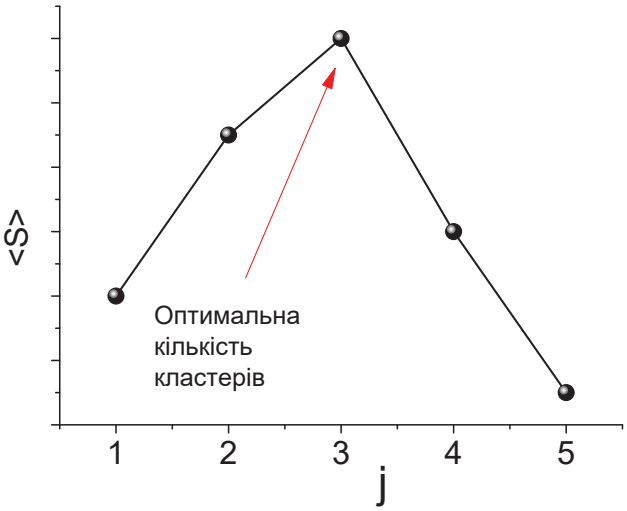
Іншим критерієм установлення оптимальної кількості кластерів є коефіцієнт «силуету» (*Silhouette Score*). Коефіцієнт «силуету» обчислюється за допомогою середньої внутрішньокластерної відстані a та середньої відстані до найближчого кластера b за кожним зразком:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)},$$

де a_i – середня відстань від об'єкта i до об'єктів свого кластера;
b_i – середня відстань від об'єкта i до об'єктів іншого кластера.



На рисунку наведено типову залежність $\langle S \rangle$. Як можемо помітити, середнє значення коефіцієнта «силуету» зростає до $k = 3$ та різко знижується зі збільшенням значень k. Тобто ми одержуємо виражений пік при $k = 3$.



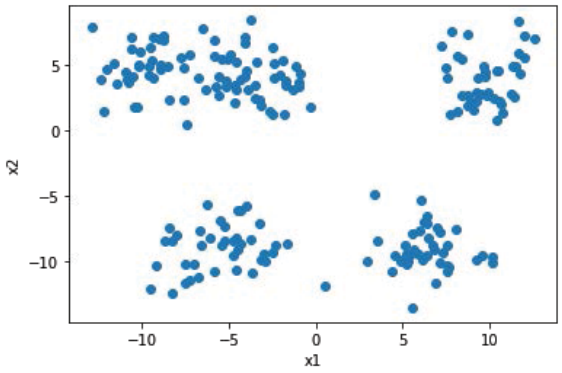
6.2.3 Постановлення задачі

Кластеризувати дані з використанням методу k-середніх.
Етапи виконання

1. Імпортувати/згенерувати розмічені/нерозмічені дані.
2. У разі кількості ознак $n > 2$ звести до $n = 2$ із використанням SVD та PCA.
3. Кластеризувати дані методом k-means за фіксованого значення кількості кластерів.
4. Візуалізувати одержані результати.
5. Установити оптимальну кількість кластерів із застосуванням «метода ліктя» та коефіцієнта «силуету». Порівняти одержані результати з розміченими даними (за наявності).
6. Порівняти результати з k-means зі sklearn.
7. Оформити результати у вигляді звіту.

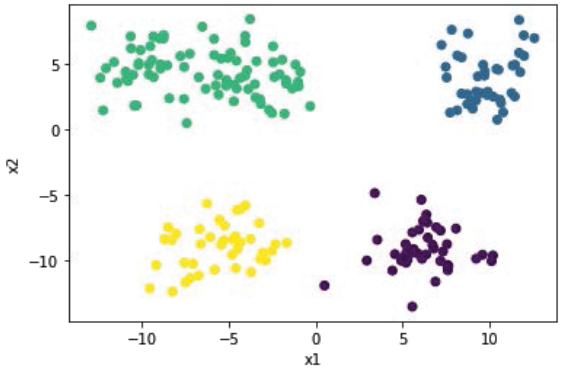
6.2.4 Приклад подання результатів

Імпортуємо/генеруємо вибірку з розміченими даними. Якщо кількість ознак $n > 2$, зводимо її до $n = 2$ (зменшуємо розмірність) за допомогою методів SVD та PCA.



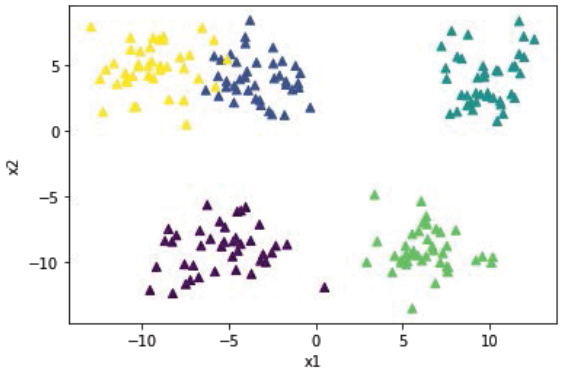
Візуалізуємо дані. Визначаємо (вибираємо) кількість кластерів, що на нашу думку, реалізуються у вибірці. Фіксуємо значення параметра k_0, що задає кількість кластерів.

Поділяємо всю вибірку на навчальну й тестову. Будуємо алгоритм кластеризації для $k = k_0$.



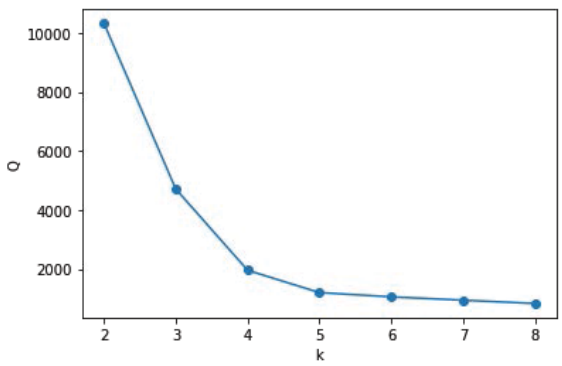
Візуалізуємо результати роботи алгоритму для $k_0 = 4$. У цьому разі кружками різного кольору позначено різні кластери.

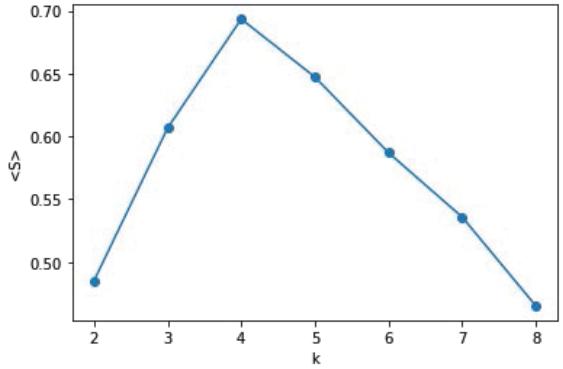
Візуалізуємо розмічені дані (за наявності) та рахуємо помилки кластеризації.



У розмічених даних існує 5 кластерів. З одержаних результатів робимо висновок про правильність кластеризації.

Оптимізуємо алгоритм. Рахуємо внутрішньокластерні відстані Q за «методом ліктя» та коефіцієнт «силуету».





Робимо висновок щодо оптимальної кількості кластерів у вибірці. У цьому разі $n = 4$.

Порівнюємо одержані результати з убудованим алгоритмом k-mean із бібліотеки sklearn.

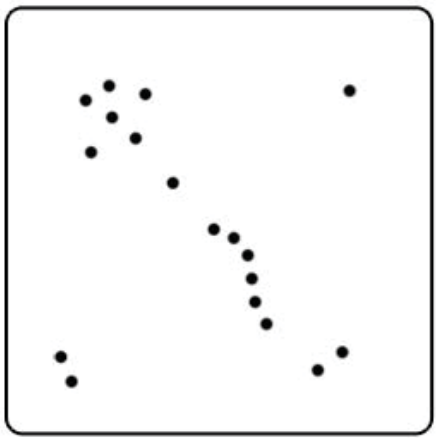
6.3 Метод DBSCAN

Маючи справу з просторовими кластерами різної щільності, розміру та форми, часто складно виявити групу точок. Завдання може бути ще складнішим, якщо дані містять шум і викиди. Для роботи з великими просторовими базами даних був запропонований метод DBSCAN (Density-based spatial clustering of applications with noise, густинний алгоритм просторової кластеризації з наявністю шуму), що, як випливає з назви, оперує щільністю даних. Як вхідні параметри він потребує радіус ϵ-околиці й кількість сусідів. Можна виділити три основні переваги використання алгоритму:

1. потребує мінімальних знань предметної галузі;
2. може виявляти кластери довільної форми;
3. ефективний для великих вибірок.

6.3.1 Інтуїтивне пояснення

У гігантській залі натовп людей святкує чийсь день народження. Хтось «тиняється» один, але більшість – із товаришами. Деякі компанії просто юрмляться, деякі – водять хороводи або танцюють ламбаду.



Завдання: поділити людей у залі на гурти.

Але як виділити групи такої різної форми та ще й не забути про одинаків? Спробуємо оцінити щільність юрби навколо кожної людини. Напевно, якщо щільність натовпу між двома людьми вища за певний поріг, то вони належать до однієї компанії. Справді, буде дивно, якщо люди, які ведуть «потяг», належатимуть до різних груп, навіть якщо щільність ланцюжка між ними змінюється в деяких межах.

- Будемо говорити, що поруч із деякою людиною зібрався натовп, якщо близько до неї стоять кілька інших людей. Отже, відразу зрозуміло, що потрібно задати два такі параметри.
  - Що означає «близько»? Візьмемо якусь інтуїтивно зрозумілу відстань. Скажімо, якщо люди можуть торкнутися голів ода одної, то вони перебувають близько (близько метра).
  - Скільки саме «кілька інших людей»? Припустимо, троє. Двоє можуть гуляти і просто так, але третій – безумовно зайвий.

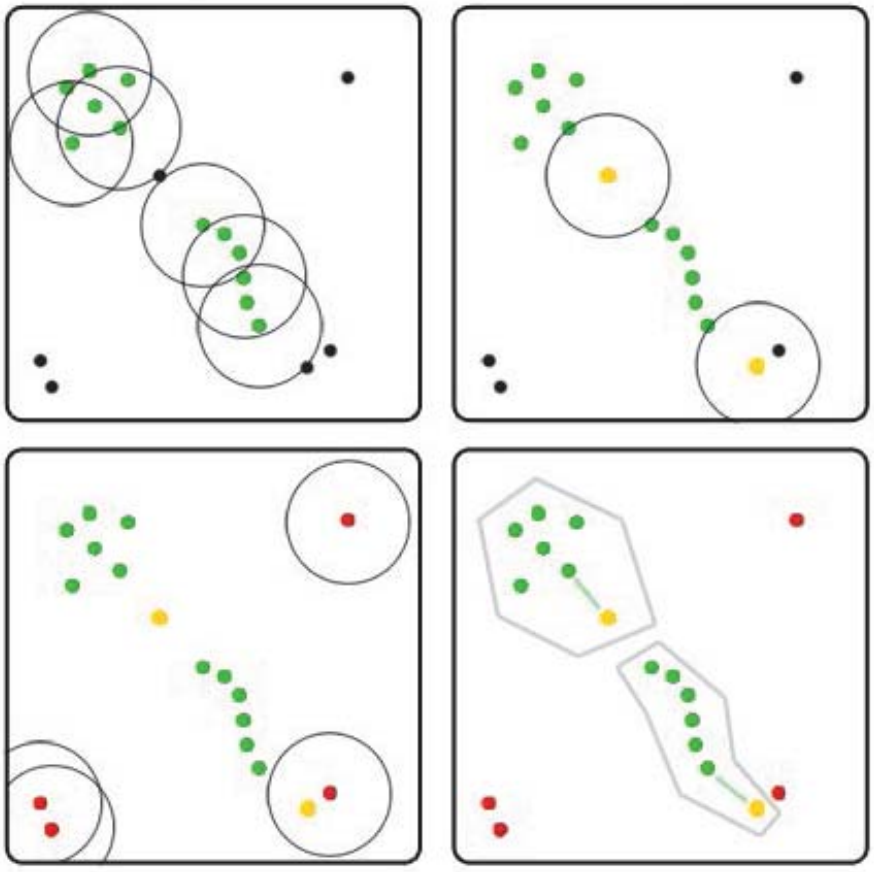
- Нехай кожен підрахує, скільки людей перебувають у радіусі метра від нього. Усі, хто має хоча б трьох сусідів, беруть у руки зелені прапорці. Тепер вони є корінними елементами, саме вони формують групи.

- Звернемося до людей, які мають менш ніж три сусіди. Виберемо тих, у яких принаймні один із сусідів має зелений прапорець, і вручимо їм жовті прапорці. Скажімо, вони перебувають на межі груп.

- Залишилися одинаки, у яких не лише немає трьох сусідів, а й жоден із них не тримає зелений прапорець. Роздамо їм червоні прапорці. Вважатимемо, що вони не належать жодній групі.

Отже, якщо від однієї людини до іншої можна створити ланцюжок людей із зеленими прапорцями, то ці дві людини належать до однієї групи. Вочевидь, ці скупчення розділені порожнім простором чи людьми з жовтими прапорцями. Можна їх пронумерувати: кожен у першій групі може досягти по ланцюжку рук кожного іншого першої групи, але нікого в групах 2, 3 тощо. Те саме для інших груп.

Якщо поруч із людиною з жовтим прапорцем є лише один сусід із зеленим, то він буде належати тій групі, до якої належить його сусід. Якщо таких сусідів кілька, й у них різні групи, то доведеться вибирати. Для цього можна скористатися різними методами – подивитися, хто із сусідів найближчий. Доведеться уникати крайових випадків.



Зазвичай немає сенсу помічати всіх корінних елементів натовпу відразу. Оскільки від кожного корінного елемента групи можна провести ланцюжок до кожного іншого, то все одно з якого починати обхід: рано чи пізно знайдеш усіх. У цьому разі краще підходить ітеративний варіант.

1. Підходимо до випадкової людини з натовпу.
2. Якщо поряд із нею менше три особи, заносимо її до списку можливих одинаків (викидів, шумів) і вибираємо когось іншого.
3. Інакше:
   а) виключаємо її зі списку людей, яких необхідно оминути;
   б) вручаємо цій людині зелений прапорець і створюємо нову групу, в якій вона поки що єдиний учасник;
   в) обходимо всіх її сусідів. Якщо її сусід уже в списку потенційних одинаків чи поруч із нею мало інших людей, то перед нами край натовпу. Для простоти можна відразу помітити його жовтим прапорцем, приєднати до групи

й продовжити обхід. Якщо сусід теж має зелений прапорець, то він не починає нову групу, а приєднується до вже створеної; крім того, ми додаємо до списку обходу сусідів сусіда. Повторюємо цей процес доти, поки список обходу не виявиться порожнім.

4. Повторюємо кроки 1–3, поки так чи інакше не обійдемо всіх людей.

5. Розбираємося зі списком викидів. Якщо на кроці 3 ми вже розкидали всіх крайових, то в ньому залишилися лише викиди-одинаки – можна відразу завершити. Якщо ні, необхідно якось розподілити людей, які залишилися в списку.

6.3.2 Формальний підхід

Уведемо кілька визначень.
Нехай задана деяка симетрична функція відстані $\rho(x, y)$ та константи ϵ і m. Зважаючи на це,

1) назвемо область $E(x)$, для якої $\forall y : \rho(x, y) \leq \epsilon$, ϵ-околиця об'єкта x;

2) кореневим об'єктом (core point) ступеня m називається об'єкт, ϵ-околиця якого містить щонайменше m об'єктів: $|E(x)| \geq m$;

3) об'єкт p безпосередньо щільно досягається з об'єкта q, якщо $p \in E(q)$ і q – кореневий об'єкт;

4) об'єкт p щільно-досяжний з об'єкта q, якщо $\exists p_1, p_2 \dots p_n, p_1 = q, p_n = p$, такі що $\forall i \in 1 \dots n-1 : p_{i+1}$ безпосередньо щільно-досяжний із p_i.

Виберемо будь-який кореневий об'єкт p із датасету, позначимо його й помістимо всіх його безпосередньо щільно-досяжних сусідів у список обходу. Тепер для кожного об'єкта q зі списку: позначимо його, і якщо він теж є кореневим, додамо всіх його сусідів до списку обходу. Кластери помічених точок, сформовані відповідно до цього алгоритму, є максимально заповнені (тобто їх не можна

розширити ще однією точкою, щоб задовольнялися умови) і зв'язні в сенсі досяжності. З огляду на це якщо ми обійшли не всі точки, можна перезапустити обхід з іншого кореневого об'єкта, і новий кластер не поглине попередній.

6.3.3 Нюанси застосування

За ідеальних умов DBSCAN може досягти складності $O(N)$, але не варто на це сподіватися. Якщо не перераховувати щоразу $E(x)$ точок, то очікувана складність – $O(N \log N)$. Найгірший випадок (погані дані) – $O(N^2)$. Наївні реалізації DBSCAN виділяють $O(N^2)$ пам'яті під матрицю відстаней – це явно надмірно.

DBSCAN із невипадковим правилом обробки крайових точок детермінований. Проте більшість реалізацій для прискорення роботи та зменшення кількості параметрів віддають крайові точки першим кластерам, що до них дотяглися. Наприклад, центральна жовта точка на зображенні вище в різних запусках може належати як нижньому, так і верхньому кластеру. Зазвичай це не впливає на якість роботи алгоритму, адже через граничні точки кластер усе одно не поширюється далі і ситуація, коли точка перескакує з кластера в кластер і «відкриває дорогу» до інших точок, неможлива.

DBSCAN самостійно не обчислює центри кластерів, але навряд це проблема, особливо враховуючи довільну форму кластерів. Проте DBSCAN автоматично визначає викиди, що важливо.

Співвідношення (m/ϵ^n), де n – розмірність простору, можна інтуїтивно розглядати як граничну щільність точок даних у сфері простору. Очікується, що за однакового співвідношення (m/ϵ^n) результати будуть приблизно однакові. Іноді це справді так, але є причина, чому алгоритму потрібно задати два параметри, а не один. По-перше, типова відстань між точками в різних датасетах різна: явно задавати радіус доводиться завжди. По-друге, важливу роль відіграють неоднорідності датасету. Чим більші m і ϵ, тим частіше алгоритм схильний «пробачати» варіації щільності в кластерах. З одного боку, це може бути корисно: неприємно побачити в кластері «дірки», де просто не вистачило даних. З іншого боку, шкідливо, коли між кластерами немає чіткої межі чи шум створює «міст» між скупченнями. У цьому разі DBSCAN запросто поєднає дві різні

групи. У балансі цих параметрів полягає складність застосування DBSCAN: реальні набори даних містять кластери різної щільності з межами різного ступеня розмитості. В умовах, за яких щільність деяких меж між кластерами більша або дорівнює щільності певних відокремлених кластерів, доводиться чимось жертвувати.

6.3.4 Налаштування параметрів

Є варіанти DBSCAN, здатні частково вирішити цю проблему. Ідея полягає в підлаштовуванні ϵ в різних галузях під час роботи алгоритму. На жаль, зростає кількість параметрів алгоритму.

Передбачено евристики для вибору m та ϵ. Найчастіше застосовують такий метод і його варіації.

1. Вибираємо m. Зазвичай використовують значення від 3 до 9. Чим неоднорідніший очікують датасет і чим вищий рівень шуму, тим більшим варто взяти m.

2. Обчислюємо середню відстань за m найближчими сусідами кожної точки. Тобто, якщо $m = 3$, потрібно вибрати трьох найближчих сусідів, скласти відстані й поділити на три.

3. Сортуємо знайдені значення за зростанням і виводимо на екран.

4. Одержуємо зростаючу залежність K-Distance Graph. Вибираємо ϵ десь у смузі, де відбувається найсильніший перегин. Чим більше ϵ, тим більшими вийдуть кластери, і тим менше їх буде.

У будь-якому разі, основні недоліки DBSCAN – нездатність з'єднувати кластери через прорізи й навпаки, здатність пов'язувати різні кластери через перемички. Частково тому зі збільшенням розмірності даних n виявляється більше місць, де можуть випадково виникнути отвори чи мости.

DBSCAN добре піддається модифікації, однією з яких є схрещення DBSCAN із k-means для прискорення. DBSCAN розпаралелюється, але це нетривіально зробити ефективно. Якщо перший процес виявив, що множина A – субкластер, а другий процес, що B – субкластер, і $A \cap B \neq \emptyset$, то A та B можна об'єднати. Проблема полягає в тому, щоб вчасно помітити, що перетин множин не порожній.

6.3.5 Підсумок

DBSCAN варто застосовувати, якщо:

- датасет є в міру великий з $N \approx 10^6$ (навіть $N \approx 10^7 - 10^8$,якщо під рукою оптимізована та розпаралена реалізація);
- наперед відома функція близькості, симетрична, бажано, не дуже складна;
- очікувані кластери складної форми: вкладені та аномальні кластери, згустки даних малої розмірності;
- щільність меж між згустками менша за щільність найменш щільного кластера (краще якщо кластери зовсім відокремлені один від одного);
- складність елементів датасета не має значення не має, проте їх повинно бути достатньо, щоб не виникало значних розривів у густині;
- кількість елементів у кластері може змінюватись як завгодно;
- кількість викидів не має значення (в розумних межах), якщо вони розсіяні у великому об'ємі.
- кількість кластерів не має значення.

6.3.6 Сфери застосування

DBSCAN має історію успішних застосувань. Наприклад, можна відзначити його використання в завданнях:

- виявлення соціальних гуртків;
- сегментування зображень;
- моделювання поведінки користувачів вебсайтів;
- попереднього оброблення для прогнозування погоди.

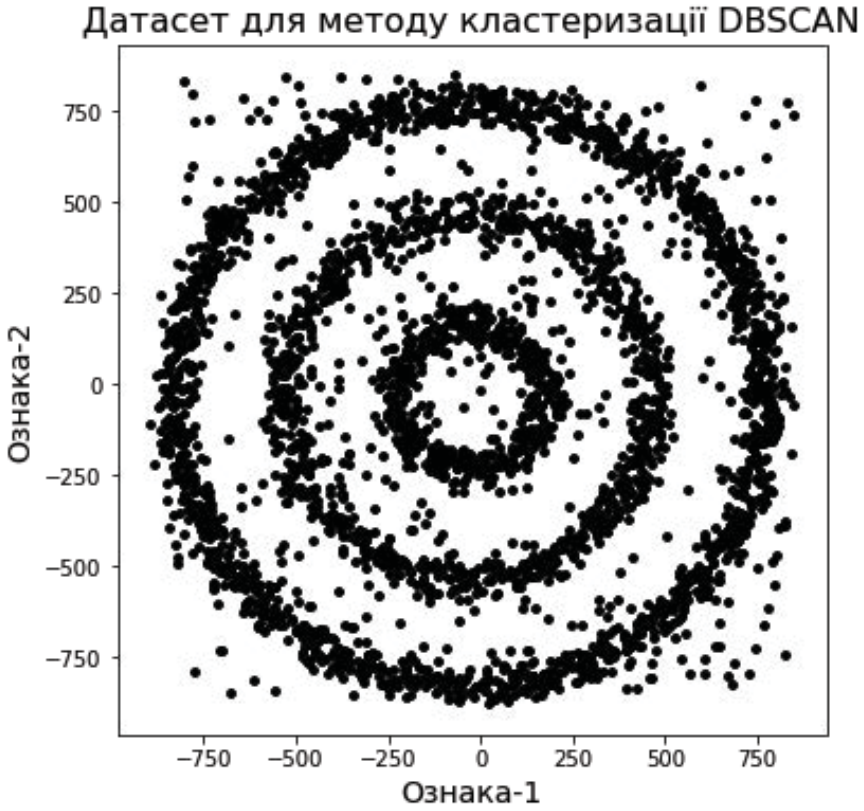
6.3.7 Поставлення задачі

Кластеризувати дані із застосуванням методу DBSCAN.
Етапи виконання

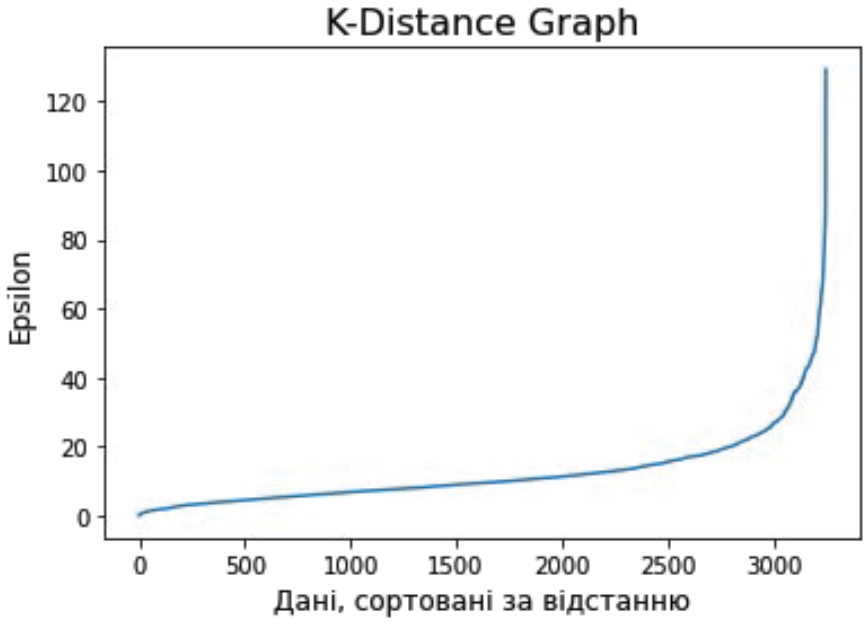
1. Імпортувати/згенерувати розмічені/нерозмічені дані.
2. У разі кількості ознак $n > 2$ звести до $n = 2$ із використанням SVD та PCA.
3. Зафіксувати кількість сусідів m, побудувати K-Distance Graph і визначити оптимальне значення щільності ϵ.
4. Провести кластеризацію методом DBSCAN за вибраних параметрів m та ϵ.
5. Візуалізувати одержані результати.
6. Порівняти результати з DBSCAN зі sklearn.
7. Оформити результати у вигляді звіту.

6.3.8 Приклад подання результатів

Імпортуємо/генеруємо вибірку з розміченими даними. Якщо кількість ознак $n > 2$, зводимо до $n = 2$ (зменшуємо розмірність) за допомогою методів SVD та PCA.

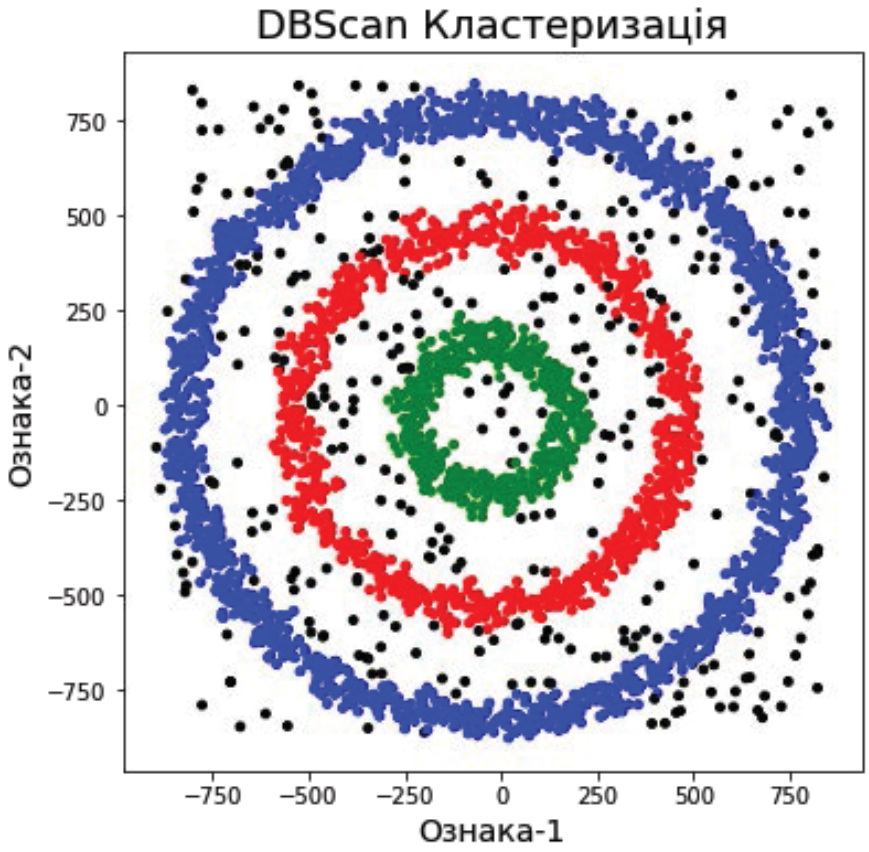


Фіксуємо кількість сусідів $m = 3$, і будуємо K-Distance Graph.



Визначаємо оптимальне значення щільності $\epsilon = 30$.

Кластеризуємо дані методом DBSCAN.
Візуалізуємо результати.



Порівнюємо одержані результати з убудованим класифікатором DSSCAN бібліотеки sklearn.

Розділ 7

Навчання без учителя: пошук асоціативних правил

У цьому розділі буде надано основний теоретичний матеріал щодо методів пошуку й виявлення асоціативних правил у потоці даних. Асоціативні правила дають змогу знаходити закономірності між пов'язаними подіями. Розглянуто принципи аналізу асоціативних правил, проблему виявлення узагальнених асоціативних правил і два основних та найбільш часто застосовуваних алгоритми:

- Apriori;
- Frequent Pattern-Growth Strategy (FPG).

Для кожного алгоритму наведено приклад його застосування, поставлено завдання для практичної роботи й наведено приклад подання результатів.

7.1 Уведення в аналіз асоціативних правил

Великі обсяги сучасних баз даних викликали стійкий попит на нові алгоритми аналізу даних. Одним із популярних методів виявлення знань стали алгоритми пошуку асоціативних правил.

Асоціативні правила дають змогу знаходити закономірності між пов'язаними подіями. Прикладом такого правила є твердження, що покупець, який купує «хліб», придбає і «молоко» з імовірністю 75 %. Перший алгоритм пошуку асоціативних правил був розроблений у 1993 р. співробітниками дослідницького центру IBM Almaden і називався AIS. На середину 90-х років минулого століття припав пік досліджень у цій галузі, і з того часу щороку з'являлося по кілька алгоритмів.

Завдання пошуку асоціативних правил уперше було застосовано для аналізу ринкового кошика. Асоціативні правила ефективно використовують у сегментації покупців за поведінкою під час купівлі, аналізі переваг клієнтів, плануванні розміщення товарів у супермаркетах, кроспродажах, адресному розсиланні. Проте сфера застосування цих алгоритмів не обмежується лише однією торгівлею. Їх також успішно застосовують в інших галузях: медицині, для аналізу відвідувань вебсторінок (Web Mining), тексту (Text Mining), даних із перепису населення, аналізу й прогнозування збоїв телекомунікаційного обладнання тощо.

7.1.1 Асоціативні правила (Association Rules)

Уперше завдання пошуку асоціативних правил було запропоновано для знаходження типових шаблонів покупок, здійснюваних у супермаркетах, тому іноді завдання пошуку асоціативних правил ще називають аналізом ринкового кошика (market basket analysis).

Нехай є база даних, що складається з купівельних транзакцій. Кожна транзакція – набір товарів, куплених покупцем за один візит. Таку транзакцію ще називають ринковим кошиком.

Постановка завдання

Нехай $I = \{i_1, i_2, i_3, \ldots, i_n\}$ – безліч (набір) товарів, що називають елементами; D – безліч транзакцій, де кожна транзакція T – набір

елементів з I, $T \subseteq I$. Кожна транзакція є бінарним вектором, де $t[k] = 1$, якщо i_k – елемент, наявний у транзакції, інакше $t[k] = 0$. Ми говоримо, що транзакція T містить X, деякий набір елементів з I, якщо $X \subset T$.

Підтримка (Support)
Підтримка свідчить про те, наскільки часто набір предметів з'являється в наборі даних. Підтримка набору X щодо Y визначається як частка транзакцій, що містять X і Y у наборі даних, від загального набору даних: частота зустрічі (підтримка, support) набору, що містить товар X та Y у вибірці D:

$$supp(X \cup Y) = \frac{\text{кількість транзакцій, що містять X і Y}}{\text{загальна кількість транзакцій}}.$$

Асоціативним правилом називається імплікація $X \Rightarrow Y$, де $X \subset I$, $Y \subset I$ і $X \cap Y = \oslash$. Правило $X \Rightarrow Y$ має підтримку s, якщо s % транзакцій із D містять $X \cup Y$, $supp(X \Rightarrow Y) = supp(X \cup Y)$.

Достовірність (confidence)
Достовірність (упевненість) правила показує, яка ймовірність того, що з X випливає Y. Правило $X \Rightarrow Y$ справедливе з достовірністю c, якщо c % транзакцій із D, що містять X, також містять Y:

$$conf(X \Rightarrow Y) = \frac{\text{кількість транзакцій, що містять X і Y}}{\text{кількість транзакцій, що містять X}},$$

або в математичному вигляді

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}.$$

Цікавість (lift)
Цікавість правила $lift(X \Rightarrow Y)$ – імовірність купівлі товару Y в разі купівлі товару X. На відміну від упевненості (достовірності) $conf(X \Rightarrow Y)$, у цьому разі враховують популярність товару Y:

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) \times supp(Y)}.$$

- Якщо цікавість $lift(X \Rightarrow Y) = 1$, то кореляції в наборі товарів немає.
- Якщо цікавість $lift(X \Rightarrow Y) > 1$, кореляція в наборі товарів позитивна, тобто ймовірність спільної купівлі товарів X та Y вища.
- Якщо цікавість $lift(X \Rightarrow Y) < 1$, кореляція в наборі товарів негативна, тобто спільна купівля товарів X та Y малоймовірна.

Приклад

Приклад бази даних із 5 транзакціями й 5 товарами

№ транзакції	Молоко	Хліб	Масло	Сік	Підгузки
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

З таблиці можна зробити, наприклад, такі висновки:

- підтримка набору (молоко $\cup$ хліб) становить $2/5 = 0,4$ (40 %);
- підтримка набору (масло $\cup$ хліб) становить $1/5 = 0,2$ (20 %);
- підтримка набору (сік $\cup$ підгузки) становить $1/5 = 0,2$ (20 %);
- правило {масло, хліб} $\Rightarrow$ {молоко} має значущість $0,2/0,2 = 1$, що означає, що в 100 % випадків споживач, який купив масло та хліб, також купив молоко;
- якщо куплений {хліб}, то з імовірністю 67 % буде куплене {молоко}, хоча обидва товари купляють з імовірністю 40 %.

Отже, метою аналізу є встановлення таких залежностей: якщо в транзакції зустрівся деякий набір елементів X, то на підставі цього можна зробити висновок про те, що інший набір елементів Y

також повинен з'явитися в цій транзакції. Виявлення таких залежностей дає змогу знаходити дуже прості й інтуїтивно зрозумілі правила.

Алгоритми пошуку асоціативних правил призначені для знаходження всіх правил $X \Rightarrow Y$, водночас підтримка й достовірність цих правил повинні бути вищими за деякі наперед задані порогові значення, що називаються відповідно мінімальною підтримкою ($MinSupp$) і мінімальною достовірністю ($MinConf$).

Завдання знаходження асоціативних правил охоплює два підзавдання:

1. знаходження всіх наборів елементів, що задовольняють поріг $MinSupp$ і такі набори елементів називаються такими, що часто зустрічаються;

2. генерацію правил наборів елементів, знайдених згідно з п. 1. із достовірністю, що задовольняє поріг $MinConf$.

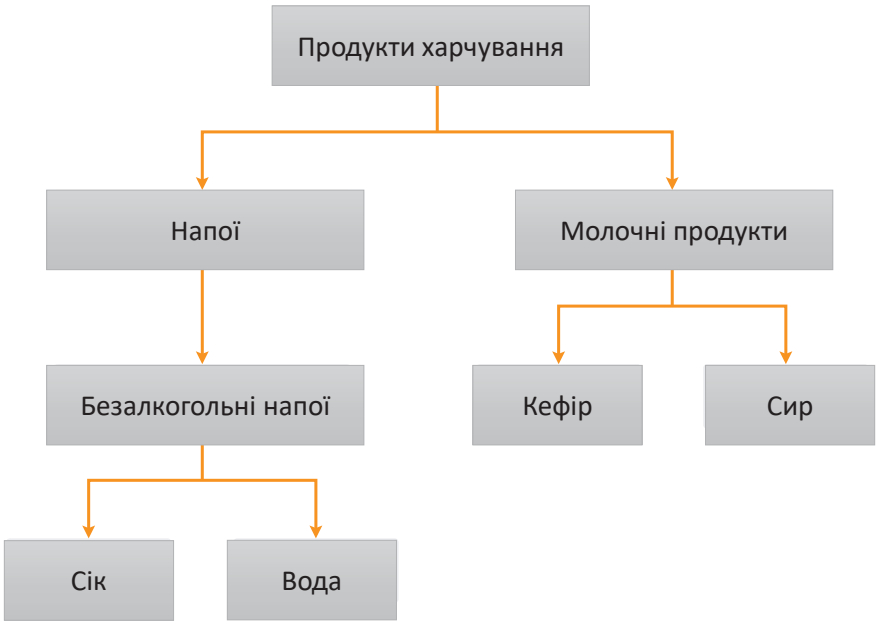
Значення параметрів мінімальна підтримка $MinSupp$ та мінімальна достовірність $MinConf$ вибирають так, щоб обмежити кількість знайдених правил. Якщо підтримка має велике значення, то алгоритми будуть знаходити правила, добре відомі аналітикам або настільки очевидні, що немає сенсу проводити такий аналіз.

З іншого боку, низьке значення підтримки приводить до генерації величезної кількості правил, що, звісно, потребує істотних обчислювальних ресурсів. Проте більшість цікавих правил характеризується саме низьким значенням порогу підтримки. Водночас занадто низьке значення підтримки спричиняє створення статистично необґрунтованих правил.

Пошук асоціативних правил – зовсім не очевидне завдання, як може здатися на перший погляд. Одна з проблем – алгоритмічна складність знаходження наборів елементів, що часто зустрічаються, оскільки зі зростанням кількості елементів експоненційно зростає кількість потенційних наборів елементів.

7.1.2 Узагальнені асоціативні правила (Generalized Association Rules)

Під час пошуку асоціативних правил зазвичай припускають, що аналізовані елементи однорідні. Повертаючись до аналізу ринкового кошика, це товари, що мають абсолютно однакові атрибути, крім назви. Проте можна доповнити транзакцію інформацією про те, в яку товарну групу входить товар, і побудувати ієрархію товарів. Наведемо приклад такого угруповання (таксономії), як ієрархічну модель.



Таксономія – результат класифікації й угруповання складних систем зазвичай репрезентованих у вигляді ієрархічної структури. Виділені для дослідження елементи та групи об'єктів підсистеми називаються таксонами. Таксономія може розглядатися як форма уявлення знань. З математичної точки зору ієрархічна таксономія є деревоподібною структурою класифікацій для конкретного набору об'єктів. У верхній частині цієї структури знаходиться єдина класифікація (кореневий вузол), застосовувана до всіх об'єктів. Вузли нижчих рівнів ієрархії дають детальніші класифікації.

Нехай нам дано базу транзакцій D і відомо до яких груп (таксонів) належать елементи. З огляду на це можна формувати з правил, що зв'язують групи з групами, правила, що зв'язують окремі елементи з групами і т. д.

Наприклад, якщо Покупець купив товар із групи «Безалкогольні напої», то він купить і товар із групи «Молочні продукти» або «Сік» і «Молочні продукти». Ці правила називають узагальненими асоціативними правилами.

Уведення додаткової інформації про угруповання елементів у вигляді ієрархії дасть такі переваги:

- допоможе встановити асоціативні правила як між окремими елементами, так і між різними рівнями ієрархії (групами);
- окремі елементи можуть мати недостатню підтримку, але загалом група може задовольняти поріг $MinSupp$.

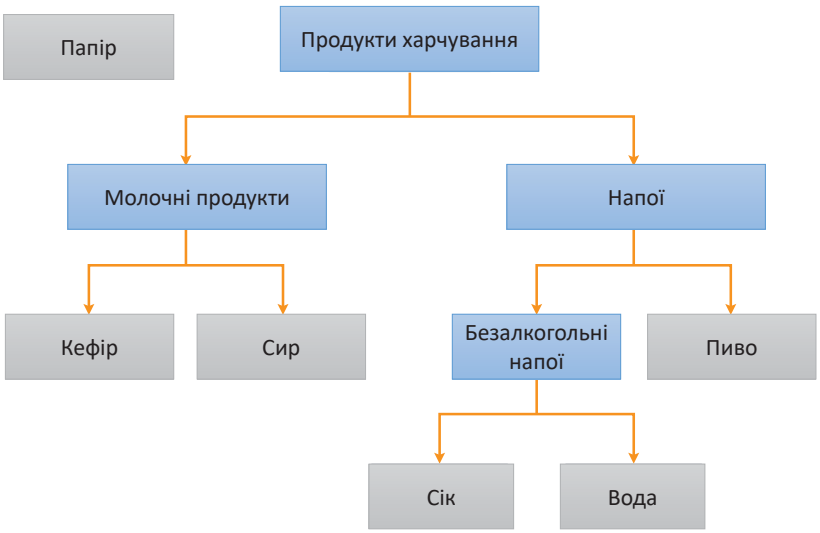
Для знаходження таких правил можна використовувати будь-який із вищезгаданих алгоритмів. Для цього кожну транзакцію потрібно доповнити всіма предками кожного елемента, що входить до неї. Проте пряме застосування цих алгоритмів неминуче приводить до таких проблем:

- елементи на верхніх рівнях ієрархії прагнуть до значно більших значень підтримки порівняно з елементами на нижніх рівнях;
- з додаванням у транзакції груп збільшиться кількість атрибутів і розмірність вхідного простору, що ускладнює завдання, а також приводить до створення більшої кількості правил;
- поява надлишкових правил, що суперечать визначенню узагальненого асоціативного правила, наприклад, «Сік» $\Rightarrow$ «Прохолодні напої» – потрібно видалять такі надлишкові правила.

Для знаходження узагальнених асоціативних правил зазвичай використовують спеціалізовані алгоритми, що усувають вищеописані проблеми.

Групувати елементи можна не лише за входженням у певну товарну групу, а й за іншими характеристиками, наприклад, ціною (дешево, дорого), брендом тощо.

Методи пошуку узагальнених правил під час обчислення використовують інформацію про угруповання елементів (таксономію), що дає змогу значно розширити коло завдань, виконуваних алгоритмами пошуку асоціативних правил. Прикладом узагальненого асоціативного правила може бути висловлювання: «Якщо людина купила сир, то вона, найімовірніше, купить товар із групи Хлібобулочні вироби». Основною відмінністю узагальнених асоціативних правил від асоціативних є те, що одержувані правила містять у собі елементи, які є предками елементів, що входять до безлічі транзакцій.



Розглянемо приклад ієрархії товарів і товарних груп. Маючи таку ієрархію, можна одержати, наприклад, такі правила: «Якщо покупець купив Сік, то він, найімовірніше, захоче купити Кефір»; «Якщо покупець купив Молочні продукти, він, найімовірніше, захоче купити Воду». Тобто в одержуваних правилах, як в умові, так і в наслідку, можуть бути елементи, що знаходяться на різних рівнях таксономії.

Уведення інформації про належність елементів транзакцій до тієї чи іншої групи може надати такі переваги.

1. Будуть виявлені асоціативні правила як між окремими елементами, так і між різними рівнями ієрархії.

2. За певних умов окремі елементи можуть мати дуже маленьку підтримку, але, значення підтримки всієї групи, до якої належить цей елемент, може бути більшим за поріг мінімальної підтримки. Це приводить до раніше не виявлених потенційно цікавих правил, побудованих на елементах нижнього рівня ієрархії.

3. Уведення інформації про угруповання елементів може використовуватися для відсікання «нецікавих» правил.

Визначення

Нехай $I = \{i_1, i_2, \dots i_m\}$ – це безліч елементів; I – ліс спрямованих дерев; дуги в I – залежність між елементами; елементи, що належать I, розміщені в певній ієрархії. Якщо є дуга від a до b, то кажуть, що a – предок b, а b – нащадок a (a – це узагальнення b). Нехай є безліч транзакцій D, де кожна транзакція T – безліч елементів (подій), що відбулися одночасно: $T \subseteq I$.

Визначення

Розширеною транзакцією називається транзакція, розширена предками всіх елементів, що входять до цієї транзакції.

Визначення

Узагальненим асоціативним правилом називається імплікація $X \Rightarrow Y$, де $X \subset I$, $Y \subset I$ і $X \cap Y = \oslash$ і жоден з елементів набору Y не є предком жодного елемента, що належить до X, і елементи, які входять до цього правила, можуть бути на будь-якому рівні таксономії. Підтримку й достовірніст підраховують так само, як у разі асоціативних правил (див. попередній підрозділ).

Будемо називати

- $\widehat{x}$ предком x;
- x нащадком $\widehat{x}$.

Завдання знаходження закономірностей, що є узагальненими асоціативними правилами на зразок $X \Rightarrow Y$ проблематичне з огляду на необхідність виявлення «зайвих» узагальнених асоціативних правил. Для вирішення цієї проблеми розглянемо таку властивість правила, як рівень інтересу. Нехай $Pr(X)$ – це ймовірність того, що всі елементи з X утримуватимуться в одній розширеній транзакції. Зважаючи на це, $supp(X \cup Y) = Pr(X \cup Y)$ і $conf(X \Rightarrow Y) = Pr(Y|X)$. Якщо підтримка $[x, y]$ більша за значення мінімальної підтримки $MinSupp$, то й підтримка $[\widehat{x}, y]$ і підтримка $[x, \widehat{y}]$ й підтримка $[\widehat{x}, \widehat{y}]$ буде більшою за поріг мінімальної

підтримки. Проте, якщо достовірність правила $X \Rightarrow Y$ більша за мінімальну достовірність $MinConf$, лише правило $X \Rightarrow \widehat{Y}$ гарантовано матиме достовірність більшу, ніж мінімальна. *Підтримка елемента, взятого з внутрішнього рівня ієрархії, не дорівнює сумі підтримки елементів, що є безпосередніми нащадками цього елемента.*

Визначення «цікавих» правил

Багато узагальнених асоціативних правил, що можуть бути знайдені, є потенційно нецікавими (приблизно 20 %–70 %). Для визначення того, які правила є «цікавими», а які ні, визначимо такий параметр, як рівень інтересу (зацікавленості).

Нехай $\widehat{Z}$ – це предок Z, де Z і $\widehat{Z}$ – безлічі елементів, що входять до ієрархії ($Z, \widehat{Z} \subseteq I$). $\widehat{Z}$ є предком Z, лише якщо $\widehat{Z}$ можна одержати із Z шляхом підміни одного чи кількох елементів їх предками. Якщо розглядати ієрархію на рисунку вище, то прикладом можуть бути ці дві множини: $Z = \{$Сік, Кефір, Папір$\}$, $\widehat{Z} = \{$Напої, Молочні продукти, Папір$\}$. Будемо називати правила $\widehat{X} \Rightarrow Y$, $X \Rightarrow \widehat{Y}$, $\widehat{X} \Rightarrow \widehat{Y}$ предками правила $X \Rightarrow Y$.

Визначення

Правило $\widehat{X} \Rightarrow Y$ є найближчим предком правила $X \Rightarrow Y$, якщо не існує такого правила $X' \Rightarrow Y'$, коли $X' \Rightarrow Y'$ – предок правила $X \Rightarrow Y$, а правило $\widehat{X} \Rightarrow \widehat{Y}$ – предок правила $X' \Rightarrow Y'$. Подібні визначення можна дати й для правил: $\widehat{X} \Rightarrow Y$ та $X \Rightarrow \widehat{Y}$.

Розглянемо правило $X \Rightarrow Y$ і візьмемо $Z = X \cup Y$. Зауважимо, що підтримка для правила $X \Rightarrow Y$ буде такою самою як для набору Z: $sup(X \Rightarrow Y) = supp(Z)$. Позначимо $E_{\widehat{Z}[Pr(Z)]}$ як очікуване значення $Pr(Z)$, що дає $Pr(\widehat{Z})$, де $\widehat{Z}$ є предком Z. Нехай $Z = \{z_1, \ldots, z_n\}$ та $\widehat{Z} = \{\widehat{z_1}, \ldots, \widehat{z_j}, z_{j+1}, \ldots, z_n\}$, $1 \leq j \leq n$, де $\widehat{z_i}$ є предком для z_i. Зважаючи на це можна визначити

$$E_{\widehat{Z}[Pr(Z)]} = \frac{Pr(z_1)}{Pr(\widehat{z_1})} \times \cdots \times \frac{Pr(z_j)}{Pr(\widehat{z_j})} \times Pr(\widehat{Z})$$

як очікуване значення $Pr(Z)$ для заданого набору елементів $\widehat{Z}$.

Аналогічно можна визначити $E_{\widehat{X}\Rightarrow\widehat{Y}}[Pr(Y|X)]$ як очікуване значення достовірності правила $X \Rightarrow Y$ щодо правила $\widehat{X} \Rightarrow \widehat{Y}$. Нехай $Y = \{y_1, \ldots, y_n\}$ та $\widehat{Y} = \{\widehat{y_1}, \ldots, \widehat{y_j}, y_{j+1}, \ldots, y_n\}$, $1 \leq j \leq n$, де $\widehat{y_i}$ є предком для y_i. З огляду на це можна визначити

$$E_{\widehat{X}\Rightarrow\widehat{Y}[Pr(Y|X)]} = \frac{Pr(y_1)}{Pr(\widehat{y_1})} \times \cdots \times \frac{Pr(y_j)}{Pr(\widehat{y_j})} \times Pr(\widehat{Y}|\widehat{X}).$$

Зазначимо, що $E_{\widehat{X}\Rightarrow Y}[Pr(Y|X)] = Pr(Y|\widehat{X})$.

Визначення

Правило $X \Rightarrow Y$ називається R-цікавим щодо правила-предка $\widehat{X} \Rightarrow \widehat{Y}$, якщо підтримка правила $X \Rightarrow Y$ у R разів більша за очікувану підтримку правила-предка $\widehat{X} \Rightarrow \widehat{Y}$, або якщо достовірність правила $X \Rightarrow Y$ у R разів більша за очікувану достовірність правила-предка.

Визначення

Правило називається цікавим, якщо воно не має предків або є R цікавим щодо всіх своїх найближчих предків.

Визначення

Правило називається частково цікавим, якщо воно не має предків або є R-цікавим щодо будь-якого свого найближчого предка.

Отже, після одержання правил, що задовольняють мінімальну достовірність, необхідно вилучити всі нецікаві правила.

Приклад

Розглянемо ієрархію товарів, подану на рисунку вище. Нехай у результаті роботи алгоритму ми одержали правила, наведені в першій таблиці, а підтримка елементів що входять до них, зазначена в другій таблиці. Рівень інтересу візьмемо $R = 1.3$.

№ правила	Текст правила	Підтримка, %
1	Сік ⇒ Молочні продукти	10
2	Безалкогольні напої ⇒ Кефір	15
3	Сік ⇒ Кефір	9

Елемент	Підтримка, %
Напої	7
Безалкогольні напої	5
Сік	3
Молочні продукти	6
Кефір	3

Із першої таблиці можна зробити висновок, що правило 2 є ближнім предком правила 3, оскільки набір «Безалкогольні напої» – предок для набору «Сік». Крім того, правило 1 також є ближнім предком правила 3 оскільки набір «Молочні продукти» – предок для набору «Кефір». Отже, розглянемо правило 3 і визначимо, є це правило цікавим, частково цікавим чи ні. Іншими словами, нам необхідно перевірити нерівність

$$Pr[\text{Сік} \Rightarrow \text{Кефір}] > E_{\widehat{\text{Сік}} \Rightarrow \widehat{\text{Кефір}}}[Pr(\text{Сік} \Rightarrow \text{Кефір})] \times R.$$

Використовуючи правило 2 як предок до правила 3, порахуємо очікувану підтримку:

$$\begin{aligned} E_{\text{Безалк.напої} \Rightarrow \text{Кефір}} &= \frac{Pr(\text{Сік})}{Pr(\text{Безалк.напої})} \times Pr(\text{Безалк.напої} \Rightarrow \text{Кефір}) \\ &= \tfrac{3}{5} \times 15 = 9 \end{aligned}.$$

Оскільки підтримка правила 3 дорівнює 9, то нерівність $9 > 9 \times 1,3$ не виконується і правило 3 не є цікавим. Далі використаємо правило 1 як предка до правила 3 та порахуємо очікувану підтримку:

$$\begin{aligned} E_{\text{Сік} \Rightarrow \text{Мол.прод.}} &= \frac{Pr(\text{Кефір})}{Pr(\text{Мол.прод.})} \times Pr(\text{Сік} \Rightarrow \text{Мол.прод.}) \\ &= \tfrac{3}{6} \times 10 = 5 \end{aligned}.$$

І знову нерівність $5 > 9 \times 1,3$ не виконується. Отже, правило 3 є не цікавим і його можна виключити зі списку асоціативних правил.

Один із перших алгоритмів, що ефективно виконує подібний клас завдань, – це алгоритм APriori. Крім нього, були розроблені й інші алгоритми: DHP, Partition, DIC та інші.

7.2 Алгоритм Apriori

Сучасні бази даних мають дуже великі розміри, що досягають гіга- та терабайтів, і тенденцію до подальшого збільшення. Тому для знаходження асоціативних правил потрібні ефективні масштабовані алгоритми, що дають змогу виконувати завдання за прийнятний час. **Apriori** – один із найпопулярніших алгоритмів пошуку асоціативних правил. Завдяки використанню властивості антимонотонності він здатний швидко обробляти великі обсяги даних. Розглянемо роботу алгоритму й особливості його реалізації. Щоб було можливо застосувати алгоритм, необхідно провести попередню обробку даних:

- звести всі дані до бінарного вигляду;
- змінити їх структуру.

Звичайний вигляд бази даних транзакцій

Номер транзакції	Найменування елемента	Кількість
1001	A	2
1001	D	3
1001	E	1
1002	A	2
1002	F	1
1003	B	2
1003	A	2
1003	C	2
...	...	...

Нормалізований вигляд бази даних транзакцій

TID	A	B	C	D	E	F	G	H	I	K	...
1001	1	0	0	1	1	0	0	0	0	0	...
1002	1	0	0	0	0	1	0	0	0	0	...
1003	1	1	1	0	0	0	0	0	1	0	...
...	...	...	...	...	...	...	...	...	...	...	...

Кількість стовпців у таблиці дорівнює кількості елементів, наявних у багатьох транзакціях. Кожен запис відповідає транзакції, де у відповідному стовпчику стоїть 1, якщо елемент наявний

у транзакції, і 0 – якщо ні. Водночас вихідний вигляд таблиці може бути відмінним від наведеного в таблиці, головне, щоб дані були перетворені до нормалізованого вигляду, інакше алгоритм не застосовуватиметься. Усі елементи таблиці впорядковані за алфавітом (якщо це числа, вони мають бути розміщені в числовому порядку). Це зроблено невипадково. Отже, дані перетворені, тепер можна почати опис самого алгоритму. Алгоритм Apriori працює у два етапи:

1. пошук наборів елементів, що часто зустрічаються;
2. визначення правил асоціації з вищезазначених частих наборів предметів.

Кількість елементів у наборі будемо називати розміром набору, а набір, що складається з k елементів, – k-елементний набір.

7.2.1 Властивість антимонотонності

Виявлення найпоширеніших наборів елементів – операція, що вимагає багато обчислювальних ресурсів і відповідно часу. Примітивний підхід до виконання цього завдання – простий перебір усіх можливих наборів елементів. Apriori використовує одну з властивостей підтримки, а саме: *підтримка будь-якого набору елементів не може перевищувати мінімальної підтримки будь-якого з його підмножин.* Наприклад, підтримка набору з трьох елементів {Хліб, Олія, Молоко} завжди буде меншою або дорівнюватиме підтримці 2-елементних наборів {Хліб, Олія}, {Хліб, Молоко}, {Олія, Молоко}. Справа в тому, що будь-яка транзакція, що містить {Хліб, Олія, Молоко}, також повинна містити {Хліб, Олія}, {Хліб, Молоко}, {Олія, Молоко}, водночас протилежне неправильно. Ця властивість називається **антимонотонністю** і сприяє зниженню розмірності простору пошуку. За її відсутності знаходження багатоелементних наборів було б фактично нездійсненним завданням з огляду на експоненційне зростання обчислень.

Властивості антимонотонності можна дати й інше формулювання: зі зростанням розміру набору елементів підтримка зменшується або залишається такою самою. З усього вищезазначеного

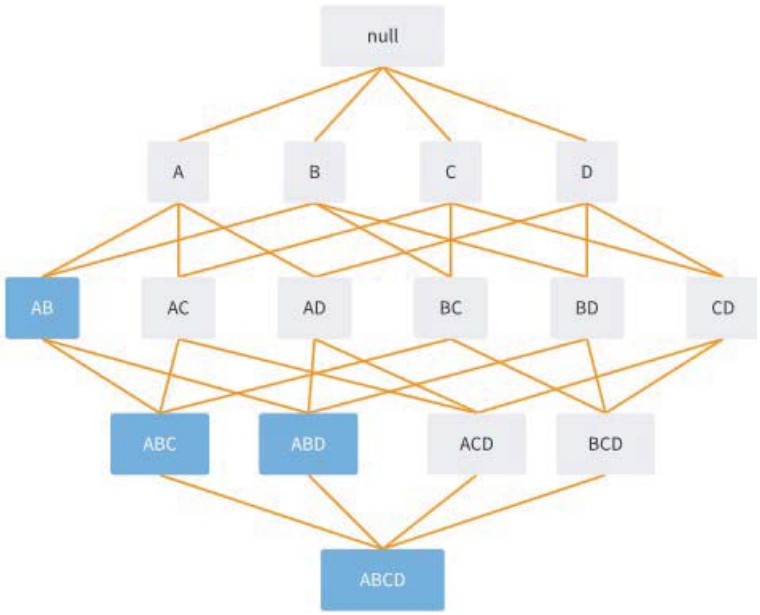
випливає, що будь-який k-елементний набір буде часто зустрічатися тоді та лише тоді, коли всі його $(k-1)$-елементні підмножини часто зустрічаються.

Наслідки властивості антимонотонності:

- якщо набір ψ часто зустрічається, то всі набори ϕ, що є його підмножинами, $\phi \subset \psi$ також часто зустрічаються;
- якщо ϕ зустрічається рідко, то всі набори ψ, які його охоплюють, $\psi \supset \phi$ також зустрічаються рідко;
- для будь-яких ψ, ϕ виконується співвідношення $supp(\phi \cup \psi) \leq supp(\phi)$.

Апріорна теорема: якщо набір елементів є частим, то всі його підмножини також повинні бути частими.

Усі можливі набори елементів з I можна репрезентувати у вигляді ґраток, що починаються з порожньої множини, потім на першому рівні – 1-елементні набори, на 2-му – 2-елементні і т. д. на k-рівні розміщені k-елементні набори, пов'язані з усіма своїми $(k-1)$-елементними підмножинами. Розглянемо рисунок, що ілюструє набір елементів $I = A, B, C, D$. Припустимо, що набір з елементів $\{A, B\}$ має підтримку, нижчу ніж заданий поріг, $supp(A \cup B) < MinSupp$ і відповідно не є частим. Зважаючи на це, згідно з властивістю антимонотонності всі його супермножини також не часто зустрічаються і відкидається вся ця гілка, починаючи з $\{A, B\}$, виділена синім кольором. Використання цієї евристики дає змогу значно скоротити простір пошуку.



7.2.2 Алгоритм Apriori – пошук у ширину

Алгоритм Apriori – послідовність кроків, яких варто дотримуватися, щоб знайти найчастіший набір елементів у конкретній базі даних. Ця техніка послідовно повторює кроки з'єднання та обрізки, поки не буде досягнуто найчастіший набір елементів. У проблемі зазначено мінімальний поріг підтримки $MinSupp$, або він передбачається користувачем.

Кроки в Apriori

1. На першій ітерації алгоритму кожен предмет береться як кандидат набору з одного предмета. Алгоритм буде рахувати випадки появи кожного елемента.

2. Визначається набір 1-елементних наборів предметів, поява яких задовольняє мінімальну підтримку. Лише тих кандидатів, для яких підтримка більша або дорівнює $MinSupp$, приймають на наступну ітерацію, а інших обрізають.

3. Виявляють часті набори предметів. Для цього набір із двох елементів формується шляхом формування групи з двох елементів, комбінуючи їх між собою.

4. Кандидати з двох елементів обрізаються з використанням порогового значення $MinSupp$. Тепер у таблиці будуть двоелементні набори з мінімальною підтримкою.

5. Наступна ітерація сформує триелементні набори, використовуючи кроки об'єднання та обрізки. Ця ітерація буде додержуватися властивості антимонотонності. Якщо всі підмножини з двох елементів будуть частими, надмножина буде частою, інакше вона буде обрізана.

6. Процедура зупиняється у разі максимальної кількості елементів в наборі (максимальне значення товарів у кошику).

Приклад

Розглянемо простий приклад для наочного ілюстрування принципу роботи алгоритму Apriori. Нехай є база записів купівлі різних товарів $I_1, \ldots, I_5$ для кожної транзакції $T_1, \ldots, T_6$ у такому вигляді.

Транзакція	Перелік товарів
T_1	I_1, I_2, I_3
T_2	I_2, I_3, I_4
T_3	I_4, I_5
T_4	I_1, I_2, I_4
T_5	I_1, I_2, I_3, I_5
T_6	I_1, I_2, I_3, I_4

Установимо значення мінімальної підтримки $MinSupp = 0.5$. Порахуємо підтримку для кожного товару (однокомпонентного набору). Можемо помітити, що підтримка товару I_5 менша за порогове значення. Отже, цей набір (що містить у собі лише товар I_5) є не частим, а відповідно всі набори, які його охоплюють, будуть не частими. Тому в подальшому всі набори, що містять товар I_5, не розглядаємо.

Елемент	Підтримка
I_1	4/6
I_2	5/6
I_3	4/6
I_4	4/6
I_5	2/6

Після цього побудуємо набори, що містять два різні товари. З одержаної таблиці робимо висновок про те, що набори $\{I_1, I_4\}$ та $\{I_3, I_4\}$ не відповідають критерію $MinSupp$, тому вони видаляються.

Елемент	Підтримка
I_1, I_2	4/6
I_1, I_3	3/6
I_1, I_4	2/6
I_2, I_3	4/6
I_2, I_4	3/6
I_3, I_4	2/6

Сформуємо набори товарів, що містять три різні товари, й порахуємо підтримку для цих наборів. Можемо помітити, що лише набір елементів $\{I_1, I_2, I_3\}$ відповідає критерію $MinSupp$, тому лише цей набір можна вважати частим.

Елемент	Підтримка
I_1, I_2, I_3	3/6
I_1, I_2, I_4	1/6
I_1, I_3, I_4	1/6
I_2, I_3, I_4	2/6

Обмежимося максимальною кількістю товарів у кошику, що дорівнює 3, і не будемо аналізувати наборів з більшою кількістю товарів.

Переходимо до другого кроку в алгоритмі Apriori, а саме: побудови асоціативних правил та аналізу їх достовірності. З одержаних результатів щодо частих наборів можна сформулювати наведені правила. Тепер для встановлення тих правил, що можна вважати достовірними (впевненими), зафіксуємо значення мінімальної достовірності $MinConf = 0.75$ і порахуємо достовірність кожного з одержаних шести правил.

- $\{I_1, I_2\} \Rightarrow \{I_3\}$
- $\{I_1, I_3\} \Rightarrow \{I_2\}$
- $\{I_2, I_3\} \Rightarrow \{I_1\}$
- $\{I_1\} \Rightarrow \{I_2, I_3\}$
- $\{I_2\} \Rightarrow \{I_1, I_3\}$
- $\{I_3\} \Rightarrow \{I_1, I_2\}$

- $Conf(\{I_1, I_2\} \Rightarrow \{I_3\}) = \frac{Supp(\{I_1,I_2,I_3\})}{Supp(\{I_1,I_2\})} = \frac{3}{4} = 0,75.$
- $Conf(\{I_1, I_3\} \Rightarrow \{I_2\}) = \frac{Supp(\{I_1,I_2,I_3\})}{Supp(\{I_1,I_3\})} = \frac{3}{3} = 1,0.$
- $Conf(\{I_2, I_3\} \Rightarrow \{I_1\}) = \frac{Supp(\{I_1,I_2,I_3\})}{Supp(\{I_2,I_3\})} = \frac{3}{4} = 0,75.$
- $Conf(\{I_1\} \Rightarrow \{I_2, I_3\}) = \frac{Supp(\{I_1,I_2,I_3\})}{Supp(\{I_1\})} = \frac{3}{4} = 0,75.$
- $Conf(\{I_2\} \Rightarrow \{I_1, I_3\}) = \frac{Supp(\{I_1,I_2,I_3\})}{Supp(\{I_2\})} = \frac{3}{5} = 0,6.$
- $Conf(\{I_3\} \Rightarrow \{I_1, I_2\}) = \frac{Supp(\{I_1,I_2,I_3\})}{Supp(\{I_3\})} = \frac{3}{4} = 0,75.$

Отже, робимо висновок про те, що 5 із 6 правил задовольняють критерій $MinConf$, тому їх можна вважати достовірними. Водночас з імовірністю 100 % якщо в кошику знаходяться товари I_1 і I_3, буде взятий товар I_2.

7.2.3 Оптимізація

Набагато ефективніше використовувати підхід, що базується на зберіганні кандидатів у хеш-дереві. Внутрішні вузли дерева містять хеш-таблиці з вказівниками на нащадків, а листя – на кандидатів. Це дерево знадобиться для швидкого підрахунку підтримки для кандидатів. Хеш-дерево будується щоразу, коли формуються кандидати. Спочатку дерево складається лише з кореня, що є листком, і не містить жодних кандидатів-наборів. Щоразу, коли формується новий кандидат, він заноситься до кореня дерева, і так доти, поки кількість кандидатів у корені-листі не перевищить певний поріг. Щойно кількість кандидатів стає більшою за поріг, корінь перетворюється на хеш-таблицю, тобто стає внутрішнім вузлом, і йому створюються нащадки-листя. Усі приклади розподіляються по вузлах-нащадках згідно з хеш-значеннями елементів, що входять у набір, і т. д. Кожен новий кандидат хешується на внутрішніх вузлах, поки не досягне першого вузла-листка, де він і зберігатиметься, поки кількість наборів знову не перевищить порогу.

Використовуючи хеш-дерево, легко підрахувати підтримку для кожного кандидата. Для цього потрібно «пропустити» кожну транзакцію через дерево й збільшити лічильники тих кандидатів, чиї елементи також містяться в транзакції. На кореневому рівні хеш-функція застосовується до кожного елемента транзакції.

Далі, на другому рівні, хеш-функція застосовується до інших елементів. На k-рівні хешується k-елемент. І так доти, доки не досягнемо листка. Якщо кандидат, що зберігається в листку, є підмножиною транзакції, що розглядається, тоді збільшуємо лічильник підтримки цього кандидата на одиницю.

Після того як кожна транзакція з вихідного набору даних «пропущена» через дерево, можна перевірити, чи задовольняють значення підтримки кандидатів мінімальний поріг. Кандидати, для яких ця умова виконується, переносяться до розряду тих, що часто зустрічаються. Крім того, варто запам'ятовувати і підтримку набору, вона необхідна для вилученні правил. Ці дії застосовуються для знаходження $(k+1)$-елементних наборів і т. д. Після того як знайдені всі набори елементів, що часто зустрічаються, можна безпосередньо генерувати правила.

7.2.4 Підвищення ефективності оброблення популярних наборів

Одним із напрямів підвищення ефективності обробки популярних предметних наборів є скорочення необхідної кількості сканувань БД транзакцій. Алгоритм Apriori сканує базу даних кілька разів залежно від кількості елементів у предметних наборах. Є низка алгоритмів, що дають змогу зменшити необхідну кількість сканувань або кількість популярних предметних наборів, які генеруються на кожному скануванні, або обидва ці показники.

Одним із таких методів є алгоритм поділу (*Partition-based Apriori algorithm*), що потребує всього два проходи БД. Він ґрунтується на ідеї так званих локальних предметних наборів. Уся БД поділяється на N підмножин, що не перетинаються, кожна з яких досить мала, щоб поміститися в оперативній пам'яті ПК.

Під час першого сканування алгоритм зчитує кожну підмножину та виявляє предметні набори, популярні для цієї підмножини (локально-популярні предметні набори). Упродовж другого сканування алгоритм обчислює підтримку всіх локально-популярних предметних наборів по всій БД. Отже, друге сканування визначає безліч усіх потенційних асоціативних правил.

Ще одним способом підвищення ефективності методики пошуку асоціативних правил, що базується на популярних наборах, є *семплінг*. З його допомогою здійснюється відбір випадкової вибірки R із вихідної БД транзакцій, після чого починається пошук популярних наборів на цій вибірці, тобто шукається компроміс між точністю та обчислювальними витратами.

Розмір вибірки, одержаної в результаті семплінгу, повинен бути таким, щоб забезпечити прийнятні обчислювальні витрати. Очевидно, що під час цього деякі популярні набори можуть бути втрачені. Щоб звести до мінімуму втрати, використовують поріг підтримки, нижчий ніж мінімальна підтримка для пошуку частих предметних наборів, локальних на R.

7.2.5 Поставлення задачі

Реалізувати алгоритм Apriori для пошуку асоціативних правил.

Етапи виконання

1. Імпортувати дані. Індексувати їх.
2. Зафіксувати значення мінімальної підтримки й мінімальної достовірності.
3. Установити максимальне значення товарів у кошику – довжину асоціативного правила.
4. Для кожного набору, що містить від одного елемента до максимального в кошику, обчислити підтримку. Якщо вона менша за мінімальне значення, вилучити набір з таблиці.
5. Сформувати правила асоціацій.
6. Для кожного правила визначити його достовірність. Якщо вона менша за мінімальне значення, правило відкидається.
7. Відсортувати правила за зменшенням достовірності.
8. Вивести правила на екран (у файл).
9. Порівняти результати роботи власного алгоритму з результатами алгоритму *apriori*.
10. Оформити результати у вигляді звіту.

7.2.6 Приклад подання результатів

1. Імпортуємо базу даних транзакцій.
   Наприклад "`Market_Basket_Optimisation.csv`".

2. Реалізуємо алгоритм Apriori.

3. Виводимо на екран правила, відсортовані за зменшенням достовірності.

	Left hand side	Right hand side	Support	Confidence
11	pasta	shrimp	0.005067	0.322034
6	fromage blanc	honey	0.003333	0.245098
9	light cream	olive oil	0.003200	0.205128
5	fresh tuna	honey	0.004000	0.179641
7	ground beef	herb & pepper	0.016000	0.162822
10	olive oil	whole wheat pasta	0.008000	0.121704
0	brownies	cottage cheese	0.003467	0.102767
4	fresh bread	tomato juice	0.004267	0.099071
1	chicken	light cream	0.004533	0.075556
3	escalope	pasta	0.005867	0.073950

Робимо висновок про те, які товари краще розміщати разом на полицях.

7.3 Алгоритм Frequent Pattern-Growth (FPG)

Недоліком алгоритму Apriori є процес генерації кандидатів у популярні предметні набори. Наприклад, якщо база даних транзакцій містить 100 предметів, то потрібно згенерувати $2^{100} \sim 10^{30}$ кандидатів. Отже, обчислювальні й часові витрати, необхідні на їх обробку, можуть бути неприйнятними.

Крім того, алгоритм Apriori вимагає багаторазового сканування БД транзакцій, а саме: стільки разів, скільки предметів містить найдовший предметний набір. Тому було запропоновано низку алгоритмів, що дають змогу уникнути генерації кандидатів і скоротити необхідну кількість сканувань набору даних.

Однією з найефективніших процедур пошуку асоціативних правил є алгоритм, названий Frequent Pattern-Growth (алгоритм FPG), що можна перекласти як «вирощування популярних (таких, що часто зустрічаються) предметних наборів». Він дає змогу не лише уникнути витратної процедури генерації кандидатів, а й зменшити необхідну кількість проходів набору даних до двох. В основі алгоритму FPG лежить передобробка БД транзакцій, у процесі якої вона трансформується в компактну деревоподібну структуру, що називається **Frequent-Pattern Tree** – дерево популярних предметних наборів. Надалі для стислості будемо називати цю структуру FP-дерево. До основних переваг цього методу належать:

- стиснення БД транзакцій у компактну структуру, яка забезпечує найбільш ефективне та повне вилучення частих предметних наборів;
- під час побудови FP-дерева використовується технологія «розділяй та владарюй», що дає змогу декомпонувати одне складне завдання на безліч простих;
- уникнення витратної процедури генерації кандидатів, характерної для алгоритму Apriori.

Розглянемо роботу алгоритму FPG на простому прикладі. Нехай є БД транзакцій. Для цієї БД потрібно виявити всі популярні предметні набори з мінімальною підтримкою $MinSupp = 3$, використовуючи алгоритм FPG.

№	Набір предметів
1	abcde
2	abc
3	acde
4	bcde
5	bc
6	bde
7	cde

7.3.1 Побудова FP-дерева

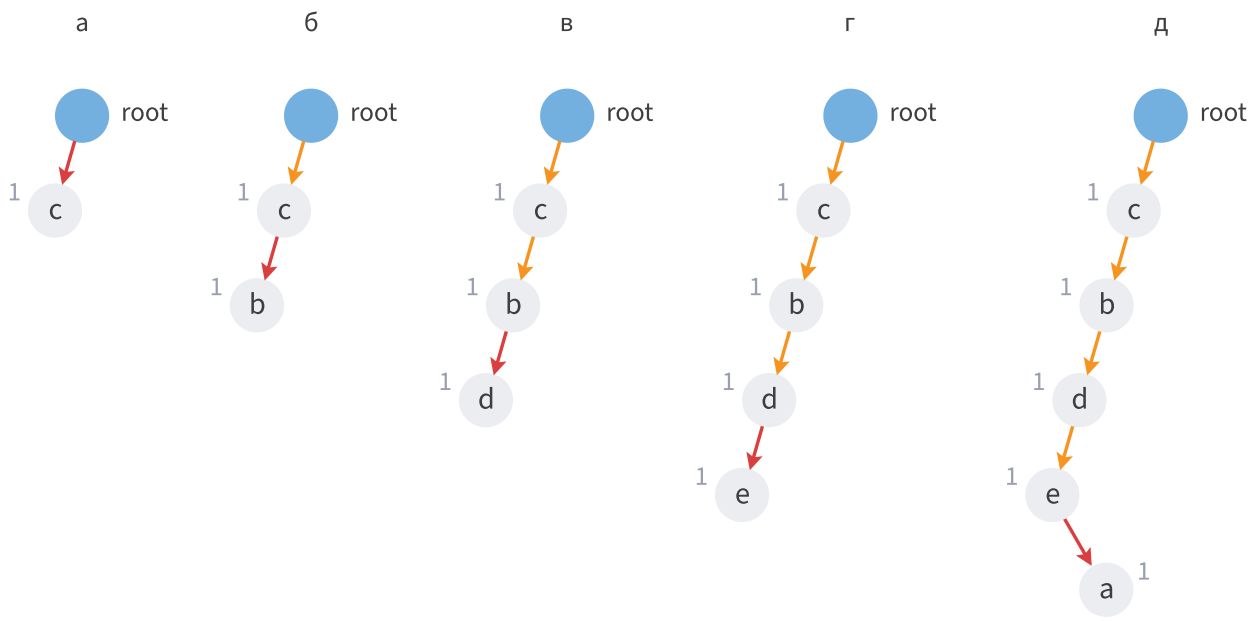
Проводиться перше сканування БД транзакцій і відбирається безліч предметів, що часто зустрічаються, тобто три чи більше разів. Упорядкуємо виявлені часті предмети в порядку зростання їх підтримки й одержимо такий набір: $(c, 6)$, $(b, 5)$, $(d, 5)$, $(e, 5)$, $(a, 3)$, де перший символ позначає предмет, а другий – кількість разів його появи. Побудуємо FP-дерево. Упорядкуємо предмети в транзакціях за спаданням значень їх підтримки.

№	Початковий набір	Впорядкований набір
1	abcde	cbdea
2	abc	cba
3	acde	cdea
4	bcde	cbde
5	bc	cb
6	bde	bde
7	cde	cde

Створимо початковий (кореневий, root) вузол FP-дерева. Почнемо побудову дерева з транзакції 1 для впорядкованих предметних наборів, тобто $(cbdea)$. Під час побудови дерева дотримуватимемося нижчезазначеного правила.

Правило.

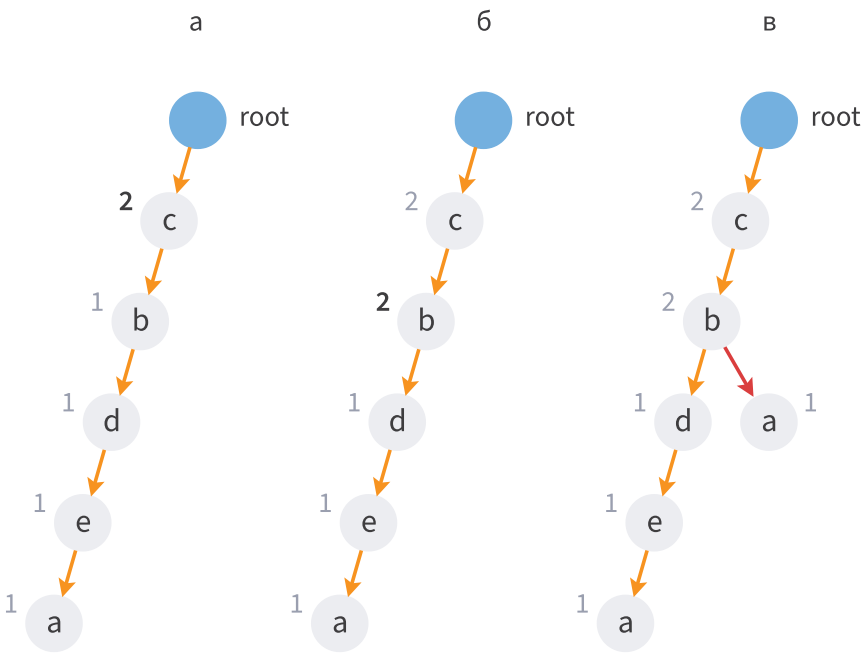
Якщо для чергового предмета у FP-дереві вже міститься вузол, то для нього не створюється новий вузол, а індекс існуючого збільшується на 1. В іншому разі для цього предмета створюється новий вузол і йому надається індекс 1.



Побудова FP-дерева для транзакції 1

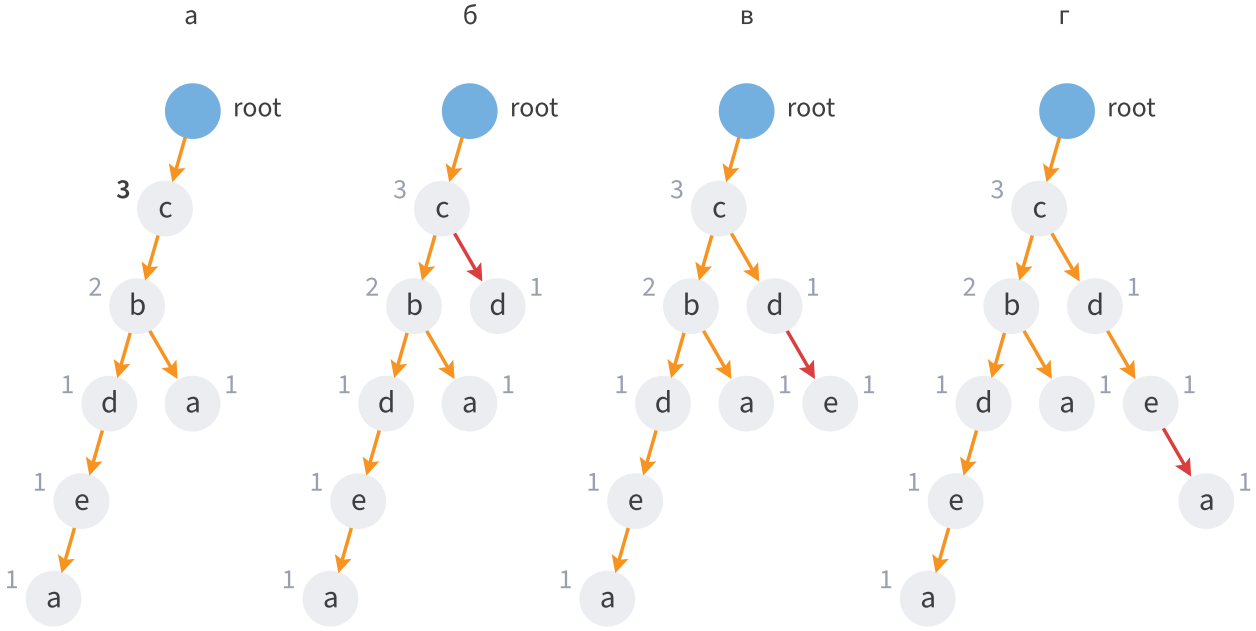
Спочатку беремо перший предмет c із транзакції 1. Оскільки він є першим, то створюємо для нього вузол та з'єднуємо з батьківським (кореневим) (рис. а). Потім беремо наступний предмет b, і оскільки інших вузлів із тим самим ім'ям FP-дерево поки не містить, додаємо його у вигляді нового вузла, нащадка вузла c (рис. б). Таким самим чином створюємо вузли для предметів d, e та a з транзакції 1 (рис. в, г, і д). На цьому використання першої транзакції для побудови FP-дерева завершено.

Для транзакції 2, що містить предмети c, b і a, вибираємо перший предмет c. Оскільки вузол із таким ім'ям вже існує, то відповідно до правила побудови FP-дерева новий вузол не створюється, а індекс існуючого збільшується на 1 (рис. а). Під час додавання наступного предмета b використовуємо те саме правило: оскільки вузол b є дочірнім щодо поточного (тобто c), ми також не створюємо новий вузол, а збільшуємо індекс для наявного (рис. б). Для наступного предмета з другої транзакції a відповідно до правила побудови FP-дерева необхідно створити новий вузол, оскільки у вузла b дочірні вузли з іменем a відсутні (рис. в).



Побудова FP-дерева для транзакцій 1 і 2

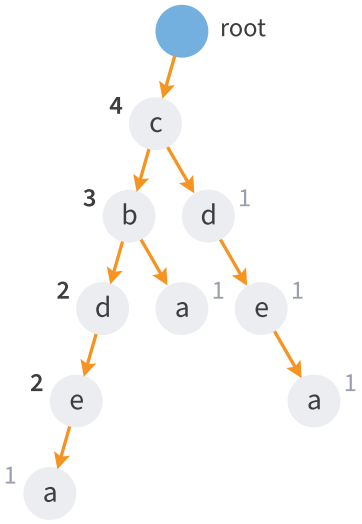
Транзакція 3 містить предмети (*cdea*). Відповідно до правила побудови FP-дерева предмет *c* не створить нового вузла, а збільшить індекс уже наявного вузла на 1 (рис. а). Наступний предмет *d* породить у FP-дереві новий вузол, дочірній до вузла *c*, оскільки той не містить нащадків із таким ім'ям (рис. б). Аналогічно предмети *e* та *a* створять нові вузли – нащадки *d* (рис. в, г).



Використання транзакції 3 для побудови FP-дерева

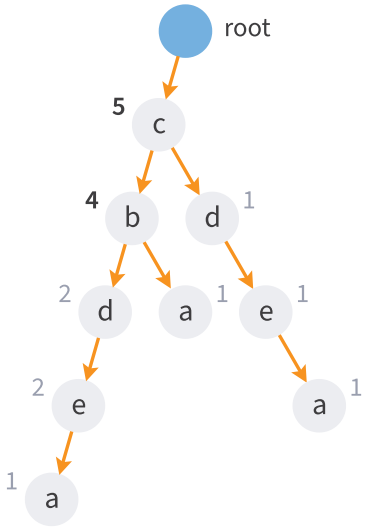
Використання транзакції 4, що містить набір предметів (*cbde*),

не створить нових вузлів, а збільшить індекси вузлів з аналогічною послідовністю імен.



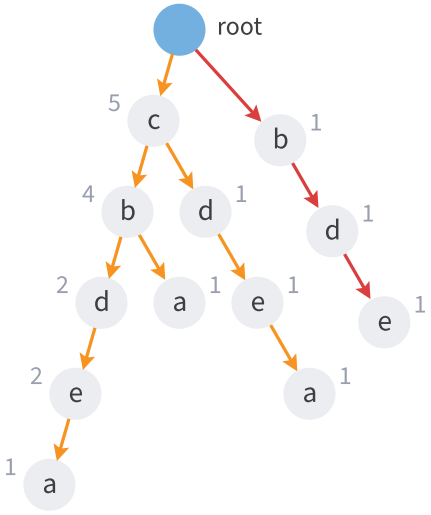
Дерево, одержане в результаті четвертої транзакції

Транзакція 5 містить набір (cb), предмети якого збільшать індекси однойменних вузлів у FP-дереві, як показано на рисунку.



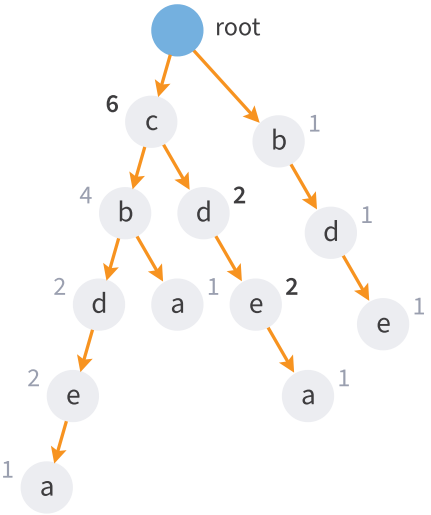
Дерево, одержане в результаті п'ятої транзакції

Транзакція 6 охоплює предмети (bde). Оскільки кореневий вузол не містить безпосереднього нащадка з ім'ям b, то відповідно до правила побудови FP-дерева для нього буде створено новий вузол, що потягне за собою два інших – d і e. Усі вузли будуть додані з індексами 1. У наслідок цього FP-дерево набирає вигляду, поданого на рисунку.



Дерево, одержане внаслідок використання 6-ї транзакції

І, нарешті, остання транзакція 7, що містить предметний набір (cde), збільшить на 1 індекс відповідних вузлів. Дерево, що вийшло, яке також є результуючим для всієї БД транзакцій, подане на рисунку.



Результуюче дерево, побудоване по всій БД транзакцій

Отже, після першого проходу БД і виконання відповідних маніпуляцій з наборами предметів ми побудували FP-дерево, яке в компактному вигляді надає інформацію про часті предметні набори і дозволяє робити їх ефективне вилучення, що відбувається під час другого сканування БД.

Подання БД транзакцій у вигляді FP-дерева очевидне. Якщо у вихідній базі даних кожен предмет повторюється багаторазово, то у FP-дереві його подають у вигляді вузла, індекс якого свідчить

про те, скільки разів цей предмет з'являється у базі даних. Іншими словами, якщо предмет у вихідній БД транзакцій з'являється 100 разів, то у FP-дереві для нього буде єдиний вузол з індексом 100.

7.3.2 Вилучення частих наборів із FP-дерева

До кожного предмета у FP-дереві, поданого своїм вузлом, можна зазначити шлях, тобто, послідовність вузлів, яку необхідно пройти від кореневого вузла до вузла, що пов'язаний із цим предметом. Якщо предмет поданий у кількох гілках FP-дерева (що найчастіше й відбувається), таких шляхів буде декілька. Наприклад, для одержаного результуючого FP-дерева для предмета a можна зазначити 3 шляхи: $\{cbde \to a, cb \to a, cde \to a\}$.

Такий набір шляхів називається умовним базисом предмета. Кожен шлях у базисі складається з двох частин: *префікса* й *суфікса. Префікс* – це власне послідовність вузлів, що утворюють шлях. *Суфікс* – це сам вузол, до якого «прокладається» шлях. Отже, в умовному базисі всі шляхи матимуть різні префікси та однаковий суфікс. Наприклад, у шляху $cbdea$ префіксом буде $cbde$, а суфіксом – a.

Процес вилучення з FP-дерева частих предметних наборів полягатиме в такому.

1. Вибираємо предмет (наприклад, a) і знаходимо в дереві всі шляхи, які ведуть до вузлів цього предмета Іншими словами, для a – це буде набір $\{cbde \to a, cb \to a, cde \to a\}$. Потім для кожного шляху підраховуємо, скільки разів цей предмет трапляється в ньому, і записуємо це як $(cbde \to a, 1)$, $(cb \to a, 1)$ і $(cde \to a, 1)$.

2. Видалимо сам предмет (суфікс набору) з шляхів, які до нього ведуть, тобто. $\{cbdea, cba, cdea\}$. Після залишаються тільки префікси: $\{cbde, cb, cde\}$.

3. Підрахуємо, скільки разів кожен предмет з'являється в префіксах шляхів, одержаних на попередньому кроці, та впорядкуємо в порядку зменшення цих значень, одержавши новий набір транзакцій.

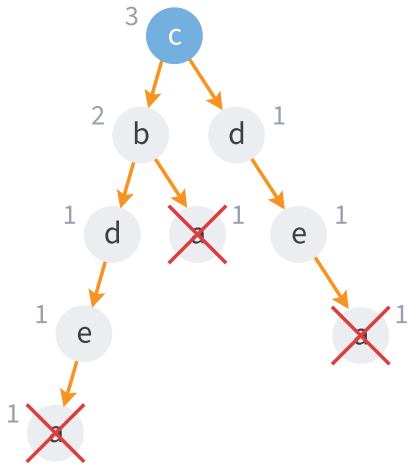
4. На його основі побудуємо нове FP-дерево, яке назвемо умовним FP-деревом, оскільки воно пов'язане лише з одним об'єктом (у нашому разі, a).

5. В умовному FP-дереві знайдемо всі предмети (вузли), для яких підтримка (кількість появи в дереві) дорівнює **3** і більше, що відповідає заданому рівню мінімальної підтримки. Якщо вузол трапляється два чи більше разів, його індекси, тобто частоти появи предмета в умовному базисі, підсумовуються.

6. Починаючи з кореня дерева, записуємо шляхи, які приводять до кожного вузла, для якого підтримка/індекс більший або дорівнює **3**, повертаємо назад предмет (суфікс), видалений на кроці 2, і підраховуємо індекс/підтримку, одержану в наслідок цього.

Для пояснення методики одержання популярного набору з FP-дерева продовжимо розгляд прикладу для БД транзакцій і побудованого для неї FP-дерева.

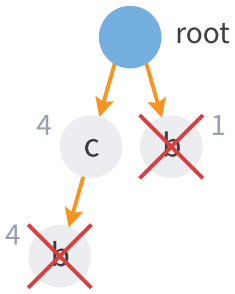
Почнемо з предмета a, який має підтримку 3 і відповідно є предметом, що часто трапляється. Префікси шляхів, що приводять до вузлів, пов'язаних з a, будуть: $(cbde \rightarrow a, 1)$, $(cb \rightarrow a, 1)$, $(cde \rightarrow a, 1)$. На основі одержаного умовного базису для суфікса a побудуємо умовне FP-дерево.



Умовне FP-дерево для предмета a

Оскільки предмети d і e трапляються двічі, їх індекси підсумовуються, й у наслідок цього одержуємо такий порядок предметів: $(c, 3)$, $(b, 2)$, $(d, 2)$, $(e, 2)$. Отже, лише вузол c задовольняє рівню мінімальної підтримки 3. Тому, для предмета a може бути згенерований лише один популярний набір $(ca, 3)$.

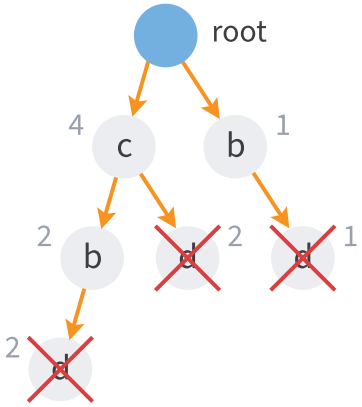
Потім переходимо до наступного предмета b з підтримкою 5. Умовне FP-дерево, побудоване для нього, буде містити лише один вузол c, оскільки в дереві є один шлях $c \to b$, а суфікс b прибираємо. Це показано на рисунку. Отже, префікси шляхів будуть $(c \to b, 4)$ і $(b, 1)$, і тому для предмета b матиме місце лише один популярний набір $(cb, 4)$.



Умовне FP-дерево для суфікса b

Для предмета c, оскільки він є безпосереднім нащадком кореневого вузла, не можна зазначити шлях (див. результуюче дерево). Отже, префікс шляхів до нього буде порожнім, із чого випливає, що і популярні набори предметів відсутні.

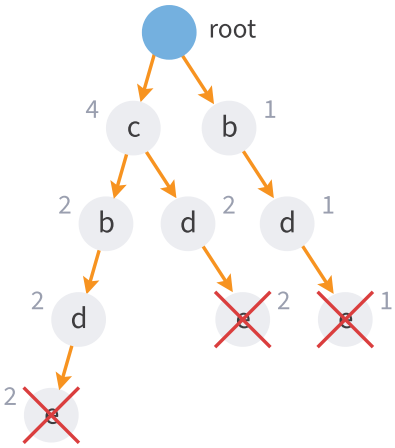
Наступний предмет, для якого ми зробимо пошук популярних предметних наборів, буде d з підтримкою 5. Умовне FP-дерево, пов'язане з предметом d подано на рисунку. Префікси шляхів для умовного дерева, пов'язаного з предметом d будуть: $(cb \to d, 2)$, $(c \to d, 2)$ та $(b \to d, 1)$. Враховуючи, що індекси для вузлів b сумуються, відповідні популярні предметні набори будуть $(cd, 4)$ і $(bd, 3)$.



Умовне FP-дерево для предмета d

І, нарешті, для останнього предмета e, що має підтримку 5, умовне FP-дерево подане на рисунку.

Префікси шляхів, що приводять в умовному дереві до вузлів, пов'язаних з предметом e, будуть: $(cbd \to e, 2)$, $(cd \to e, 2)$, $(bd \to e, 1)$. Підрахувавши сумарну підтримку кожного предмета в умовному дереві та впорядкувавши предмети з її спаданням, одержимо: $(d, 5)$, $(c, 4)$, $(b, 3)$. Отже, популярними наборами для предмета e будуть: $(de, 5)$, $(dce, 4)$, $(dbe, 3)$.

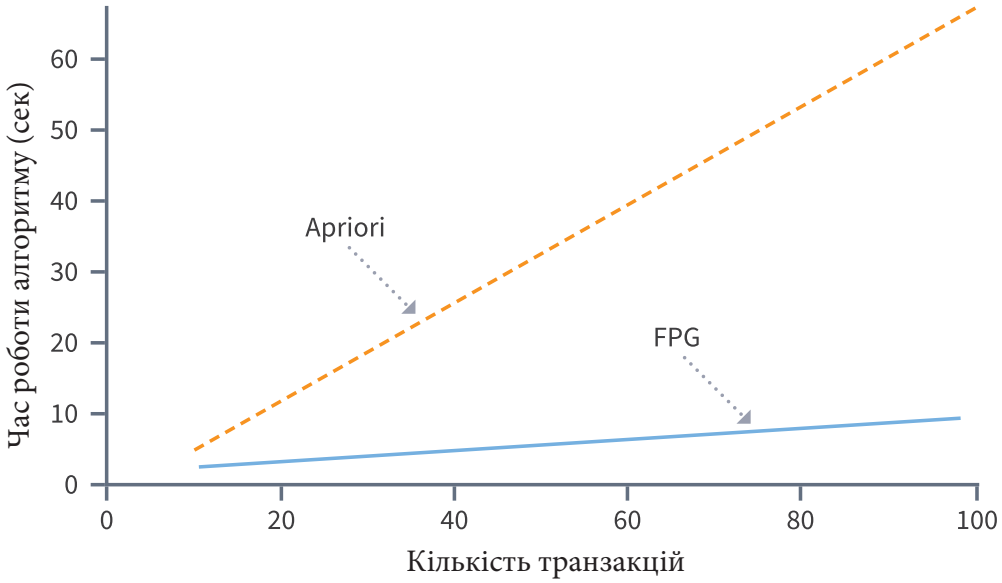


Умовне FP-дерево для предмета e

Таким чином, ми одержали такі популярні предметні набори:

$$(ca, 3),\ (cb, 4),\ (cd, 4),\ (bd, 3),\ (de, 5),\ (dce, 4),\ (dbe, 3).$$

Порівняльні дослідження класичного алгоритму Apriori та FPG показали, що зі збільшенням числа транзакцій у БД часові витрати на пошук частих предметних наборів зростають для FPG набагато повільніше, ніж для Apriori, що проілюстровано нижче.



Порівняння алгоритмів FPG та Apriori

7.3.3 Поставлення задачі

Реалізувати алгоритм FP-Growth для пошуку асоціативних правил.
Етапи виконання

1. Імпортувати дані. Індексувати їх.
2. Зафіксувати значення мінімальної підтримки та мінімальної впевненості.
3. Побудувати FP-дерево для всіх товарів в усіх транзакціях.
4. Вилучити часті набори з одержаного результуючого FP-дерева.
5. Сформувати правила асоціацій.
6. Для кожного правила визначити його достовірність. Якщо вона менша за мінімальне значення, правило відкидається.
7. Відсортувати правила за зменшенням достовірності.
8. Вивести правила на екран (у файл).
9. Порівняти результати роботи власного алгоритму з результатами алгоритму $pyfpgrowth$.
10. Оформити результати у вигляді звіту.

7.3.4 Приклад подання результатів

1. Імпортуємо базу даних транзакцій.
   Наприклад "`Market_Basket_Optimisation.csv`".

2. Реалізуємо алгоритм FP-Growth.

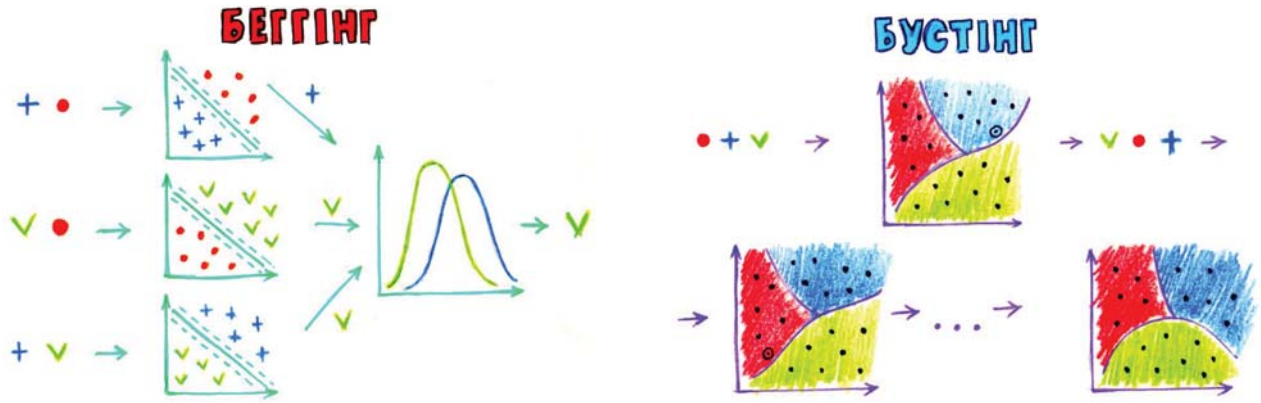
3. Виводимо на екран правила, відсортовані за зменшенням достовірності.

	Left hand side	Right hand side	Support	Confidence
11	pasta	shrimp	0.005067	0.322034
6	fromage blanc	honey	0.003333	0.245098
9	light cream	olive oil	0.003200	0.205128
5	fresh tuna	honey	0.004000	0.179641
7	ground beef	herb & pepper	0.016000	0.162822
10	olive oil	whole wheat pasta	0.008000	0.121704
0	brownies	cottage cheese	0.003467	0.102767
4	fresh bread	tomato juice	0.004267	0.099071
1	chicken	light cream	0.004533	0.075556
3	escalope	pasta	0.005867	0.073950

Робимо висновок про те, які товари краще розміщувати разом на полицях.

Розділ 8

Ансамблеві методи навчання



У цьому розділі буде надано основний теоретичний матеріал щодо використання ансамблевих методів у машинному навчанні. Буде розглянуто два основні підходи до створення ансамблів:

- беггінг (Begging);
- бустінг (Boosting).

Для кожного методу буде наведений приклад його використання, поставлені завдання для практичної роботи та наведений приклад подання результатів.

8.1 Принципи побудови ансамблів

Ансамблі (комбінації моделей) є лідерами на сьогодні в класичному машинному навчанні. Вони надають найточніші результати та використовуються всіма великими компаніями. Ансамблеві методи – це парадигма машинного навчання, де кілька моделей (часто їх називають «слабкими учнями») навчаються для вирішення однієї й тієї самої проблеми і об'єднуються для одержання кращих результатів. Основна гіпотеза полягає в тому, що під час правильного поєднання слабких моделей можна одержати більш точну та/або надійну модель.

8.1.1 Один слабкий учень

У машинному навчанні, незалежно від того, чи це є класифікація або регресія, вибір моделі надзвичайно важливий, щоб мати шанси одержати хороші результати. Цей вибір може залежати від багатьох змінних задачі: кількості даних, розмірності простору, гіпотези розподілу. В ансамблевій теорії навчання вводиться поняття слабких учнів (або базових алгоритмів), яких можна використовувати як «будівельні блоки» для проєктування більш складних моделей шляхом об'єднання кількох із них. Здебільшого ці базові алгоритми працюють не дуже добре. Ідея ансамблевих методів полягає в тому, щоб спробувати зменшити похибки таких слабких учнів, об'єднуючи кілька з них разом, щоб створити сильного учня (або модель ансамблю), який досягає кращих результатів.

8.1.2 Об'єднання слабких учнів

Щоб реалізувати ансамблевий метод, спочатку потрібно відібрати слабких учнів для агрегування. В основному використовується єдиний базовий алгоритм навчання, але моделі навчаються на різних даних. Така модель ансамблю називається «однорідною». Існують також методи, які використовують різні типи базових алгоритмів навчання: деякі різнорідні слабкі учні об'єднуються в «різнорідну ансамблеву модель». Одним із важливих моментів є те,

що вибір слабких учнів повинен бути узгоджений із тим, як комбінуються ці моделі.

Під час формування ансамблю моделей необхідно вирішити три завдання:

- вибрати базові моделі;
- визначити підхід до використання навчальної множини;
- вибрати метод комбінування результатів.

Оскільки ансамбль – це складна модель, що складається з окремих базових моделей, то під час його формування можливі два випадки:

- ансамбль складається з базових моделей одного типу, наприклад, лише з дерев рішень, лише з нейронних мереж тощо;
- ансамбль складається з моделей різного типу – дерев рішень, нейронних мереж, регресійних моделей тощо.

Під час побудови ансамблю використовується навчальна вибірка, для якої існують два підходи:

- перевибірка, тобто з існуючої навчальної вибірки витягується кілька підмножин, кожна з яких використовується для навчання однієї з моделей ансамблю;
- використання однієї навчальної вибірки для навчання всіх моделей ансамблю.

Для комбінування результатів, виданих окремими моделями, використовують три способи:

Просте голосування – вибирається той результат, який був виданий простою більшістю моделей ансамблю.

Зважене голосування – для моделей ансамблю встановлюються значення ваг, з врахуванням яких виноситься результат.

Усереднення (зважене або незважене) – результат роботи ансамблю визначається як просте середнє значення результатів усіх моделей, під час зваженого усереднення результати всіх моделей множать на відповідні ваги.

На сьогодні розроблено багато різних методів і алгоритмів формування ансамблів. Основні типи ансамблевих алгоритмів, які спрямовані на об'єднання слабких учнів:

Беггінг. Однорідні слабкі учні навчаються паралельно й незалежно один від одного, далі вони об'єднуються, використовуючи певний детермінований процес усереднення.

Бустінг. Однорідні слабкі учні навчаються послідовно адаптивним способом (кожен наступний слабкий учень залежить від попередніх), вони об'єднуються, використовуючи детерміновану стратегію.

Стекінг. Різнорідні слабкі учні навчаються паралельно й об'єднуються, навчаючи метамодель для виведення результату, заснованого на результатах різних слабких моделей.

8.2 Визначення ансамблю

Вихідні дані:

- $X^\ell = (x_i, y_i)_{i=1}^\ell \subset X \times Y$ – навчальна вибірка;
- $b_t : X \to Y$, $t = 1, \ldots, T$ – базові алгоритми, що навчаються.

Ідея ансамблю: побудувати з множини «слабких» алгоритмів b_t один «сильний».
Декомпозиція базових алгоритмів: $a_t(x) = C(b_t(x))$:

$$a_t : X \xrightarrow{b_t} R \xrightarrow{C} Y :$$

- R – найбільш зручний простір оцінок;
- C – вирішальне правило, як правило досить простого вигляду.

Ансамбль базових алгоритмів $b_1, \ldots, b_t$:
$F : R^T \times X \to R$ – агрегуюча функція або мета-алгоритм.

8.2.1 Агрегуючі функції

Загальні вимоги до агрегуючих функцій:

- $F(b_1, \ldots, b_T, x) \in [\min_t b_t, \max_t b_t]$ середнє для всіх x;
- $F(b_1, \ldots, b_T, x)$ – монотонно не спадає за всіма b_t.

Приклади агрегуючих функцій

- просте голосування (simple voting)

$$F(b_1, \ldots, b_T) = \frac{1}{T} \sum_{t=1}^{T} b_t;$$

- зважене голосування (weighted voting)

$$F(b_1, \ldots, b_T) = \sum_{t=1}^{T} a_t b_t, \quad \sum_{t=1}^{T} a_t = 1;$$

- суміш алгоритмів (mixture of experts) із функціями компетентності

$$F(b_1, \ldots, b_T, x) = \sum_{t=1}^{T} g_t(x) b_t.$$

Чудовим прикладом ансамблів вважається теорема Кондорсе «про журі присяжних» (1784). Якщо кожен член журі присяжних має незалежну думку, і якщо ймовірність правильного рішення члена журі більше 0,5, тоді ймовірність правильного рішення присяжних загалом зростає зі збільшенням кількості членів журі та прямує до одиниці. Якщо ж можливість бути правим у кожного з членів журі менше ніж 0,5, то можливість ухвалення правильного рішення присяжними в цілому монотонно зменшується і прямує до нуля зі збільшенням кількості присяжних.

Математично ймовірність правильного рішення всього журі можна подати за допомогою формули

$$\mathcal{P} = \sum_{i=m}^{N} C_N^i p^i (1-p)^{N-i},$$

де

- $\mathcal{P}$ – імовірність правильного вирішення всього журі;
- N – кількість присяжних;
- p – імовірність правильного рішення присяжного;
- m – мінімальна більшість членів журі;
- C_N^i – число поєднань з N по i.

Властивості:

- якщо $p > 1/2$, то $\mathcal{P} > p$: за $N \to \infty \Rightarrow \mathcal{P} \to 1$;
- якщо $p < 1/2$, то $\mathcal{P} < p$: за $N \to \infty \Rightarrow \mathcal{P} \to 0$.

Інший приклад ансамблів заснований на принципі «Мудрість натовпу». Френсіс Гальтон у 1906 році відвідав ринок, де проводилася лотерея для селян. Їх зібралося близько 800 людей і вони намагалися вгадати вагу бика, який стояв перед ними. Його вага становила 1198 фунтів. Жоден селянин не вгадав точну вагу бика, але коли порахували середнє від їхніх передбачень, то отримали 1197 фунтів. Цю ідею зменшення помилки застосували й у машинному навчанні.

8.2.2 Проблема різновиду базових алгоритмів

Якщо розглядати значення базових алгоритмів b_t на об'єкті як незалежні випадкові величини ξ_t з однаковим математичним сподіванням E та однаковою дисперсією D, то випадкова величина $\xi = (1/T)(\xi_1 + \cdots + \xi_T)$ має таке саме математичне сподівання, але меншу дисперсію:

- $E\xi = \frac{1}{T}\left(E\xi_1 + \cdots + E\xi_T\right) = E\xi_t$;
- $D\xi = \frac{1}{T^2}\left(D\xi_1 + \cdots + D\xi_T\right) = \frac{1}{T}D\xi_t$.

Отже, дисперсія суми випадкових величин прямує до нуля $D \to 0$ під час збільшення кількості випадкових величин $T \to \infty$.

Базові алгоритми не є незалежними випадковими величинами:

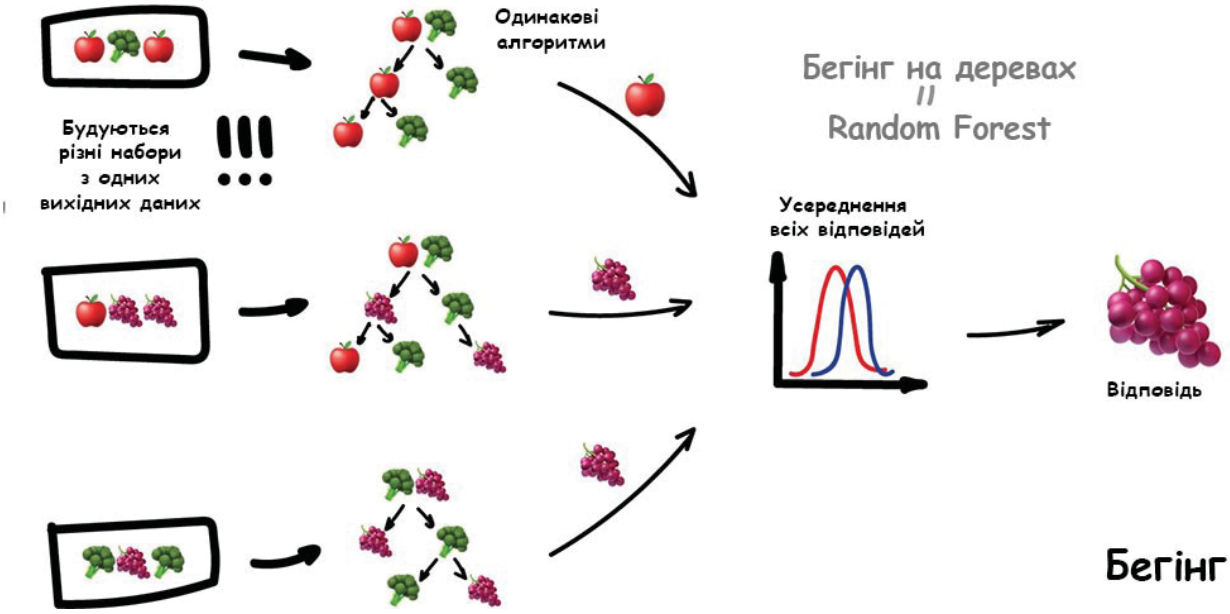
- вирішують одне й те саме завдання;
- налаштовуються на один цільовий вектор;
- зазвичай обираються з однієї й тієї самої моделі.

Способи підвищення різновиду базових алгоритмів:

- навчання на різних (випадкових) підвибірках;
- навчання на різних (випадкових) наборах ознак;
- навчання з різних параметричних моделей;
- навчання з використанням рандомізації;
- навчання на зашумлених даних (інколи).

8.3 Бегінг (Bootstrap AGGregatING)

Бегінг – технологія навчання, де елементарні моделі (однакові) навчаються й працюють паралельно (незалежно один від одного) на кількох різних вибірках однакового розміру, одержаних шляхом випадкового відбору прикладів із вихідного набору даних. Ідея полягає в тому, що моделі не виправляють помилки одна одної, а компенсують їх під час голосування.



Алгоритм бегінгу має такі кроки.

Спочатку формується кілька вибірок шляхом випадкового відбору з початкової множини даних. Дані в випадкових вибірках можуть повторюватися. На них навчається один і той самий алгоритм кілька разів, а наприкінці відповідь обчислюється простим голосуванням або усереднюється. Точність результату, одержаного за способом бегінгу, виявляється значно вище, ніж точність окремої моделі, що навчається на повному наборі навчальних даних. Різниця між бегінгом та ідеальною процедурою навчання моделей на незалежних вибірках полягає в способі формування навчальних множин. Замість одержання незалежних множин із предметної області бегінг просто комбінує перевибірку з існуючої множини даних. Такі підмножини відрізняються між собою, але не є незалежними, оскільки засновані на одній навчальній множині.

8.3.1 Метод бутстрепу (bootstrap)

Статистичний метод бутстрепу дозволяє оцінювати багато різних статистичних характеристик складних моделей. Він полягає в такому.

Нехай є вибірка X розміру ℓ. Поступово візьмемо з вибірки N об'єктів із поверненням. Це означає, що ми N раз вибиратимемо довільний об'єкт вибірки (вважаємо, що кожен об'єкт обирається з однаковою ймовірністю $\frac{1}{\ell}$), причому щоразу ми вибираємо зі всіх вихідних об'єктів: обраний на якомусь кроці об'єкт повертається назад у вибірку, і наступний вибір знову робиться рівноймовірно з тієї загальної вибірки. Зазначимо, що через повернення серед таких об'єктів виявляться повтори.

Імовірність P_- того, що об'єкт **НЕ** попаде до підвибірки (тобто його не взяли ℓ разів): визначається за допомогою формули $P_- = (1 - 1/\ell)^\ell$. За $\ell \to \infty$ одержуємо одну з визначних границь $1/e$. Тоді ймовірність попадання конкретного об'єкта до підвибірки $P_+ \simeq 1 - 1/e \simeq 63$ %. Отже, приблизно 37 % об'єктів залишаються поза вибіркою і не використовуються під час організації k-ї вибірки.

8.3.2 Методи стохастичного ансамблювання

Способи підвищення різновиду з використанням рандомізації:

- Bagging (bootstrap aggregating) – використання підвибірок навчальної вибірки з «поверненням», в кожну вибірку потрапляє $(1 - 1/e) \simeq 63\%$ об'єктів;
- Pasting – випадкові підвибірки для навчання;
- Random subspaces – випадкові підмножини ознак;
- Random patches – випадкові підмножини об'єктів та ознак;
- Cross-validated committees – вся вибірка розбивається на k блоків і проводиться навчання k разів без одного блоку.

У загальному вигляді задачу навчання кожного алгоритму на своїй підвибірці можна сформулювати так: $\mu : \ (G, U) \to b_t$ – метод

навчання на підвибірці $U \subseteq X^\ell$, який використовує лише ознаки з множини $G \subseteq F^n\{f_1, \ldots, f_n\}$ Отже, формується k вибірок за своїми випадковими об'єктами та ознаками.

Приклад на псевдокоді:

Вхід: навчальна вибірка X^ℓ, параметри T базових алгоритмів:

ℓ' – розмір кожної навчальної підвибірки;

n' – розмірність підпросторів ознак;

ε_1 – поріг якості базових алгоритмів на навчанні;

ε_2 – поріг якості базових алгоритмів на тестуванні;

Вихід: базові алгоритми b_t, $t = 1, \ldots, T$.

Для всіх $t = 1, \ldots, T$:

- визначаємо випадкову підвибірку U_t розміром ℓ' із X^ℓ;
- визначаємо випадкову множину ознак G_t розмірності n' із F^n;
- застосовуємо базовий алгоритм $b_t = \mu(G_t, U_t)$;
- якщо $Q(G_t, U_t) > \varepsilon_1$, то b_t не входиться до ансамблю;
- якщо $Q(G_t, X^\ell / U_t) > \epsilon_2$, то b_t не входиться до ансамблю.

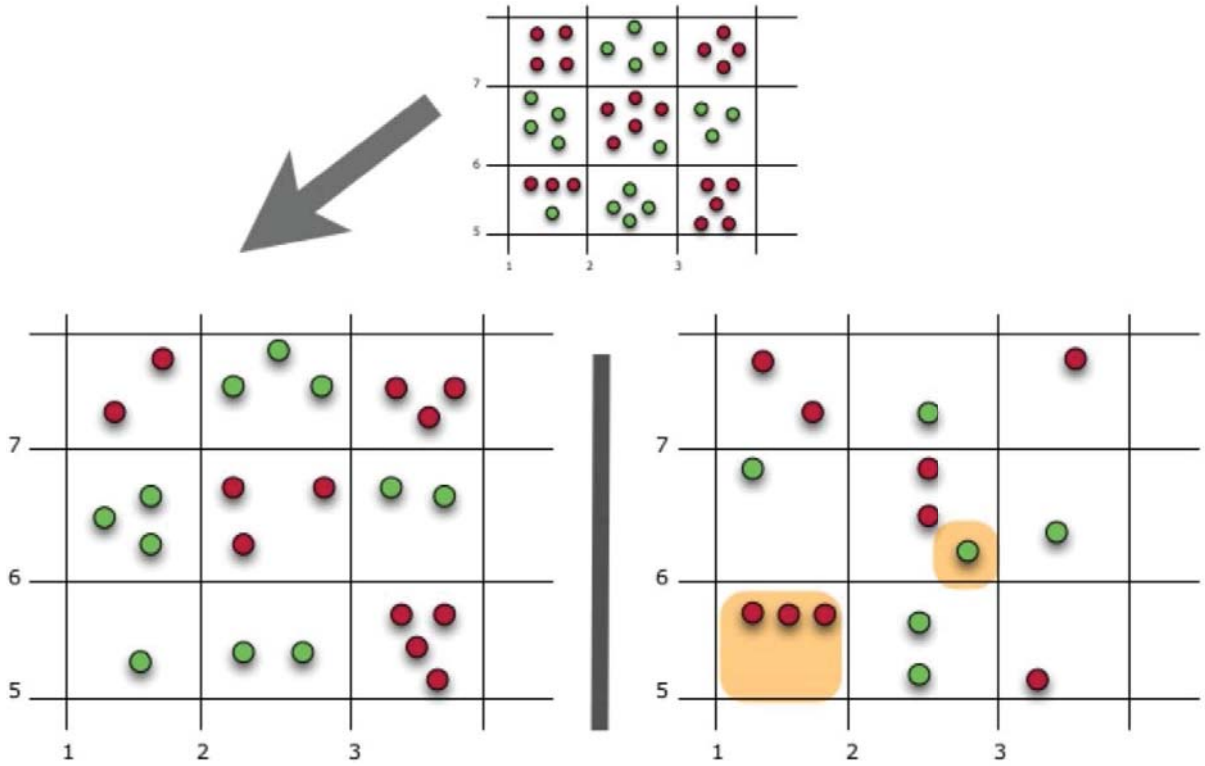
Результуючий ансамбль – просте голосування $b(x) = \frac{1}{T}\sum_{t=1}^{T} b_t(x)$

8.3.3 Незміщене оцінювання помилок (Out-of-bag)

Приклад застосування bootstrap.

На рисунку наведено варіант формування навчальної (зліва) та тестової (справа) вибірки із загальної вибірки. Застосування певного класифікатора показало, що він помилився на чотирьох об'єктах, які не використовувалися для навчання. Отже, якість роботи класифікатора становить: $\frac{11}{15} \times 100\ \% \simeq 73.3\ \%$. Тому, кожен базовий алгоритм навчається на $\sim 63\ \%$ всіх об'єктах і на інших $\sim 37\ \%$ його можна відразу перевіряти.

Out-of-Bag (OOB)-оцінка – це усереднене оцінювання базових алгоритмів на тих ~ 37 % даних, на яких алгоритми не навчались:



- незміщене оцінювання ансамблю на об'єкті

$$OOB(x_i) = \frac{1}{T_i} \sum_{t \in T_i} b_t(x_i), \quad T_i = \{t : x_i \notin U_t\};$$

- незміщене оцінювання помилки ансамблю на навчальній вибірці

$$OOB(X^\ell) = \sum_{i=1}^{\ell} \mathcal{L}(OOB(x_i), y_i),$$

де $\mathcal{L}$ – значення функції втрат на об'єкті x_i;

- оцінювання важливості (importance) ознак f_j, $j = 1, \ldots, n$:

$$I_j = \frac{OOB^j(X^\ell) - OOB(X^\ell)}{OOB(X^\ell)} \cdot 100\ \%.$$

Під час обчислення $b_t(x_i)$ для OOB^j значення ознаки f_j випадковим чином перемішуються на всіх об'єктах $x_i \notin U_t$.

8.3.4 Переваги та обмеження бегінгу

Переваги:

- метод-обгортка (envelope) над базовим методом навчання;
- підходить для класифікації, регресії й інших завдань;
- проста реалізація;
- просте розпаралелювання;
- можливість отримання незміщених оцінок OOB;
- можливість оцінювання важливості ознак.

Обмеження:

- потребує дуже багато базових алгоритмів;
- складно агрегувати стійкі базові методи навчання.

Один із найкращих та універсальних методів – **випадковий ліс** – бегінг побудований на деревах ухвалення рішень.

8.3.5 Випадковий ліс (Random forest)

Розглянемо приклад побудови begging-ансамблю на прикладі випадкового лісу, коли як базові алгоритми будуть дерева ухвалення рішень. Припустимо, у нас є дані що мільйона музичних кліпів на ютубі. Щодо кожного є 100 критеріїв, наприклад:

- чи триває кліп довше, ніж три хвилини;
- цей трек у жанрі «хіп-хоп» чи ні;
- чи випустив кліп популярний «лейбл»;
- чи записано кліп у дворі на мобільний телефон;
- і т.д.

Також у нас є дані про те, чи набрав кліп більше ніж мільйон переглядів. Ми хочемо навчитися передбачати цей критерій – назвемо його популярністю. Тобто ми хочемо одержати якийсь алгоритм, якому на вхід подаєш 100 критеріїв кліпу у форматі так/ні, а на виході одержати відповідь: «Цьому кліпу судилося стати популярним».

Кліп	Критерій 1	Критерій 2	Критерій 3	...	Критерій 100	10^6 переглядів
1	Так	Ні	Ні	...	Ні	Так
2	Так	Так	Так	...	Так	Так
3	Ні	Ні	Так	...	Так	Ні
4	Ні	Так	Так	...	Ні	Ні
5	Ні	Так	Ні	...	Ні	Ні
6	Так	Так	Так	...	Так	Так
7	Ні	Ні	Так	...	Ні	Так
...	...	...	...	...	...	...
10^6	Так	Так	Ні	...	Ні	Так

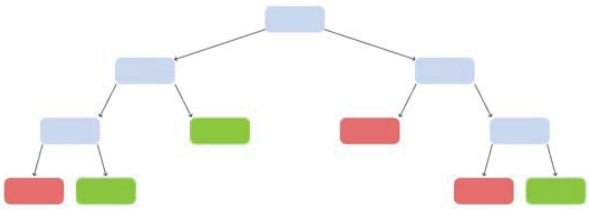
Візьмемо випадкову вибірку з наших вихідних даних. Не мільйон кліпів, а 10 000.

Кліп	Критерій 1	Критерій 2	Критерій 3	...	Критерій 100	10^6 переглядів
1	Так	Ні	Ні	...	Ні	Так
2	Так	Так	Так	...	Так	Так
3	Ні	Ні	Так	...	Так	Ні
4	Ні	Так	Так	...	Ні	Ні
5	Ні	Так	Ні	...	Ні	Ні
6	Так	Так	Так	...	Так	Так
7	Ні	Ні	Так	...	Ні	Так
...	...	...	...	...	...	...
10^6	Так	Так	Ні	...	Ні	Так

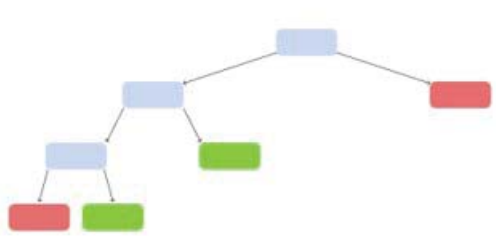
До них – випадковий набір критеріїв, не всі 100, а 5.

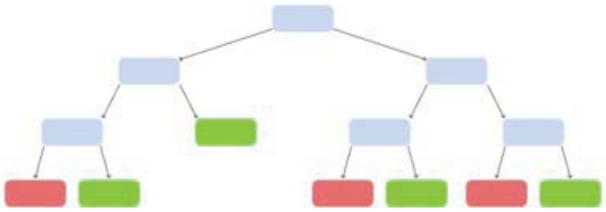
Кліп	Критерій 1	Критерій 2	Критерій 3	…	Критерій 100	10^6 переглядів
1	Так	Ні	Ні	…	Ні	Так
2	Так	Так	Так	…	Так	Так
3	Ні	Ні	Так	…	Так	Ні
4	Ні	Так	Так	…	Ні	Ні
5	Ні	Так	Ні	…	Ні	Ні
6	Так	Так	Так	…	Так	Так
7	Ні	Ні	Так	…	Ні	Так
…	…	…	…	…	…	…
10^6	Так	Так	Ні	…	Ні	Так

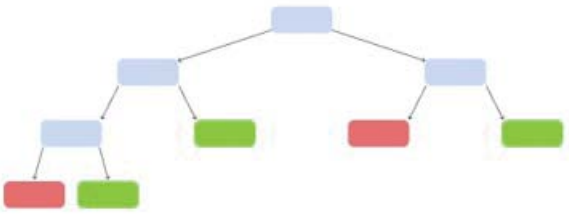
Будуємо дерево



Так будуємо ще декілька дерев, кожне – на своєму наборі даних та своєму наборі критеріїв.

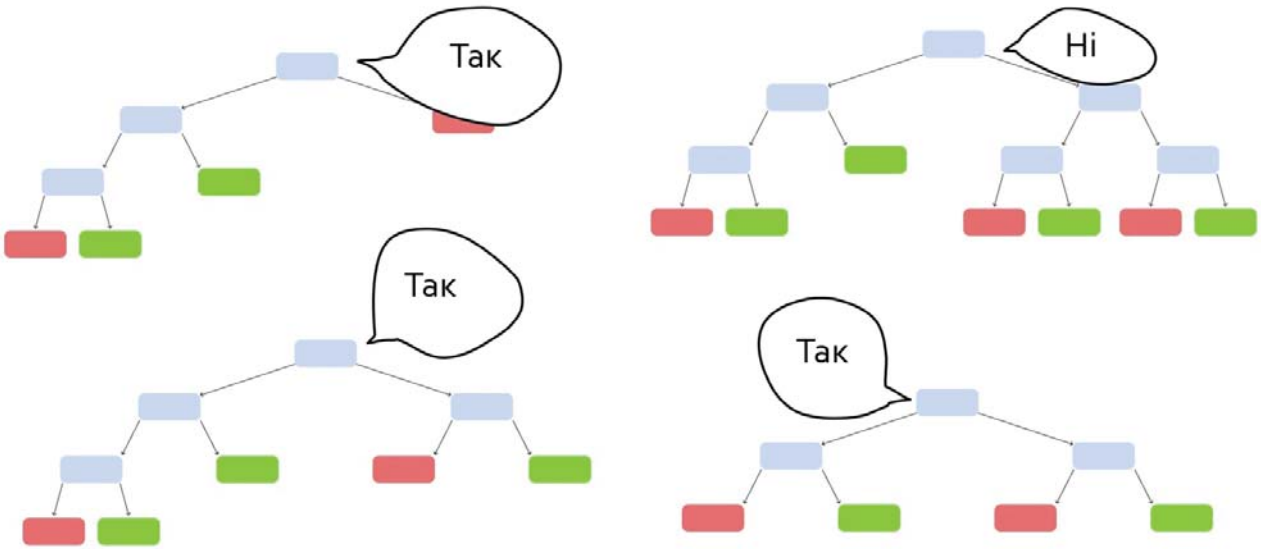






У нас з'явився випадковий ліс: випадковий, бо ми щоразу брали випадковий набір даних і критеріїв; ліс, бо багато дерев. Тепер запустимо кліп, якого не було в навчальній вибірці. Кожне

дерево видасть свій вердикт, чи стане він популярним: так чи ні. Простим голосуванням обираємо варіант, який одержить найбільше голосів.



8.3.6 Поставлення задачі

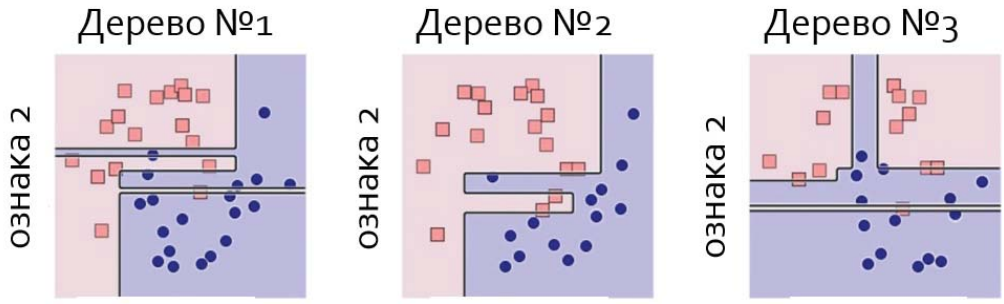
Провести класифікацію даних (або регресійний аналіз) із використанням **begging**-ансамблю.
Етапи розв'язання

1. Імпортувати вибірку для проведення навчання.
2. Розділити всю вибірку на навчальну та тестову.
3. Обрати базові алгоритми.
4. Побудувати алгоритм навчання на навчальній вибірці.
5. За можливості реалізувати процедуру паралельного навчання.
6. Подати графічно результат класифікації (регресії) для навчальної вибірки.
7. Побудувати залежність точності алгоритму від кількості базових алгоритмів.
8. Перевірити точність роботи алгоритму на тестовій вибірці.
9. Подати графічно результат класифікації для тестової вибірки.
10. Порівняти власні результати з результатами роботи вбудованих алгоритмів з sklearn.
11. Оформити результати у вигляді звіту.

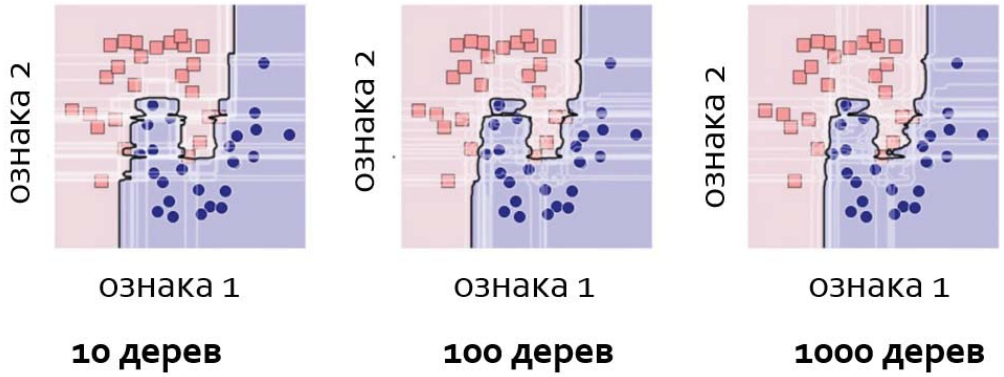
8.3.7 Приклад подання результатів

Класифікація даних за допомогою лісу дерев

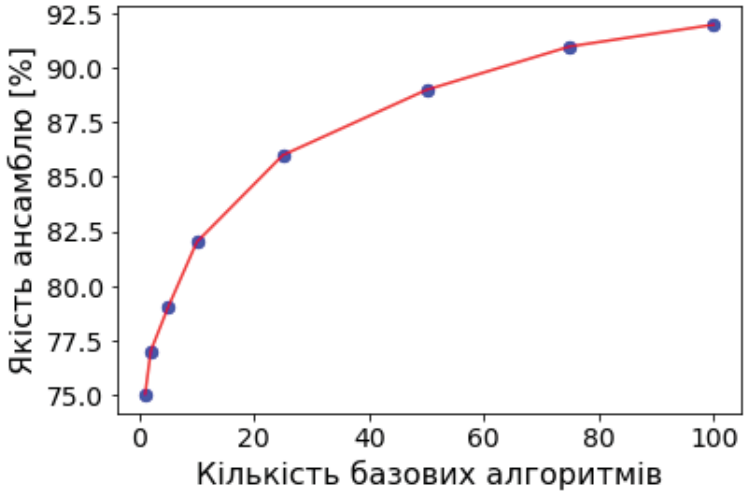
Результат класифікації трьох окремих дерев



Результат класифікації лісу з різною кількістю дерев

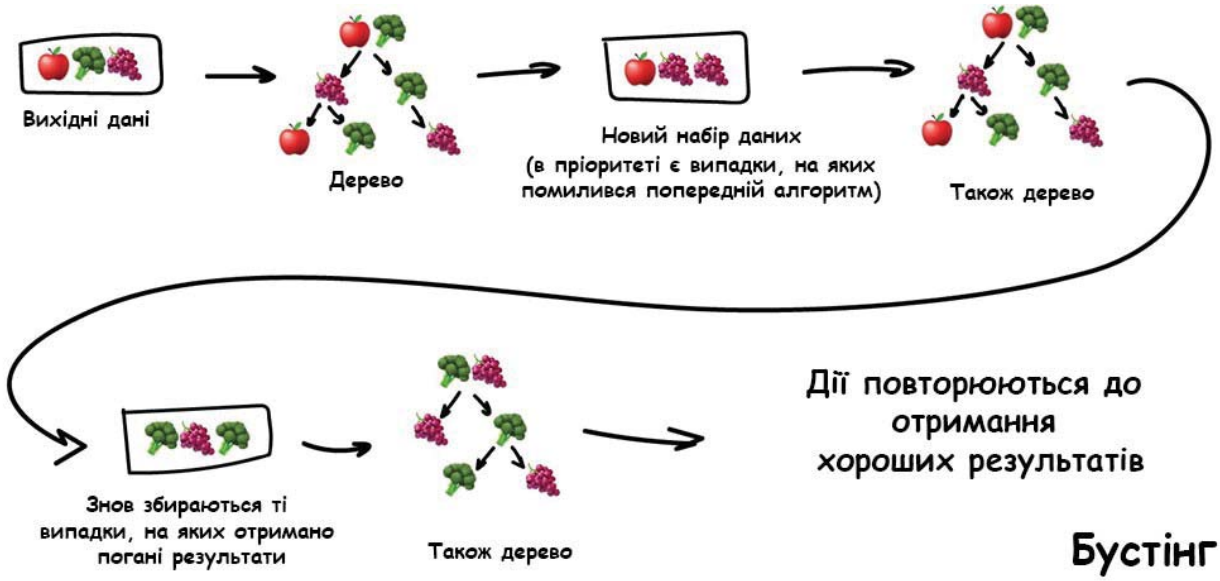


Залежність якості ансамблю від кількості базових алгоритмів



8.4 Бустінг (Boosting)

Бустінг – це процедура послідовної побудови композиції алгоритмів машинного навчання, коли кожен наступний алгоритм прагне компенсувати недоліки композиції всіх попередніх алгоритмів. Алгоритми виконуються послідовно, кожен наступний приділяє особливу увагу тим випадкам, на яких помилився попередній.



У машинному навчанні є поняття сильної й слабкої моделей. Сильною моделлю називається та модель, яка допускає мінімальну кількість помилок класифікації. Слабка модель, навпаки, допускає багато помилок – тобто не є точною (або втрачає в надійності). **Бустінгом** називається метод, що спрямований на перетворення слабких моделей у сильні шляхом побудови ансамблю моделей.

Порівняно з бегінгом бустінг є більш складною процедурою, але в багатьох випадках працює ефективніше.

- Бустінг починає створення ансамблю на основі єдиної вихідної множини, але на відміну від бегінгу, кожна нова модель будується на основі результатів попередньої, тобто моделі будуються послідовно.

- Бустінг створює нові моделі так, щоб вони доповнювали раніше побудовані, виконували ту роботу, яку інші моделі зробити не змогли на попередніх кроках.

- І нарешті, остання відмінність бустінгу від бегінгу полягає в тому, що всі побудовані моделі в залежності від їх точності мають різні вагові коефіцієнти.

Бустінг належить до ітераційних алгоритмів. Він вчиться розпізнавати приклади на межах класів. Кожному запису даних на кожній ітерації алгоритму надається значення вагового коефіцієнта. Перший класифікатор навчається на всіх прикладах із однаковими вагами. На кожній наступній ітерації ваги розставляються відповідно до класифікованих прикладів, тобто ваги правильно класифікованих прикладів зменшуються, а неправильно класифікованих – збільшуються. Отже, пріоритетними для наступного класифікатора стануть неправильно розпізнані приклади, навчаючись на яких новий класифікатор буде виправляти помилки класифікатора минулої ітерації.

Значним плюсом є висока точність результатів, але мінусом є не паралельний процес, хоча цей алгоритм працює швидше, ніж нейромережі.

8.4.1 Бустінг для задачі класифікації з 2 класами

Поставлення задачі:

- навчальна вибірка розміру ℓ: X^ℓ;
- відповіді $Y = \{\pm 1\}$;
- базові алгоритми $b_t : X \to \{-1, 0, 1\}$. $b_t(x) = 0$ – відмова (краще промовчати ніж збрехати);
- вирішальне правило $C(b) = sign(b)$;
- зважене голосування

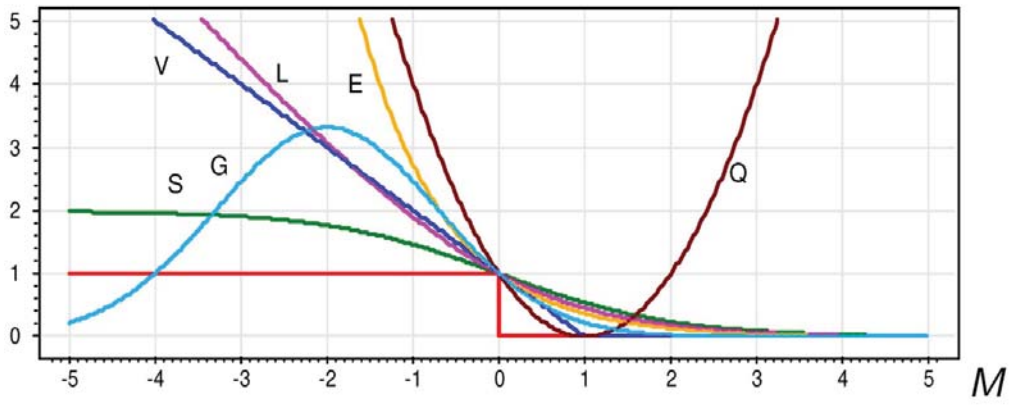
$$a(x) = sign\left(\sum_{t=1}^{T} \alpha_t b_t(x)\right), \quad x \in X^\ell;$$

- функціонал якості композиції – кількість помилок на X^ℓ

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

8.4.2 Гладкі апроксимації порогової функції втрат

Типи порогових функцій втрат від відступу M для різних варіантів бустінгу:



- $E(M) = e^{-M}$ – експоненційна (AdaBoost);
- $E(M) = \log_2(1 + e^{-M})$ – логарифмічна (LogitBoost);
- $(1 - M)^2$ – квадратична (GentleBoost);
- $exp(-cM(M + s))$ – Гаусова (BrownBoost);
- $2(1 + e^M)^{-1}$ – сигмоїдна;
- $(1 - M)_+$ – кусково-лінійна (SVM).

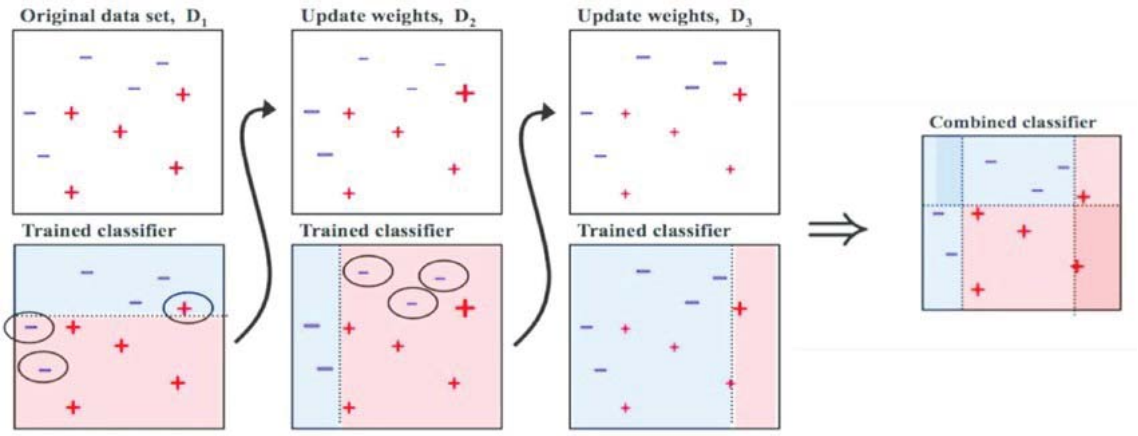
8.4.3 Адаптивний бустінг (AdaBoost)

Адаптивний бустінг, відомий як AdaBoost, – це алгоритм бустінгу, який мінімізує помилки попередніх моделей, концентруючи увагу на недонавченості алгоритму. Тобто за кожного нового пророкування алгоритм працюватиме над складними, неочевидними для передбачення підвиборками.

Побудова класифікатора AdaBoost.

Як перший базовий алгоритм навчаємо вирішальне дерево, щоб робити передбачення на навчальній вибірці. Тепер, за методом адаптивного бустінгу, збільшуємо значення вагових коефіцієнтів для помилково класифікованих елементів. Другий алгоритм на-

вчається вже з урахуванням оновлених коефіцієнтів, і так ця процедура повторюється щоразу. Після передбачень кожної моделі збільшуємо значення вагових коефіцієнтів для помилково класифікованих об'єктів, щоб наступна модель краще спрацювала на них.



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

Кліп	10^6 переглядів	Відповідь
1	Так	Так
2	Так	Так
3	Ні	Ні
4	Ні	Ні
5	Ні	Так
6	Так	Так
7	Так	Ні
...	...	...
10^6	Так	Так

Звернути увагу

Побудуємо схожий ліс, але набір даних буде невипадковим. Перше дерево ми побудуємо так само, як і раніше, на випадкових

даних і випадкових критеріях. А потім проженемо через це дерево контрольну вибірку: інші кліпи, за якими у нас є всі дані, але які не беруть участі в навчанні. Тепер робимо таке дерево. Звернімо увагу на місця, де перше дерево помилилося. Приділимо цим помилкам більшу вагу під час підбору даних і критеріїв для навчання. Завдання – зробити дерево, яке виправить помилки попереднього. Але друге дерево наробить своїх помилок. Робимо третє, яке їх виправить. Потім четверте. Потім п'яте. Робимо такі дерева, поки не досягнемо бажаної точності або поки точність не почне падати через перенавчання. Виходить, що у нас багато дерев, кожне з яких не дуже сильне, але разом вони складаються в ліс, який дає хорошу точність.

Експоненційна функція втрат

Оцінка функціоналу якості Q_T зверху:

$$Q_T \leq \tilde{Q}_T = \sum_{i=1}^{\ell} \exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t b_t(x_i)\right) \exp(-y_i \alpha_T b_T(x_i)).$$

Уведемо вагові коефіцієнти:

$$w_i = \exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t b_t(x_i)\right).$$

Нормовані вагові коефіцієнти $\tilde{W}^\ell = (\tilde{w}_1, \ldots, \tilde{w}_\ell)$:

$$\tilde{w}_i = \frac{w_i}{\sum_{j=1}^{\ell} w_j}.$$

Зважена кількість помилок N (негативні) і правильних класифікацій P (позитивні) за вектора вагових коефіцієнтів $U^\ell = (u_1, \ldots, u_\ell)$:

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

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

Зважена кількість відмов від класифікації: $(1 - P - N)$.

Теорема AdaBoost

Нехай $\mathcal{B}$ – доволі велике сімейство базових алгоритмів. Нехай для будь-якого нормованого вектора вагових коефіцієнтів $U^\ell = (u_1, \ldots, u_\ell)$ існує алгоритм $b \in \mathcal{B}$, який класифікує вибірку хоча б трішки краще, ніж навмання: $P(b, U^\ell) > N(b, U^\ell)$.
Тоді мінімум функціоналу $\tilde{Q}_T$ досягається за умови

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

$$\alpha_T = \frac{1}{2} \ln \frac{P(b, U^\ell)}{N(b, U^\ell)}.$$

Алгоритм AdaBoost

Кроки реалізації алгоритму AdaBoost.

1. Спочатку всім об'єктам надано однакові значення вагових коефіцієнтів.

2. Модель будується на підвиборці даних.

3. За цією моделлю виходять передбачення для всіх даних.

4. За прогнозами та справжніми значеннями обчислюються помилки.

5. У побудові наступної моделі найбільші для об'єктів, у передбаченні яких алгоритм помилився, збільшують значення вагових коефіцієнтів.

6. Значення вагового коефіцієнту може бути визначене за величиною помилки: чим більша помилка, тим більше значення вагового коефіцієнта.

7. Цей процес повторюється доти, поки функція помилки не перестане змінюватися або поки не буде досягнуто максимальної кількості предикторів.

Вхід: навчальна вибірка X^ℓ, параметри T базових алгоритмів.

Вихід: базові алгоритми b_t, та їх вагові коефіцієнти α_t, $t = 1, \ldots, T$.

1. Ініціювати ваги для всіх об'єктів:
   $w_i = 1/\ell$, $i = 1, \ldots, \ell$.
2. Для всіх $t = 1, \ldots, T$:
   - навчити базовий алгоритм
     $b_T = \arg\min_b N(b, \tilde{W}^\ell)$;
   - порахувати ваговий коефіцієнт алгоритму
     $\alpha_T = \frac{1}{2} \ln \frac{1 - N(b, \tilde{W}^\ell)}{N(b, \tilde{W}^\ell)}$;
   - оновити вагові коефіцієнти об'єктів
     $w_i = w_i \exp(-y_i \alpha_t b_t(x_i))$, $i = 1, \ldots, \ell$;
   - віднормувати вагові коефіцієнти об'єктів
     $w_i = w_i / \sum_{j=1}^{\ell} w_j$, $i = 1, \ldots, \ell$

Рекомендації

Базові класифікатори
Дерева пошуку рішень використовуються найчастіше
Порогові правила «пні» (data stumps)

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

Відсів шуму
Відкинути об'єкти з найбільшими w_i.

Модифікація формули для α_T на випадок $N = 0$

$$\alpha_T = \frac{1}{2} \ln \frac{1 - N(b, \tilde{W}^\ell) + \frac{1}{\ell}}{N(b, \tilde{W}^\ell) + \frac{1}{\ell}}.$$

Додатковий критерій зупинення
Збільшення частоти помилок на контрольній вибірці.

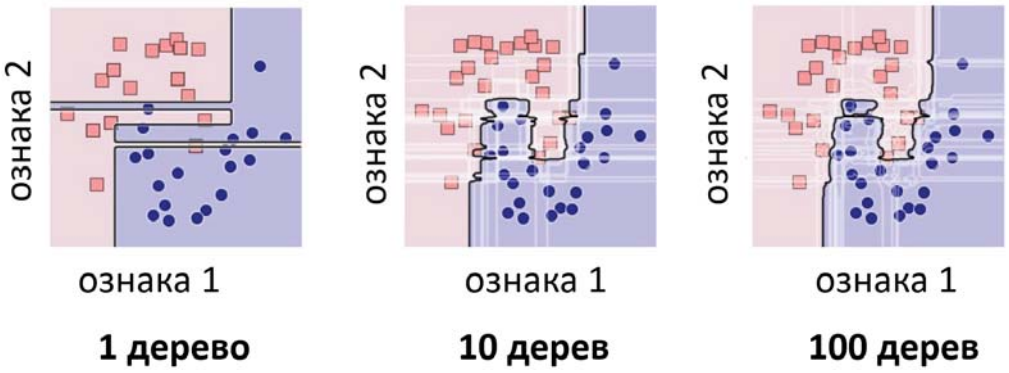
8.4.4 Поставлення задачі

Провести класифікацію даних (або регресійний аналіз) із використанням **boosting**-ансамблю.
Етапи розв'язання

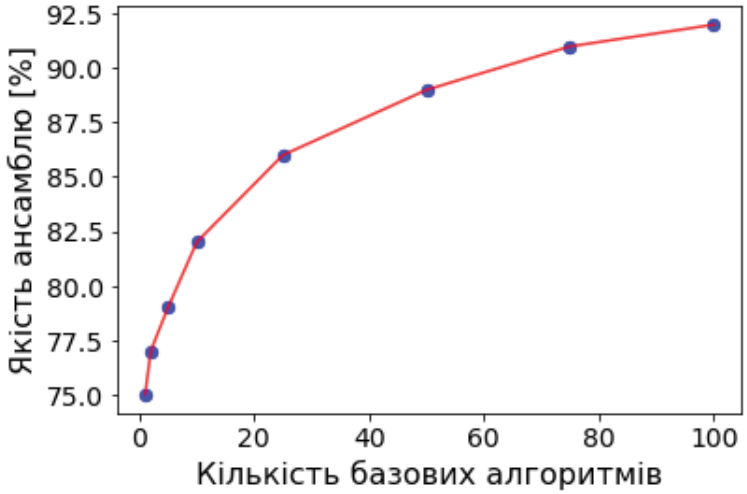
1. Імпортувати вибірку для проведення навчання.
2. Розділити всю вибірку на навчальну та тестову.
3. Обрати базові алгоритми.
4. Побудувати алгоритм навчання на навчальній вибірці.
5. Подати графічно результат класифікації (регресії) для навчальної вибірки.
6. Побудувати залежність точності алгоритму від кількості базових алгоритмів.
7. Перевірити точність роботи алгоритму на тестовій вибірці.
8. Подати графічно результат класифікації для тестової вибірки.
9. Порівняти власні результати з результатами роботи вбудованих алгоритмів з sklearn.
10. Оформити результати у вигляді звіту.

8.4.5 Приклад подання результатів

Розділення вибірки за допомогою бустінгу на деревах

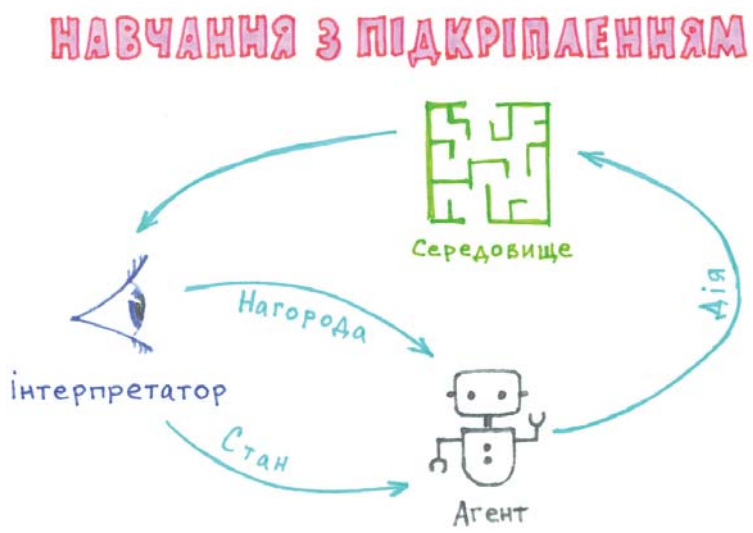


Залежність якості ансамблю від кількості базових алгоритмів



Розділ 9

Навчання з підкріпленням



У цьому розділі буде надано основний теоретичний матеріал щодо підходів навчання з підкріпленням (Reinforcement Learning) у машинному навчанні. Буде проведено огляд основних методів та алгоритмів для навчання агента в середовищі.

Буде поставлено завдання завдання для лабораторних робіт й наведено приклад подання результатів.

9.1 Основні поняття

Навчання з підкріпленням (Reinforcement Learning) – це метод машинного навчання, в якому наша система (агент) навчається методом спроб і помилок. Ідея полягає в тому, що агент взаємодіє із середовищем, паралельно навчаючись, і одержує винагороду за виконання дій.

У навчанні з підкріпленням використовується спосіб позитивної нагороди за правильну дію, щоб спонукати агента, та негативну – за неправильну. Це програмує нашого агента на пошук довгострокової та максимальної загальної винагороди для досягнення оптимального рішення. Ці довгострокові цілі не дають агенту можливості зупинятися на досягнутому. Згодом система вчиться уникати негативних дій і робить лише позитивні.

Навчання з підкріпленням використовується у таких сферах:

- робототехніка для промислової автоматизації (наприклад, конвеєрне складання);
- планування бізнес-стратегії;
- автоматизація всередині машинного навчання;
- просунуті рекомендаційні системи;
- управління рухом робота, автопілот.

Необхідно пам'ятати, що навчання з підкріпленням потребує великих обчислювальних ресурсів і часу, особливо коли простір для дій у моделі великий.

Компоненти системи навчання з підкріпленням

Агент (agent): наша система, яка виконує дії в середовищі, щоб отримати певну винагороду.

Дія (action, a): те, що може зробити агент і що впливає на його стан. Виконання дії завжди приводить агента до наступного стану, який також супроводжується можливою винагородою.

Середовище (environment, E): сценарій/оточення, з яким повинен зіштовхнутися агент.

Нагорода (reward, R): надається агенту після виконання певної дії чи завдання. Є позитивною та негативною, як було згадано вище.

Стан (state, s): належить до поточного положення, що повертається середовищем.

Політика (policy, π): стратегія, яка застосовується агентом для ухвалення рішення про наступну дію на основі поточного стану.

Вартість (value, V): нагорода, що очікується в довгостроковій перспективі. Порівняно з короткостроковою винагородою, береться до уваги знижка (discount).

Функція значення (Value Function): визначає розмір загальної суми нагороди.

Модель середовища (Model of the environment): імітатор поведінки середовища (просто кажучи, демо-версія моделі). Це допомагає визначити, як поводитиметься середовище.

Q-значення або значення дії: відрізняється від V тим, що набуває додаткового параметра – поточну дію.

Із поняттям винагороди пов'язана та обставина, що вона просто показує, наскільки гарною була дія агента, неважливо, правильна вона чи ні. Вона нічого не говорить про те, чи була дія кращою або гіршою, це просто число. Винагорода, яку одержить агент упродовж діяльності, не обов'язково презентує можливу винагороду, яку можна отримати в майбутньому. Можна вести пошук у не найкращій частині простору станів і досягти локального максимуму в 10 очок за глобального максимуму в 1000 очок.

Середовище зазвичай формулюється як марківський процес прийняття рішень (МППР) із кінцевою множиною станів, і в цьому сенсі алгоритми навчання з підкріпленням тісно пов'язані з динамічним програмуванням. Імовірності виграшів і переходу станів у

МППР зазвичай є величинами випадковими, але стаціонарними в межах задачі.

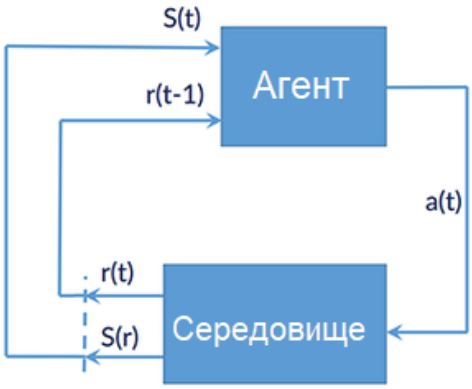
Під час навчання з підкріпленням, на відміну від навчання з учителем, не надаються правильні пари «вхідні дані – відповідь», а ухвалення суб-оптимальних рішень (що дають локальний екстремум) не обмежується явно. Навчання з підкріпленням намагається знайти компроміс між дослідженням невивчених областей і застосуванням наявних знань (exploration vs exploitation).

9.2 Поставлення задачі навчання з підкріпленням

У довільний час t агент характеризується станом $s_t \in S$ і безліччю можливих дій $A(s_t)$. Вибираючи дію $a \in A(s_t)$, він переходить у стан s_{t+1} і одержує виграш r_t. Ґрунтуючись на такій взаємодії з довкіллям, агент, який навчається з підкріпленням, повинен виробити стратегію $\pi : S \to A$, яка максимізує величину $R = r_0 + r_1 + \cdots + r_n$. У разі МППР, що має термінальний стан, або величину: $R = \sum_t \gamma^t r_t$ для МППР без термінальних станів (де $0 \leq \gamma \leq 1$ – дисконтуючий множник для «майбутнього виграшу»). Отже, навчання з підкріпленням особливо добре підходить для вирішення завдань, пов'язаних із вибором між довгостроковою та короткостроковою вигодою.

Взаємодія агента із середовищем:

- ініціалізація стратегії $\pi_1(a|s)$ та стану середовища s_1;
- для всіх $t = 1, \ldots, T$
  - агент вибирає дію $a_t \sim \pi_t(a|s_t)$;
  - середовище генерує нагороду $r_{t+1} \sim p(r|a_t, s_t)$ та новий стан $s_{t+1} \sim p(s|a_t, s_t)$;
  - агент коригує стратегію $\pi_{t+1}(a|s)$.



9.3 Алгоритми

Тепер, коли було визначено функцію виграшу, потрібно визначити алгоритм, який буде використовуватися для знаходження стратегії, що забезпечує найкращий результат. Наївний підхід до вирішення цього завдання передбачає такі кроки:

- випробувати всі можливі стратегії;
- вибрати стратегію з найбільшим очікуваним виграшем.

Перша проблема такого підходу полягає в тому, що кількість доступних стратегій може бути дуже великою або нескінченною. Друга проблема виникає, якщо виграші стохастичні – щоб точно оцінити виграш від кожної стратегії потрібно багаторазово застосувати кожну з них. Цих проблем можна уникнути, якщо допустити деяку структуризацію і, можливо, дозволити результатам, одержаним від спроби однієї стратегії, проводити оцінку для іншої. Двома основними підходами для реалізації цих ідей є оцінка функцій корисності та пряма оптимізація стратегій.

Підхід із використанням функції корисності використовує безліч оцінок очікуваного виграшу лише для однієї стратегії (або поточної, або оптимальної). Водночас намагаються оцінити або очікуваний виграш, починаючи зі стану s, за подальшого дотримання стратегії π,

$$V(s) = E[R|s, \pi], \tag{9.1}$$

або очікуваний виграш після ухвалення рішення a у стані s та подальшому дотриманні π,

$$Q(s, a) = E[R|s, \pi, a]. \tag{9.2}$$

Якщо для вибору оптимальної стратегії використовується функція корисності Q, то оптимальні дії завжди можна вибрати як дії, що максимізують корисність. Якщо ми користуємося функцією V, необхідно мати модель оточення у вигляді ймовірностей $P(s'|s, a)$, що дозволяє побудувати функцію корисності вигляду

$$Q(s, a) = \sum s'V(s')P(s'|s, a), \tag{9.3}$$

або застосувати так званий метод «виконавець-критик», в якому модель ділиться на дві частини: критик, який оцінює корисність стану V, і виконавець, який вибирає відповідну дію в кожному стані.

Маючи фіксовану стратегію π, оцінити $E[R|\cdot]$ за $\gamma = 1$ можна просто усереднивши безпосередні виграші. Найбільш очевидний спосіб оцінювання за $\gamma \in (0, 1)$ – усереднити сумарний виграш після кожного стану. Проте для цього потрібно, щоб МППР досяг термінального стану (завершився). Тому побудова шуканої оцінки за $\gamma \in (0, 1)$ не очевидна. Проте, можна помітити, що R утворюють рекурсивне рівняння Беллмана:

$$E[R|s_t] = r_t + \gamma E[R|s_{t+1}]. \tag{9.4}$$

Підставляючи наявні оцінки V та застосовуючи метод градієнтного спуску з квадратичною функцією помилок, ми приходимо до алгоритму навчання з тимчасовими впливами (temporal difference (TD) learning). У найпростішому випадку стани та дії дискретні і можна дотримуватись табличних оцінок для кожного стану.

Загалом, область навчання з підкріпленням складається з кількох алгоритмів, які використовують різні підходи. Відмінності переважно пов'язані зі своїми стратегіями взаємодії з середовищем.

State-Action-Reward-State-Action (SARSA). Цей алгоритм навчання з підкріпленням починається з надання агенту такого коефіцієнта, як політика (on-policy) – це ймовірність, за допомогою якої алгоритм оцінює шанси певних дій, які приводять до винагород чи позитивних станів.

Q-Learning. У цьому підході Reinforcement Learning використовується протилежний підхід. Агент не одержує політики (on-policy), відповідно, його дослідження середовища є самостійним. У Q-learning ми не маємо обмежень на вибір дії (action) для алгоритму. Він вважає, що всі наступні вибори дії будуть оптимальними за замовчуванням, тому алгоритм робить операцію вибору, виходячи з максимізації оцінки Q.

Deep Q-Networks (Глибокі Q-мережі). Цей алгоритм використовує нейронні мережі на додаток до методів навчання з під-

кріпленням. Нейромережі здійснюють самостійне дослідження середовища для вибору найбільш оптимального значення. Те, як алгоритм поводитиметься і підбиратиме значення, засноване на вибірці минулих позитивних дій, отриманих нейронною мережею.

9.3.1 Алгоритм SARSA

SARSA (State-Action-Reward-State-Action) – алгоритм пошуку стратегії Марковського процесу вирішування, який використовується в області навчання з підкріпленням машинного навчання. Згідно з цим алгоритмом оновлення Q-функції залежить від поточного стану агента s_1, дії a_1, яку агент обирає, винагороди r_1, яку отримує агент за вибір цієї дії, стану s_2, в який переходить агент після виконання цієї дії, та, нарешті, наступної дії a_2, яку агент обирає виходячи зі свого нового стану.

За алгоритмом SARSA, агент взаємодіє з середовищем та оновлює стратегію згідно з виконаними діями, отже, цей алгоритм можна віднести до класу алгоритмів навчання за стратегією. Значення Q-функції для дії та стану оновлюється відповідно похибці, що регулюється за допомогою коефіцієнта швидкості навчання α:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \qquad (9.5)$$

Значення Q-функції презентує сумарну винагороду, яку можна отримати за весь час, що залишився, в межах цього епізоду, за умови виконання дії a в стані s, з додаванням знеціненої винагороди за виконання дій у наступному стані.

Оскільки SARSA є ітераційним алгоритмом, він передбачає наявність початкових умов до того, як відбудеться перше оновлення. Низьке (нескінченне) початкове значення, також відоме як «оптимістичні початкові умови» може заохочувати дослідження: незалежно від того, які дії виконує агент, формула оновлення призводить до того, що наступні ітерації мають більш високі значення винагороди, ніж попередні, тим самим збільшуючи ймовірність їх вибору. Одна з ідей – використання першої винагороди r як початкових умов. За такого підходу, після виконання агентом першої

дії, одержана винагорода використовується як початкове значення Q. За фіксованих винагород, це дозволяє навчати відразу після першого кроку. Такий метод обирання початкових умов повторює поведінку людини в багатьох експериментах із бінарним вибором.

9.3.2 Алгоритм Q-learning

Q-навчання (Q-learning) – це алгоритм безмодельного навчання з підкріпленням. Метою Q-навчання є навчитися стратегії, яка показує агентові, до якої дії вдаватися за яких обставин. Воно не вимагає моделі середовища (звідси уточнення «безмодельного»), і може розв'язувати задачі зі стохастичними переходами та винагородами, не вимагаючи пристосувань.

Важливою передумовою для цього методу є Марківська властивість – кожна дія залежить лише від попередньої. Це дозволяє ухвалити рішення з урахуванням кількох станів, а не всього набору можливих станів. Для будь-якого скінченного марковського процесу вирішування Q-навчання знаходить стратегію, яка є оптимальною в тому сенсі, що вона максимізує очікуване значення повної винагороди над будь-якими та усіма послідовними кроками, починаючи з поточного стану. Q-навчання може визначати оптимальну стратегію обирання дій за умови нескінченного часу на розвідування та частково випадкової стратегії.

Навчання ґрунтується на Q-таблиці, яка для кожного стану s та кожної можливої дії a з цього стану а зберігає Q-значення. Q-таблиця ініціалізується нулями або випадковими числами та заповнюється в процесі навчання. Заповнення таблиці відбувається з урахуванням рішень агента. Він може вибирати дії за допомогою exploitation (експлуатація) та exploration (дослідження). У режимі експлуатації агент вибирає дію з максимальним значенням Q:

$$a_{t+1} = \arg\max_{a_t} Q(s_t, a_t). \tag{9.6}$$

У режимі дослідження дія вибирається випадково. Вибраною дією агент переходить у наступний стан s_{t+1}. Випадкові вибори дають змогу заповнювати нові частини таблиці. Вибір між режимами відбувається випадково. Насправді можливість експлуатації встановлюють вищою.

На основі одержуваної від середовища винагороди агент формує функцію корисності Q, що згодом дає можливість вже не випадково вибирати стратегію поведінки, а враховувати досвід попередньої взаємодії з середовищем. Одна з переваг Q-навчання – те, що воно може порівняти очікувану корисність доступних дій, не формуючи моделі довкілля.

Отже, алгоритм – це функція якості від стану та дії:

$$Q: \ S \times A \to R. \tag{9.7}$$

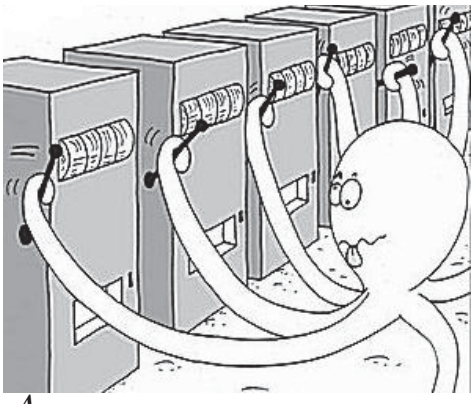
У кожний момент часу t агент вибирає дію a_t, одержує нагороду r_t, переходить у новий стан s_{t+1}, який може залежати від попереднього стану s_t та обраної дії, та оновлює функцію Q. Оновлення функції використовує зважене середнє між старим і новим значеннями:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a_t)\right), \tag{9.8}$$

де r_t – це нагорода, одержана під час переходу зі стану s_t у стан s_{t+1}, і α – швидкість навчання. Майбутні значення дисконтуються з коефіцієнтом $0 < \gamma < 1$ – таким чином близькі нагороди важливіші за далекі, і результатом оптимізації є найкоротший шлях до найбільшого значення. Алгоритм закінчується, коли агент переходить у термінальний стан s_{t+1}.

9.4 Стратегії

9.4.1 Задача про багаторукого бандита (The multi-armed bandit problem)



Розглянемо модельну задачу для розуміння конфлікту між exploitation (використання) та exploration (дослідження). У нас є автомат – N-рукий бандит, на кожному кроці ми вибираємо за яку з N ручок автомата смикнути, тобто безліч дій $A = 1, 2 \ldots, N$. Вибір дії a_t на кроці t тягне нагороду $R(a_t)$, водночас $R(a) \forall a \in A$ є випадкова величина,

розподіл якої невідомий. Стан середовища у нас від кроку до кроку не змінюється, а отже безліч станів S тривіально, ні на що не впливає, тому його можна проігнорувати.

Для простоти будемо вважати, що кожній дії відповідає деякий розподіл, який не змінюється з часом. Якби ми знали ці розподіли, то очевидна стратегія полягала б у тому, щоб підрахувати математичне сподівання для кожного з розподілів, вибрати дію з максимальним математичним сподіванням і надалі робити це на кожному кроці. Проблема в тому, що розподіл невідомий, однак можна оцінити математичне сподівання деякої випадкової величини ξ з невідомим розподілом. Для K експериментів ξ_k, оцінка математичного сподівання – це середнє арифметичне результатів експериментів: $E(\xi) = (1/K)\sum_k \xi_k$.

Формулювання задачі

- A – безліч можливих дій (ручок автомата);
- $p_a(r)$ – невідомий розподіл нагороди $r \in R \forall a \in A$;
- $\pi_t(a)$ – стратегія агента в момент t $\forall a \in A$.

Гра агента з середовищем:

- ініціалізація стратегії $\pi_1(a)$;
- для всіх $t = 1, \ldots, T$:
  - агент вибирає дію (ручку) $a_t \sim \pi_t(a)$;
  - середовище генерує нагороду $r_t \sim p_{a_t}(r)$;
  - агент коригує стратегію $\pi_{t+1}(a)$.

Середня нагорода в t іграх:

$$Q_t(a) = \frac{\sum_{i=1}^{t} r_i[a_i = a]}{\sum_{i=1}^{t}[a_i = a]} \to \max.$$

Цінність дії a:

$$Q^*(a) = \lim_{t\to\infty} Q_t(a) \to \max.$$

9.4.2 «Жадібна» (greedy) стратегія

- $P_a = 0 \ \forall a \in \{1, \dots, N\}$ – скільки разів було обрано дію a;
- $Q_a = 0 \ \forall a \in \{1, \dots, N\}$ – поточна оцінка математичного сподівання нагороди для дії a.

На кожному кроці t

- обираємо дію з максимальною оцінкою математичного сподівання:
  $a_t = \arg\max\limits_{a \in A} Q_a$;
- виконуємо дію a_t та одержуємо нагороду $R(a_t)$;
- оновлюємо оцінку математичного сподівання для дії a_t:
  $P_{a_t} = P_{a_t} + 1$,
  $Q_{a_t} = Q_{a_t} + \frac{1}{P_{a_t}}(R(a_t) - Q_{a_t})$.

Недолік

Нехай у нас є «дворукий» бандит. Перша ручка завжди видає нагороду таку, що дорівнює 1, друга – завжди видає 2. Діючи згідно з жадібною стратегією ми смикнемо на початку першу ручку, тому що на початку оцінки математичних сподівань дорівнюють нулю, збільшимо її оцінку до $Q_1 = 1$. Надалі завжди вибиратимемо першу ручку, а отже на кожному кроці будемо отримувати на 1 менше, ніж могли б.

У цьому разі достатньо спробувати на початку кожну ручку замість того, щоб фокусуватися лише на одній. Але якщо винагорода випадкова величина, то одиничної спроби буде мало.

9.4.3 ϵ-«жадібна» (ϵ-greedy) стратегія

Уведемо параметр $\epsilon \in (0, 1)$.
На кожному кроці t:

- одержимо значення α – випадкової величини рівномірно розподіленої на відрізку $(0, 1)$;
- якщо $\alpha \in (0, \epsilon)$, то виберемо дію $a_t \in A$ випадково й рівноймовірно, інакше як у «жадібній» стратегії виберемо дію з максимальною оцінкою математичного сподівання;
- оновлюємо оцінки так само, як у «жадібній» стратегії.

Якщо $\epsilon = 0$, то це проста «жадібна» стратегія. Проте якщо $\epsilon > 0$, то на відміну від «жадібної» стратегії на кожному кроці з ймовірністю ϵ, буде обиратися випадкова дія – дослідження середовища.

9.4.4 Стратегія Softmax

Основна ідея алгоритму softmax – зменшення втрат під час дослідження за рахунок більш рідкісного вибору дій, які мають невелику нагороду в минулому. Щоб цього домогтися для кожної дії обчислюється ваговий коефіцієнт з урахуванням якого відбувається вибір дії. Чим більше $Q_t(a)$, тим більша ймовірність вибору a:

$$\pi_{t+1}(a) = \frac{\exp(Q_t(a)/\tau)}{\sum\limits_{b \in A} \exp(Q_t(b)/\tau)},$$

$\tau \in (0, \infty)$ – параметр, за допомогою якого можна налаштовувати поведінку алгоритму.

За $\tau \to \infty$ стратегія прагне рівномірної, тобто softmax менше залежатиме від значення виграшу і вибиратиме дії більш рівномірно (exploration).

При $\tau \to 0$ стратегія прагне жадібної, тобто алгоритм більше орієнтуватиметься на відомий середній виграш дій (exploitation).

Експонента використовується для того, щоб ця вага була ненульовою навіть у дій, нагорода від яких поки що нульова. Евристика: є сенс зменшувати параметр τ з часом.

9.4.5 Метод UCB (upper confidence bound)

Попередні алгоритми під час ухвалення рішення використовують дані про середній виграш. Проблема в тому, що якщо дія дає нагороду з якоюсь імовірністю, то дані від спостережень виходять зашумлені і ми можемо неправильно визначати найвигіднішу дію.

Алгоритм верхнього довірчого інтервалу (upper confidence bound або UCB) – сімейство алгоритмів, які намагаються вирішити цю проблему, використовуючи під час вибору дані не лише про середній виграш, а й про те, наскільки можна довіряти значенням виграшу.

За аналогією із softmax в UCB під час вибору дії використовується ваговий коефіцієнт, який є верхньою межею довірчого інтервалу (upper confidence bound) значення виграшу:

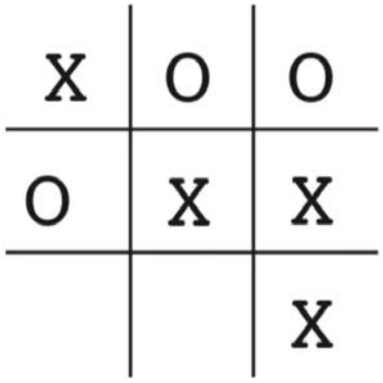
$$\pi_{t+1}(a) = Q_t(a) + b_a, \tag{9.9}$$

де $b_a = \sqrt{(2 \ln \sum_a P_a / P_a)}$ – бонусне значення, яке показує, наскільки недосліджена дія порівняно з іншими.

На відміну від попередніх алгоритмів, UCB не використовує в своїй роботі ні випадкові числа для вибору дії, ні параметри, якими можна впливати на його роботу. На початку роботи алгоритму кожна з дій вибирається по одному разу (для того, щоб можна було обчислити розмір бонусу для всіх дій). Після цього в кожний момент часу вибирається дія з максимальним значенням вагового коефіцієнта.

Незважаючи на відсутність випадковості, результати роботи цього алгоритму виглядають досить шумно порівняно з іншими. Це відбувається через те, що цей алгоритм дуже часто вибирає недосліджені дії.

9.5 Гра «хрестики-нулики» (Tic-Tac-Toe)



Саме тоді, як у загальних методах теорії ігор, наприклад алгоритм "min-max", алгоритм завжди припускає ідеального супротивника, який настільки раціональний, що кожен крок, який він робить, спрямований на максимізацію своєї винагороди та мінімізацію винагороди нашого агента, в навчанні з підкріпленням він навіть не передбачає моделі суперника, і результат може бути напрочуд хорошим.

Розглядаючи опонента як частину середовища, з яким агент може взаємодіяти, після певної кількості ітерацій агент може планувати наперед без будь-якої моделі агента чи середовища або проводити будь-який пошук можливих майбутніх дій чи станів. Перевага очевидна в тому, що метод економить зусилля, пов'язані зі складною математичною дедукцією або дослідженням великої кількості пошукового простору, але він здатний досягти найвищих навичок, просто пробуючи та навчаючись.

Етапи:

- навчити двох агентів грати один проти одного та зберегти досвід;
- завантажити досвід і грати проти агента.

9.5.1 Методика навчання

Уведемо поняття епізоду – одного прогону гри. Наприклад, гру в хрестики-нулики ми починаємо з порожньої дошки. Як тільки один із гравців поставить три своїх знаки в ряд, це означатиме кінець епізоду. Агент навчання з підкріпленням буде навчатися на безлічі епізодів, і, скажімо, після 1000, 10 000 або 100 000 зіграних епізодів ми, ймовірно, навчимо розумного агента. Необхідний для цього час, звичайно, є гіперпараметром і залежить від самої гри,

кількості станів, наскільки в грі великий елемент випадковості і так далі. Гра в хрестики-нулики називається епізодичною задачею, оскільки в неї можна грати знову та знову. Цим вона відрізняється від безперервної задачі, яка ніколи не закінчується. Коли ми говоримо про закінчення епізоду, це означає, що є певні стани в просторі станів, які показують нам, що гра закінчена. Для хрестиків-нуликів це коли один із двох гравців поставив три знаки в ряд або коли поле для гри заповнене. Такі стани називаються кінцевими (термінальними). Отже, кінцевий стан – це стан, в якому більше не можна зробити ніяких дій, а епізод закінчений. У хрестиках-нуликах – це означає, що або перший гравець виграв, або другий, або нічия.

Визначення правил

У кожній грі існують правила. Для гри «хрестики-нулики» – це визначення конфігурації ігрового поля, за якої агент виграє, чи програє, чи буде нічия. Виграшні конфігурації – розміщення у ряд символів агента (8 конфігурацій); програшні – розміщення у ряд символів агента супротивника (8 конфігурацій); нічия – конфігурація поля, коли відсутні пусті клітини для здійснення ходу і жодна виграшна чи програшна конфігурація не реалізується. Цінності таких станів є фіксованими в продовж усього етапу навчання. Зазвичай вважається, що цінності станів, в яких агент виграв r_w, фіксуються значенням $r_w = 1$, відповідно цінність $r_l = 0$ для станів, де агент програв. Існують певні варіації щодо того, яку цінність надати термінальному станові, коли реалізується нічия: такий самий негативний результат як і програш ($r_d = 0$); або певний успіх ($r_d < r_w$), проте не такий як виграш. Значення цих цінностей можна змінювати, тим самим тренувати агента з різним рівнем майстерності.

Ініціалізація станів

Для прикладу розглянемо стандартний варіант гри на полі (дошці) розміром 3×3. Вже для такої системи маємо приблизно 3^9 можливих станів із різною комбінацією хрестиків, нуликів та не заповнених клітин. Для цієї задачі функція цінності має один

аргумент – стан (конфігурація ігрового поля): $V(s)$. Це очікуване значення всіх майбутніх винагород за поточного стану s або, іншими словами, середнє значення всіх можливих майбутніх винагород за заданого поточного стану s. Отже, перше, що потрібно зробити – ініціювати функцію цінності. Оскільки на початку навчання наш агент ще не має жодного накопиченого досвіду, то цінність кожного можливого стану однакова.

Функція цінності

Після кожного зіграного епізоду кожен з агентів оновлює цінності своїх ходів, які він робив у продовж епізоду. Інтуїтивно, якщо всі ходи агента привели його до виграшу, то цінності всіх ходів потрібно збільшити і навпаки. Рівняння оновлення має нескладний вигляд, але в ньому прихований ряд неочевидних тонкощів. Виглядає воно схоже на градієнтний спуск. Ми беремо функцію цінності в стані $V(s)$ і оновлюємо її, додаючи до неї коефіцієнт навчання α, помножений на різницю $V(s^T) - V(s)$:

$$V(s) \leftarrow V(s) + \alpha \cdot \big(V(s^T) - V(s)\big)\,,$$

де $V(s^T)$ – цінність термінального стану (виграшу, програшу чи нічиєї). Тонкість у тому, що s тут означає всі стани, які нам трапилися в епізоді. Це означає, що з кожної ітерації циклу ми насправді маємо зіграти й зберегти всі стани, в яких побували. Потім ми по циклу перебираємо кожен із станів історії станів і оновлюємо цінність, використовуючи вищезазначене рівняння.

Експлуатація проти розвідки

Одним із фундаментальних компромісів у навчанні підкріплення є компроміс між експлуатацією та розвідкою. Експлуатація означає вибір дії, яка максимізує нашу нагороду (може призвести до застрягання в локальному оптимумі). Дослідження (розвідка) означає вибір дії незалежно від винагороди, яку вона дає (це допомагає нам виявити інший локальний оптимум, який може привести нас ближче до глобального оптимуму). Водночас, будь-яка експлуатація може призвести до неоптимального агента, а всі дослідження

просто дадуть нам не навченого агента, який продовжує робити випадкові дії.

Стратегія, що широко використовується для вирішення цієї проблеми полягає в такому:

- ініціалізація певної змінної ε зі значенням від 0 до 1 (зазвичай близько 0,3);
- перед здійсненням кожної дії генерується випадкове число $\xi \in (0, 1)$;
- у разі $\xi \leq \varepsilon$ агент робить випадкову дію – дослідження (розвідка); якщо $\xi > \varepsilon$ агент обирає дію відповідно до максимальної цінності ходу – експлуатація;
- зі збільшенням кількості епізодів (зіграних партій) значення ε зменшується – агенту надається перевага використовувати накопичений досвід.

Використовуючи цю стратегію, агент може досліджувати найкращі дії на ранніх етапах навчання, а потім використовувати найкращі дії на пізніших етапах гри.

Хід гри двох агентів

Спочатку потрібно зафіксувати глобальні параметри, а саме:

- максимальну кількість епізодів T;
- крок навчання α;
- функцію цінності термінальних станів $V(s^T) = \{r_w, r_l, r_d\}$.

Перший епізод

Оскільки цінність кожного нового стану однакова, то в першому епізоді кожен агент робить випадкові ходи, займаючи пусті клітини та зберігаючи історію ходів (станів). Наприкінці епізоду визначається результат гри (виграш/програш/нічия) і відповідно до зафіксованих нагород за термінальний стан перераховується функція цінності для кожного ходу (конфігурації ігрового поля), який

зробив агент. Тепер у кожного агента (гравця) є перший досвід. Зі збільшенням кількості епізодів цей досвід буде накопичуватися.

Наступні епізоди

Відповідно до значення встановленого порогу для дослідження (ε) кожен агент або робить випадковий хід, або використовує накопичений досвід, обираючи хід із максимальною цінністю з усіх можливих для кожного стану (пусті клітини на ігровому полі). Зі збільшенням кількості епізодів поріг для дослідження бажано зменшувати і на останніх епізодах дати можливість агентам грати лише з використанням накопиченого досвіду. Проте стратегія зміни порогу ε є суто індивідуальною.

Кінець навчання

Після закінчення останнього епізоду кожен агент має свій накопичений досвід, який можна використовувати для гри з людиною. Як показує практика, накопичений досвід буде містити набагато менше конфігурацій ніж 3^9.

Гра агента з людиною

Для гри з навченим агентом потрібно завантажити його досвід і спробувати виграти. Водночас, оскільки два агенти навчалися в однакових умовах, то і досвід обох є приблизно однаковим. Якщо функція цінності термінального стану $V(s^T)$ та імовірність зробити випадковий крок ε для кожного агента різні, то можна навчити двох різних агентів із різним рівнем майстерності.

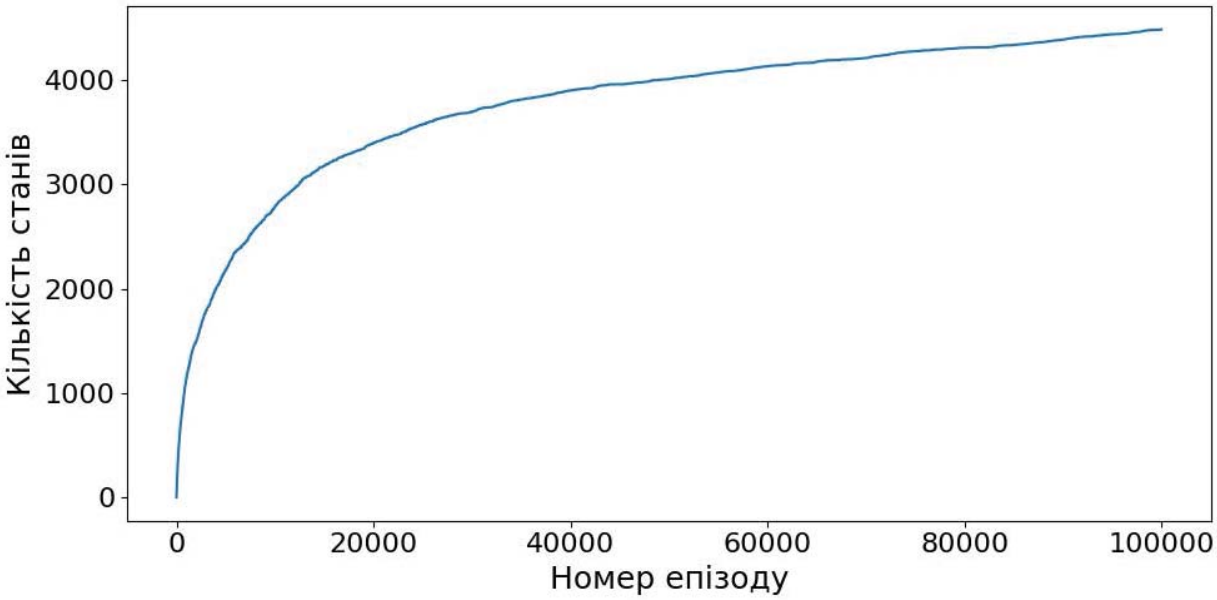
9.5.2 Поставлення задачі

Реалізувати гру «хрестики-нулики» з використанням підходу машинного навчання з підкріпленням.
Хід виконання

1. Зафіксувати глобальні параметри:
   - розмір дошки;
   - максимальну кількість епізодів T;
   - крок навчання α;
   - імовірність випадкового ходу ϵ;
   - функцію цінності термінальних станів $V(s^T) = \{r_w, r_l, r_d\}$.
2. Забезпечити незалежність алгоритму від розміру ігрового поля ($n \times n$, $n = 3, 4, \ldots$).
3. Навчати двох агентів упродовж T епізодів.
4. Зберегти досвід кожного з агентів.
5. Побудувати графічно залежність досвіду (кількість відомих станів) від кількості епізодів навчання.
6. Проілюструвати хід гри людини з агентом. Для графічної ілюстрації можна використати пакет `Tkinter`.
7. Результати оформити у вигляді звіту.

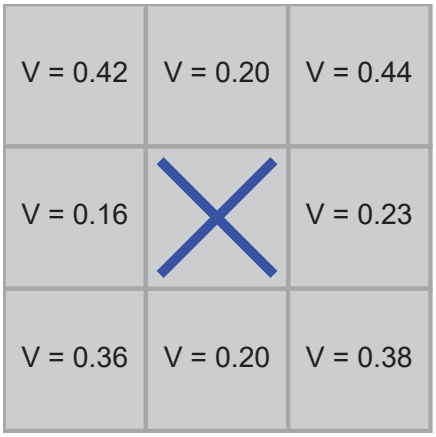
9.5.3 Приклад подання результатів

Залежність досвіду від кількості епізодів навчання

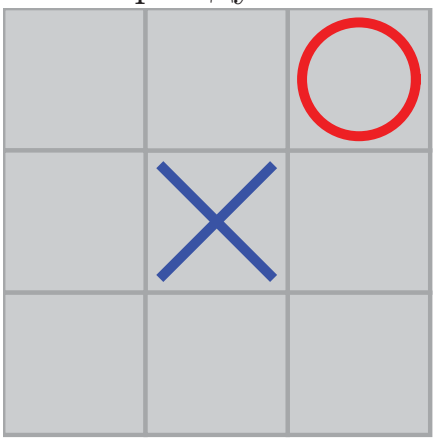


Гра агента з людиною. Людина грає X. Агент грає O.
1. Людина робить перший хід. Результат – нічия.

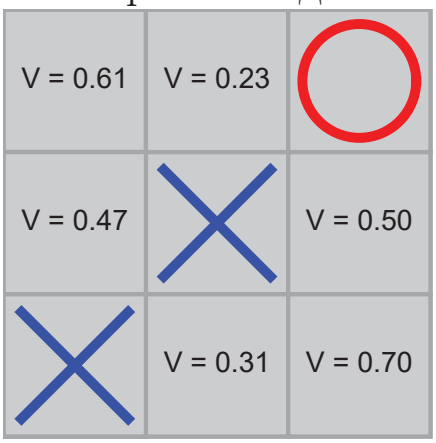
Цінності різних ходів агента



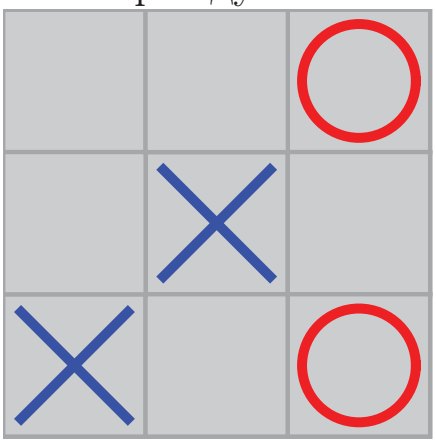
Вибір ходу агента



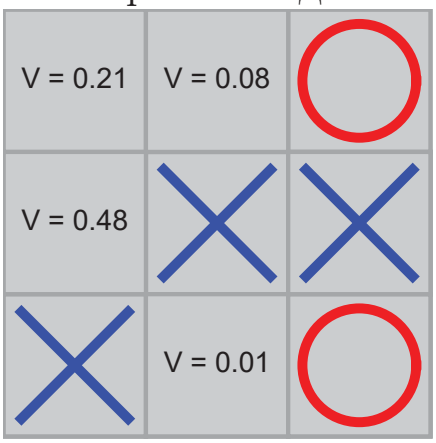
Цінності різних ходів агента



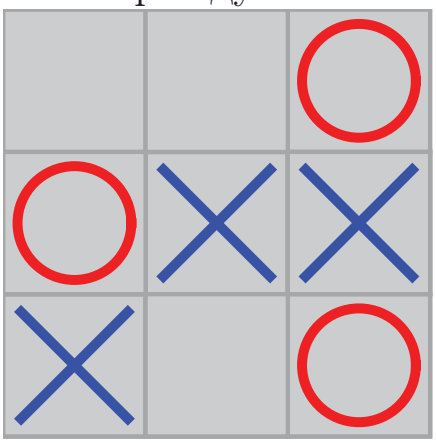
Вибір ходу агента



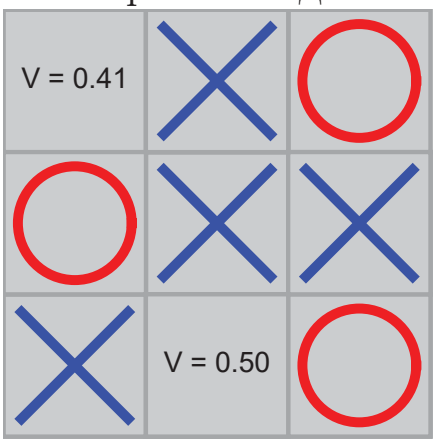
Цінності різних ходів агента



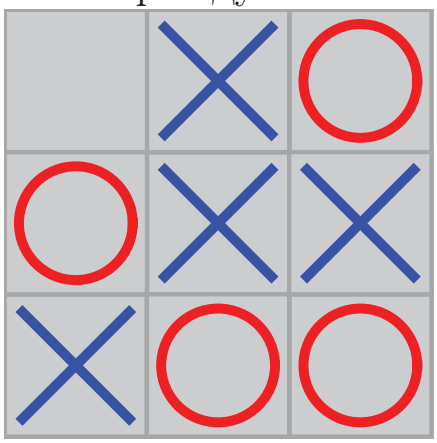
Вибір ходу агента



Цінності різних ходів агента



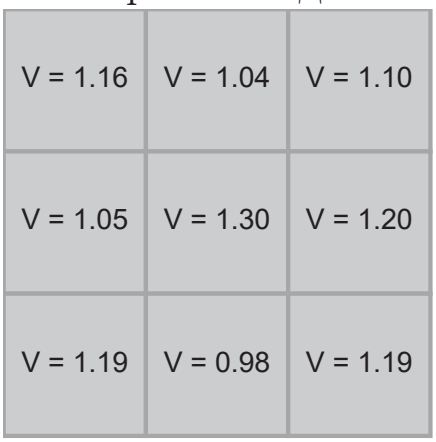
Вибір ходу агента



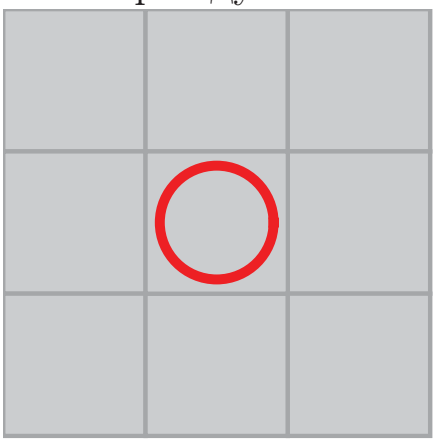
Нічия.

2. Агент робить перший хід. Результат – виграш агента.

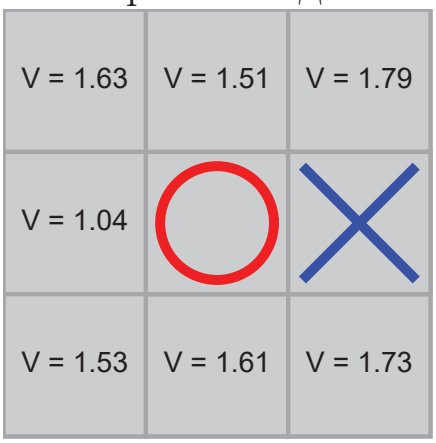
Цінності різних ходів агента



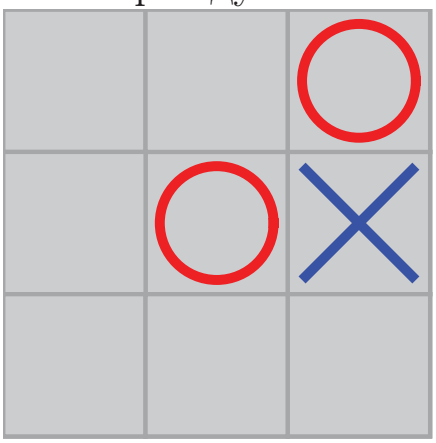
Вибір ходу агента



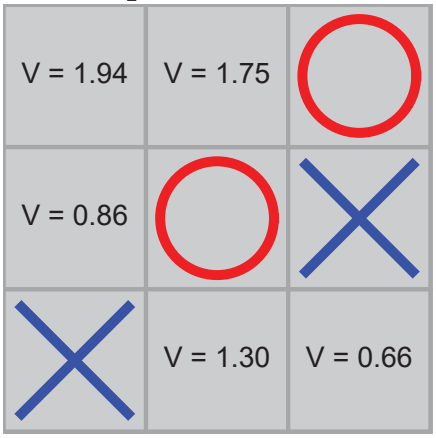
Цінності різних ходів агента



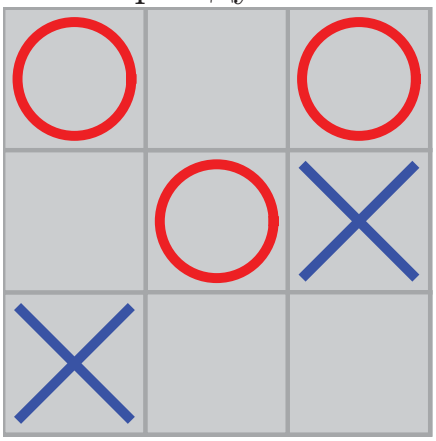
Вибір ходу агента



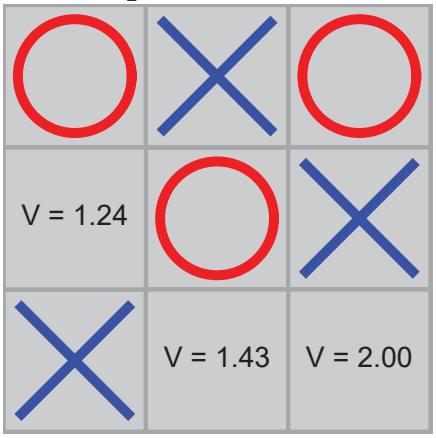
Цінності різних ходів агента



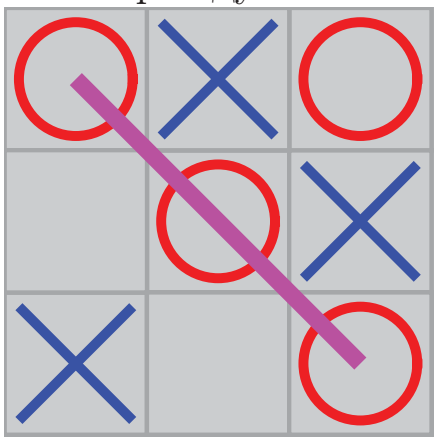
Вибір ходу агента



Цінності різних ходів агента



Вибір ходу агента

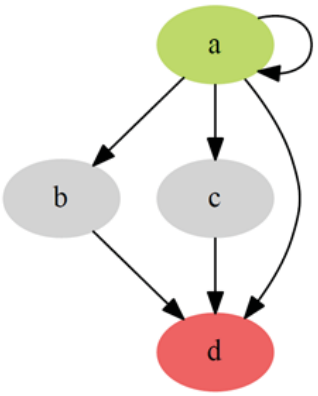


Виграш агента.

9.6 Задача пошуку мети з Q-learning

Розглянемо найпростіший приклад. Ми хочемо знайти найкоротший шлях із вершини, a (позначена зеленим) до вершини d (позначена червоним) на деякому графі G.

Задаємо винагороди в R-таблиці:



- залишатися на місці (виживання) коштуватиме 5 одиниць;
- проходження через існуюче ребро коштуватиме 15 одиниць;
- попадання у фінальну вершину d коштуватиме 100 одиниць;
- за спробу перейти не існуючими ребрами – штраф -100 одиниць.

Під час попадання у вершину d прохід графом буде зупинятися.
Зафіксуємо $\gamma = 0,8$, $\alpha = 1,0$.
Таблиця нагород для агента

R	a	b	c	d
a	5	15	15	100
b	-100	-100	-100	100
c	-100	-100	-100	100
d	-100	-100	-100	0

Ініціюємо Q-таблицю нулями

Q	a	b	c	d
a	0	0	0	0
b	0	0	0	0
c	0	0	0	0
d	0	0	0	0

Перший прохід буде фактично випадковим. Нехай обрано шлях $< a, b, d >$. Через нульові значення Q агент заповнює таблицю значеннями нагород.

$$\begin{aligned} Q_1(a,b) &= Q_0(a,b) + [R(a,b) + \gamma \max_{\rightarrow} Q(b, \rightarrow) - Q_0(a,b)] = \\ &= 0 + [15 + \gamma \cdot 0 - 0] = 15 \end{aligned}$$

$$\begin{aligned} Q_1(b,d) &= Q_0(b,d) + [R(b,d) + \gamma \max_{\rightarrow} Q(d, \rightarrow) - Q_0(b,d)] = \\ &= 0 + [100 + \gamma \cdot 0 - 0] = 100 \end{aligned}$$

Перші Q-значення для шляху $< a, b, d >$

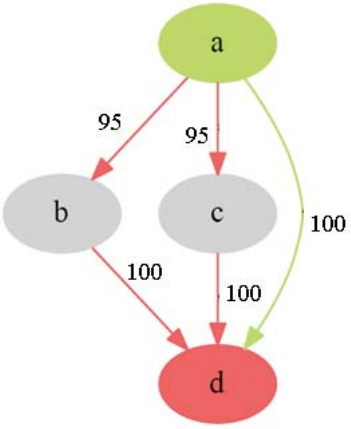
Q	a	b	c	d
a	0	15	0	0
b	0	0	0	100
c	0	0	0	0
d	0	0	0	0

Далі за exploitation агент вибирає шлях $< a, b, d >$. $Q_2(a,b) = 15 + [15 + 0,8 \cdot 100 - 15] = 95$.
$Q_2(b,d) = 100 + [100 + 0,8 \cdot 0 - 0] = 100$.
Оновлена Q-таблиця

Q	a	b	c	d
a	0	95	0	0
b	0	0	0	100
c	0	0	0	0
d	0	0	0	0

за випадкового вибору шляху $< a, c, d >$ його значення сходяться так само. Далі агент вибирає випадково шлях $< a, d >$.
Підсумкові значення Q-таблиці та фінальний граф.

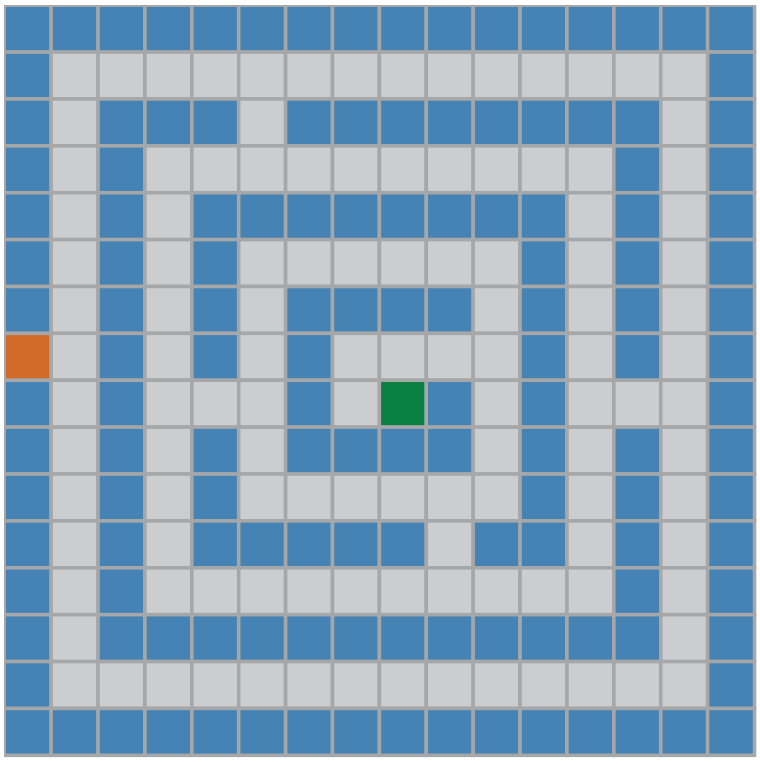
Q	a	b	c	d
a	0	95	95	100
b	0	0	0	100
c	0	0	0	100
d	0	0	0	0



Для пошуку оптимального шляху потрібно з вершини a через максимальні значення Q-таблиці прийти у вершину d: оптимальний шлях - $< a, d >$.

9.6.1 Приклад двовимірної проблеми

Нехай агент знаходиться у початковому стані (зелена комірка) у двовимірному лабіринті розміром 16×16. Йому треба найкоротшим шляхом потрапити до виходу (помаранчева комірка).



Визначимо для нього правила:

- він може рухатися лише на один квадрат за 1 хід;
- пересування за діагоналями заборонено;
- він не хочете одержати пошкодження, тому він не повинен попадати на синій квадрат, «Стіна»;
- він не може покинути межі приміщення, тому потрібно врахувати граничні умови.

Агент виконує дію, а середовище дає йому наслідок цієї дії. Загалом, середовище повертає наступний стан, нагороду та прапорець, який сигналізує про завершення досягання мети.

Таблиця винагород

Спершу потрібно ініціювати нагороди за потрапляння в кожний стан простору:

- агенту потрібно якнайшвидше досягти мети, тому за кожен крок він одержує покарання у вигляді штрафу -1 бал;
- за спробу збити стіну (потрапити на сині квадрати) або за межі простору він отримує покарання у вигляді штрафу -100 балів;
- якщо агент досягне мети, тоді винагорода буде просто нульовою, і світ сигналізуватиме, що мету досягнуто.

Q-таблиця цінностей

Тепер побудуємо Q-таблицю, яка вимірює наскільки добре буде виконати певну дію в будь-якому стані (ми можемо виміряти це за допомогою простого скалярного значення, тому чим більше значення, тим краща дія).

- У світі з 256 станів дозволено 4 дії (вгору, вниз, вліво, вправо), тому Q-таблиця буде розміром 256×4.

- Значення, що зберігаються в цій таблиці – Q-значення – оцінка суми майбутніх нагород. Іншими словами, вони оцінюють, скільки ще винагороди ми можемо одержати до кінця гри, перебуваючи в стані s і виконуючи дію a.

- Таблиця ініціалізується випадковими значеннями (або деякою константою, наприклад, нулями).

- Значення Q для кінцевого (термінального) стану дорівнює нулю. Агент не може робити жодних дій у цьому стані, тому значення для всіх дій у цьому стані вважають такими, що дорівнюють нулю.

Оновлення Q-таблиці відбувається після здійснення кожної дії за формулою

$$Q(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right].$$

При досягненні мети агент повертається у початковий стан і починається наступний епізод навчання.

Перехід у новий стан

Для кожного епізоду ми повинні ініціалізувати початковий стан – повернути агента у вихідне положення. Для кожного кроку (щоразу, коли нам потрібно діяти) агент обирає дію, яка переводить його в наступний стан.

Шляхом аналізу Q-таблиці. Виконати дію, яка має найбільшу цінність у кожному стані

$$a = \arg\max_{a'} Q(s,a).$$

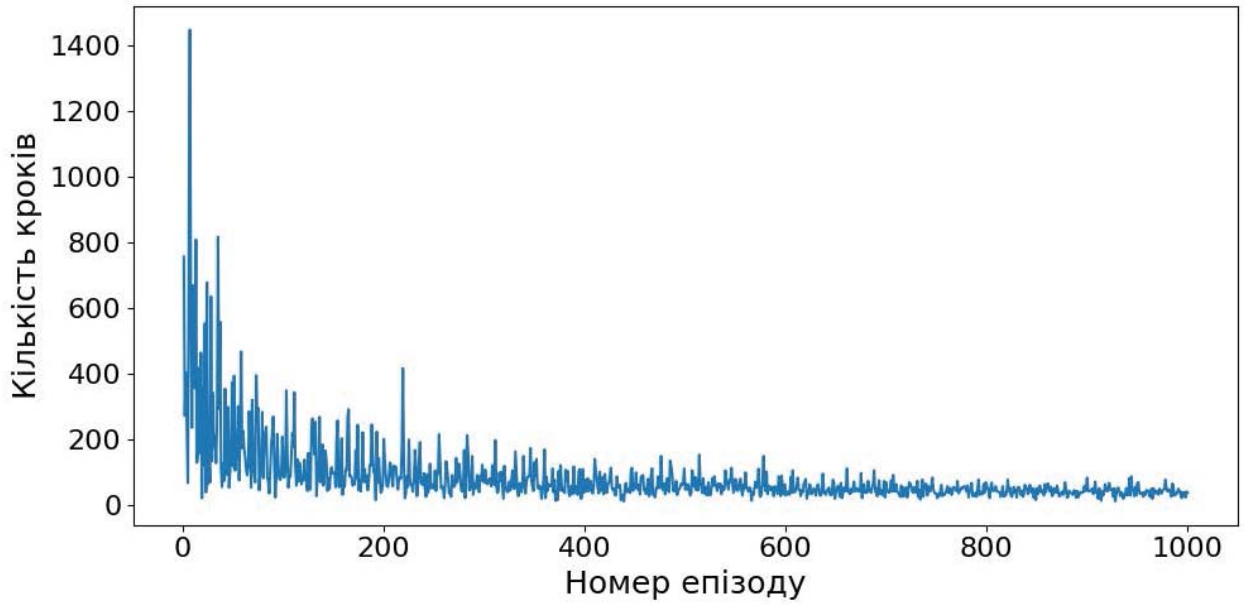
Він використовує одержані Q-значення для створення стратегії. У цьому разі це буде «жадібна» стратегія, тому що агент завжди робить дію, яка, на його думку, найкраща в кожному стані.

Шляхом дослідження середовища. Агенту потрібно переконатися, що він досить добре вивчив середовище. Для цього використовується ϵ-«жадібна» стратегія – агент повинен діяти жадібно, але робити випадкові дії з певною ймовірністю ϵ, таким чином, за нескінченної кількості спроб він повинен дослідити всі можливі стани.

Після певної кількості епізодів навчання агент вивчить весь доступний простір станів і сформує остаточну Q-таблицю, значення в якій перестануть змінюватися і дозволяють агенту робити найкращі дії в кожному стані, даючи йому в наслідок цього найкращий і найкоротший шлях до мети. Незмінність значень цінностей дій у Q-таблиці може бути сигналом до того, що процедуру навчання агента можна завершувати.

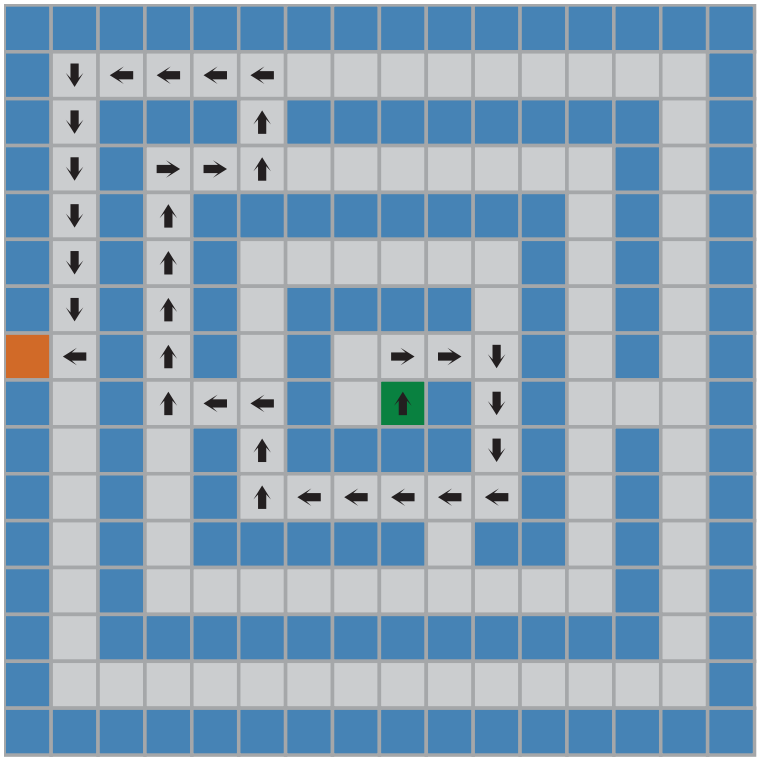
Ілюстрація результатів

Типова залежність кількості кроків, яку робить агент для досягнення мети від номера епізоду навчання



Тепер, використовуючи накопичений досвід – всю Q-таблицю, можна запустити агента в середовище й проаналізувати обраний ним оптимальний шлях.

Одержаний після 1000 епізодів оптимальний шлях від початкового до кінцевого стану.



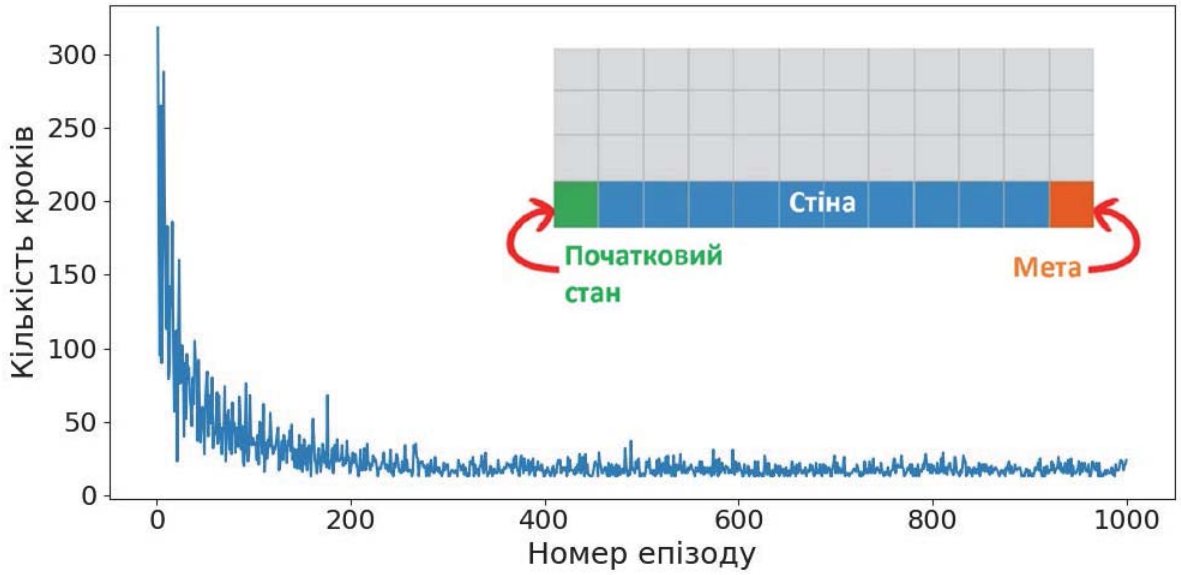
9.6.2 Поставлення задачі

Провести навчання агента знаходити оптимальний і найшвидший шлях до мети з використанням методу Q-навчання.
Етапи виконання

1. Ініціалізувати простір станів:
   - визначити початковий і термінальний стани;
   - розмістити перешкоди для пересування агента;
   - ініціалізувати R-таблицю винагород для кожного стану;
   - ініціалізувати Q-таблицю.
2. Зафіксувати глобальні параметри:
   - максимальну кількість епізодів;
   - темп навчання α;
   - коефіцієнт дисконту γ;
   - імовірність робити випадковий крок ϵ;
   - правило зменшення ймовірності робити випадковий крок з кількістю епізодів (за бажанням).
3. Провести навчання агента.
4. Побудувати залежність кількості кроків агента від епізодів.
5. Подати графічно найоптимальніший маршрут агента. Для графічної ілюстрації можна використати пакет `Tkinter`.
6. Оформити результати у вигляді звіту.

9.6.3 Приклад подання результатів

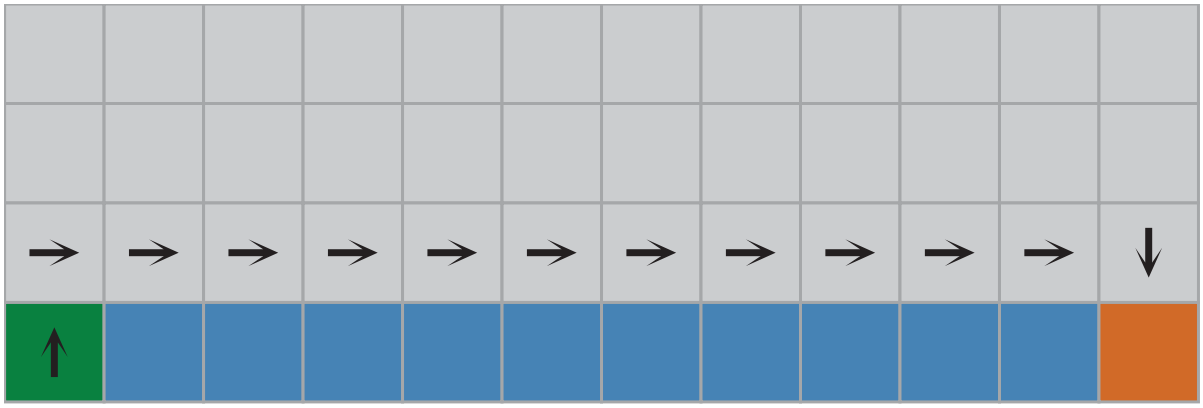
Залежність кількості кроків агента від епізодів



Одержані значення цінності кожної дії в різних станах

U = -6.84 B = -6.84 L = -6.84 R = -6.81	U = -6.63 B = -6.67 L = -6.63 R = -6.64	U = -6.44 B = -6.39 L = -6.39 R = -6.39	U = -6.10 B = -6.12 L = -6.12 R = -6.09	U = -5.77 B = -5.75 L = -5.80 R = -5.74	U = -5.43 B = -5.41 L = -5.37 R = -5.36	U = -5.01 B = -4.95 L = -5.10 R = -4.93	U = -4.50 B = -4.47 L = -4.61 R = -4.45	U = -4.04 B = -3.92 L = -4.00 R = -3.93	U = -3.51 B = -3.33 L = -3.38 R = -3.32	U = -2.65 B = -2.64 L = -2.82 R = -2.64	U = -2.22 B = -1.89 L = -2.23 R = -2.02
U = -6.97 B = -6.98 L = -6.97 R = -6.97	U = -6.79 B = -6.79 L = -6.79 R = -6.78	U = -6.52 B = -6.47 L = -6.71 R = -6.46	U = -6.14 B = -6.10 L = -6.31 R = -6.10	U = -5.80 B = -5.68 L = -5.95 R = -5.68	U = -5.38 B = -5.21 L = -5.54 R = -5.21	U = -4.94 B = -4.68 L = -5.25 R = -4.68	U = -4.54 B = -4.09 L = -4.45 R = -4.09	U = -3.90 B = -3.44 L = -3.93 R = -3.44	U = -3.52 B = -2.71 L = -3.76 R = -2.71	U = -2.29 B = -1.90 L = -2.59 R = -1.90	U = -2.30 B = -1.00 L = -1.94 R = -1.56
U = -7.20 B = -7.46 L = -7.17 R = -6.86	U = -7.06 B = -105.7 L = -7.15 R = -6.51	U = -6.77 B = -105.6 L = -6.85 R = -6.13	U = -6.42 B = -105.2 L = -6.50 R = -5.70	U = -6.06 B = -104.1 L = -6.11 R = -5.22	U = -5.66 B = -104.5 L = -5.68 R = -4.69	U = -5.21 B = -103.4 L = -5.20 R = -4.10	U = -4.67 B = -103.3 L = -4.68 R = -3.44	U = -4.07 B = -102.8 L = -4.07 R = -2.71	U = -3.43 B = -100.6 L = -3.43 R = -1.90	U = -2.70 B = -99.75 L = -2.71 R = -1.00	U = -1.88 B = 0.00 L = -1.89 R = -0.99
U = -7.18 B = -7.46 L = -7.45 R = -106.1	U = -6.86 B = -60.16 L = -7.02 R = -75.07	U = -6.50 B = -28.16 L = -28.24 R = -19.44	U = -6.10 B = -19.03 L = -19.09 R = -35.33	U = -5.64 B = -34.89 L = -19.34 R = -27.97	U = -5.20 B = -19.39 L = -27.77 R = -35.40	U = -4.66 B = -53.98 L = -10.00 R = -10.01	U = -4.07 B = -35.10 L = -48.13 R = -19.30	U = -3.43 B = -47.91 L = -35.15 R = -19.16	U = -2.66 B = -27.51 L = -10.00 R = -27.10	U = -0.72 B = -34.39 L = -27.52 R = 0.00	U = 0.00 B = 0.00 L = 0.00 R = 0.00

Оптимальний шлях від початкового стану до мети, що відповідає діям з максимальними цінностями в кожному станові



Список використаної літератури

1. Donald M. Machine learning, neural and statistical classification / M. Donald, D. J. Spiegelhalter, C. C. Taylor. – New York : Ellis Horwood, 1994. – 298 p.

2. Harrington P. Machine Learning in Action / New York : Manning Publications, 2012. – 384 p.

3. Shalev-Shwartz S. Understanding Machine Learning: From Theory to Algorithms / S. Ben-David, S. Shalev-Shwartz. – New York : Cambridge University Press, 2014. – 449 p.

4. Burkov A. The hundred-page machine learning book / Canada : Quebec City, 2019 – 100 p.

5. Deisenroth M. P. Mathematics for machine learning / M. P. Deisenroth, A. A. Faisal, C. S. Ong. – New York : Cambridge University Press, 2020. – 412 p.

6. Alpaydin E. Introduction to Machine Learning / Cambridge : Massachusetts Institute of Technology, 2010. – 640 p.

7. Hand D. Principles of Data Mining / D. Hand, H. Mannila, P. Smyth. – Cambridge : MIT Press, 2001 – 578 p.

8. Schapire R. E. Boosting: Foundations and Algorithms / R. E. Schapire, Y. Freund. – Cambridge : MIT Press, 2012. – 528 p.

9. Sutton R. S. Reinforcement Learning: An Introduction / Cambridge : MIT Press, 2018. – 352 p.

Електронне навчальне видання

Харченко Василь Олегович

ОСНОВИ МАШИННОГО НАВЧАННЯ

Навчальний посібник

Художнє оформлення обкладинки В. О. Харченко
Редактори: С. М. Симоненко, Н. М. Мажуча, О. В. Федяй
Комп'ютерне верстання В. О. Харченка

Формат 60 × 84/16. Ум. друк. арк. 15,35. Обл.-вид. арк. 14,86.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.