МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Л.М. Олещенко

МАШИННЕ НАВЧАННЯ

Комп’ютерний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів, які навчаються
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення мультимедійних та
інформаційно-пошукових систем»)

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензенти: Чумаченко Сергій Миколайович, д-р техн. наук, проф.
Мошенський Андрій Олександрович, канд. техн. наук, доц.

Відповідальний редактор Легеза Віктор Петрович, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 6 від 24.06.2022 р.)
за поданням Вченої ради факультету прикладної математики
(протокол № 9 від 30.05.2022 р.)

Електронне мережне навчальне видання

Олещенко Любов Михайлівна, канд. техн. наук, доцент

МАШИННЕ НАВЧАННЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

Машинне навчання: комп'ютерний практикум з дисципліни «Машинне навчання» [Електронний ресурс]: навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»)/ Л.М. Олещенко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані. – Київ: КПІ ім. Ігоря Сікорського, 2022. – 92 с.

Навчальний посібник розроблено для ознайомлення студентів з технологіями програмування Python для машинного навчання та вимогами до виконаних комп'ютерних практикумів, зокрема, правилами їх оформлення. Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 Інженерія програмного забезпечення (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем») факультету прикладної математики КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Машинне навчання забезпечує часткову або повну автоматизацію вирішення складних професійних завдань у різних галузях людської діяльності. Машинне навчання має широкий спектр застосування: розпізнавання мови, жестів, рукописного тексту, образів, технічна та медична діагностика, прогнозування часових рядів, біоінформатика, виявлення шахрайства, спаму, категоризація документів, біржовий аналіз, кредитний скоринг, прогнозування, ранжирування в інформаційному пошуку тощо.

Сфера застосувань машинного навчання постійно розширюється. Інформатизація суспільства призводить до накопичення величезних обсягів даних в науці, виробництві, бізнесі, транспорті, охороні здоров'я, які потрібно ефективно та швидко обробляти за допомогою технологій машинного навчання.

Метою комп’ютерного практикуму є формування практичних умінь зі створення моделей машинного навчання на наборах даних за допомогою можливостей мови програмування Python.

Навчальний посібник складається зі вступу та 3 розділів, кожен з яких присвячено окремому комп’ютерному практикуму. Розглянуто задачі кластеризації та класифікації даних, створення прогнозних моделей машинного навчання за допомогою алгоритмів кластеризації, нейронних мереж та дерев рішень. Для кожного комп’ютерного практикуму наведено теоретичну інформацію, інструкції для виконання практикумів, питання для самоперевірки та список рекомендованої літератури.

Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення», освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем» факультету прикладної математики КПІ ім. Ігоря Сікорського.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ 1.
ВИКОРИСТАННЯ АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ K-MEANS ДЛЯ АНАЛІЗУ ДАНИХ У PYTHON

Мета роботи*:* реалізувати алгоритм кластеризації *k*-means та алгоритм *k*-means ++ для ініціалізації центроїдів у Python, побудувати криву ліктя для визначення потрібної кількості кластерів, використовуючи набір даних у форматі csv та вказані програмні інструменти Python.

Теоретичні відомості

Кластеризація – це техніка машинного навчання, яка включає групування точок даних у множини, які називаються **кластерами**. Враховуючи набір точок даних, ми можемо використовувати алгоритм кластеризації, щоб класифікувати кожну точку даних у певну групу. Точки даних, які входять до однієї групи, повинні мати подібні властивості та/або особливості, тоді як точки даних у різних групах повинні мати дуже різні властивості та/або особливості. Кластеризація є методом **неконтрольованого навчання (Unsupervised learning)** і є поширеною технікою для аналізу статистичних даних.

Кластерний аналіз даних використовує два припущення.

Перше припущення – розглядувані ознаки об'єкта допускають бажане розбиття пулу (сукупності) об'єктів на кластери.

Друге припущення – правильність вибору масштабу або одиниць вимірювання ознак.

Застосування кластерного аналізу у загальному вигляді зводиться до:

- вибору вибірки об'єктів для кластеризації;
- визначення множини змінних (атрибутів), за якими будуть оцінюватися об'єкти у вибірці, при необхідності – нормалізація значень змінних;
- обчислення значень міри схожості між об'єктами;
- застосування методу кластерного аналізу для створення груп схожих об'єктів (кластерів) та представлення результатів аналізу.

Після отримання та аналізу результатів можливе коригування обраної метрики і методу кластеризації до отримання оптимального результату.

Для визначення "схожості" об'єктів потрібно скласти **вектор характеристик (атрибутів)** для кожного об'єкта – це набір числових значень, наприклад, зріст, вага людини тощо. Однак існують також алгоритми, що працюють з якісними (категорійними) характеристиками.

Після того, як ми визначили вектор характеристик, можна провести нормалізацію, щоб усі компоненти давали однаковий внесок при розрахунку "відстані". У процесі нормалізації усі значення приводяться до деякого діапазону. Нарешті, для кожної пари об'єктів вимірюється "відстань" між ними – ступінь схожості. Усі методи кластеризації можна умовно поділити на такі категорії: ітеративні (Partitioning methods), ієрархічні (Hierarchical methods), щільнісні (Density-based), графові (Graph based methods) та кластеризація на основі моделі (Model based clustering) (рис.1.1).

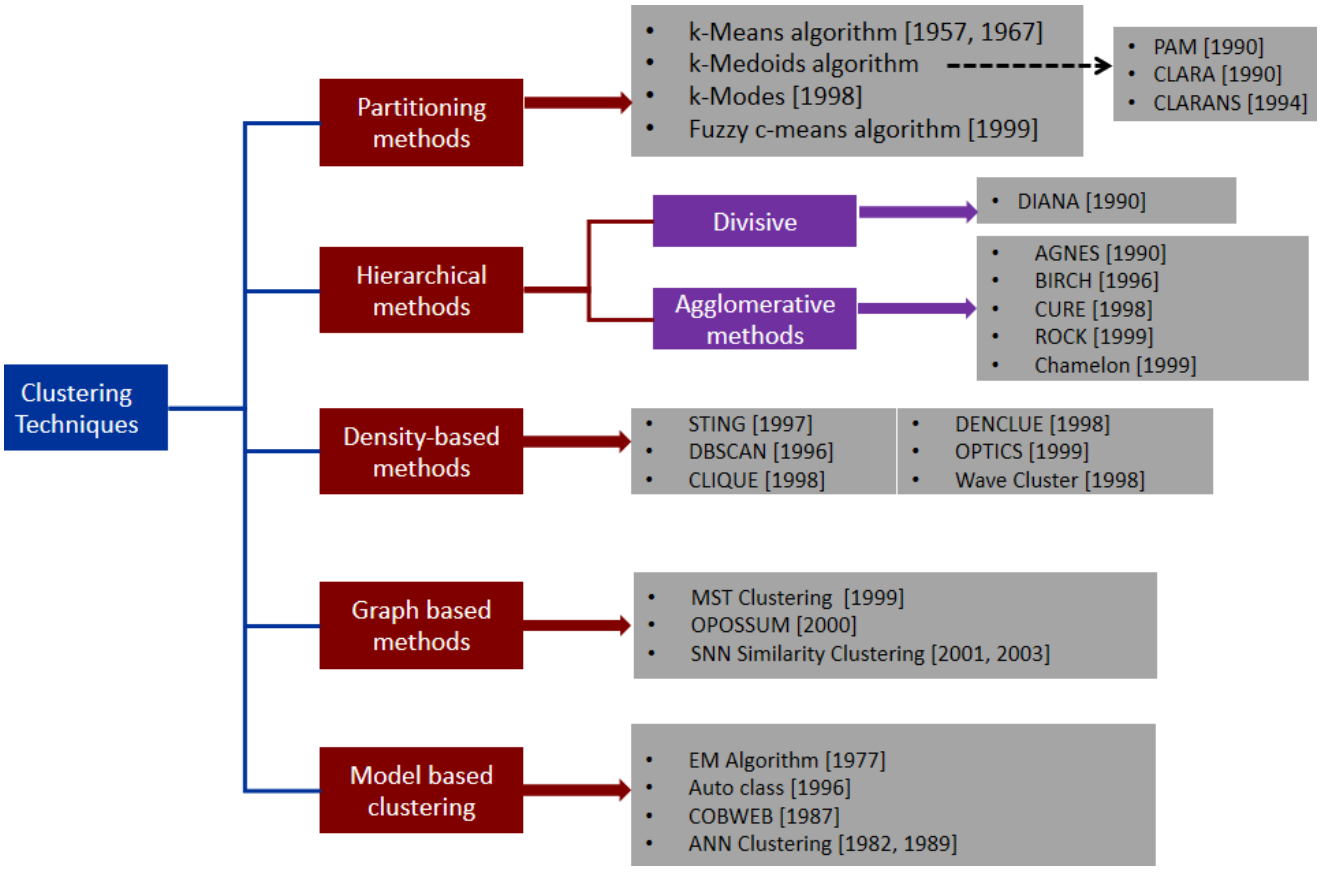


Рис.1.1. Класифікація методів кластеризації [1-2]

Ітеративні методи кластеризації

Ітеративні методи кластеризації виявляють більш високу стійкість по відношенню до шумів і **викидів (outliers)**, некоректного вибору метрики, включенню незначущих змінних у набір, який бере участь в кластеризації. Ціною, яку доводиться "платити" за ці переваги методу, є необхідність визначення аналітиком заздалегідь кількості кластерів, ітерацій або правила зупинки алгоритму, а також деякі інші параметри кластеризації.

Якщо немає припущень щодо числа кластерів, доцільно використовувати ієрархічні алгоритми. Однак, якщо обсяг вибірки не дозволяє це зробити, можливий шлях – проведення ряду експериментів з різною кількістю кластерів, наприклад, почати розбиття сукупності даних з двох груп і, поступово збільшувати їх кількість, порівнювати результати. За рахунок такого "варіювання" результатів досягається досить велика гнучкість кластеризації.

Алгоритм *k*-середніх (*k*-means)

Найбільш поширений серед ітеративних методів кластеризації, також званий швидким кластерним аналізом. На відміну від ієрархічних методів, які не вимагають попередніх припущень щодо числа кластерів, для можливості використання цього методу необхідно мати гіпотезу про найбільш ймовірну кількість кластерів. Алгоритм *k*-means будує *k* кластерів, розташованих на великих відстанях один від одного. Основний тип задач, які вирішує алгоритм *k*-means, – наявність припущень (гіпотез) щодо числа кластерів, при цьому вони повинні бути різними настільки, наскільки це можливо. Вибір числа *k* може базуватися на результатах попередніх досліджень, теоретичних міркуваннях або інтуїції. Метод базується на мінімізації суми квадратів відстаней між кожним спостереженням та центром його кластера. Використовується набір векторів, що належать до *i*-го кластеру та середнє значення цих векторів. Основна ідея полягає у тому, що на кожній ітерації заново обчислюється **центр мас (центроїд)** для кожного кластера, потім вектори розбиваються на нові класи, відповідно до того, який з отриманих центрів виявився ближчим за метрикою.

Переваги алгоритму *k*- means:

– простота та швидкість використання.

Недоліки алгоритму *k*- means:

– алгоритм дуже чутливий до викидів (outliers), які можуть спотворювати середнє. Можливим вирішенням цієї проблеми є використання модифікації алгоритму – алгоритм *k*-медіани;
– алгоритм може повільно працювати на великих наборах даних. Можливим вирішенням цієї проблеми є використання вибірки даних.

Метод найближчого сусіда (kNN-алгоритм)

Метод найближчого сусіда (*k*-nearest neighbors algorithm) є одним із найпростіших алгоритмів кластеризації. Об’єкт *x*, що кластеризується, відноситься до класу y_i, якому належить найближчий об’єкт навчальної вибірки x_i.

Для підвищення надійності класифікації об'єкт відноситься до того класу, якому належить більшість з його сусідів – *k* найближчих до нього об’єктів навчальної вибірки x_i. У задачах з двома класами число сусідів беруть непарним, щоб не виникало ситуацій неоднозначності, коли однакове число сусідів належить різним класам.

У задачах з числом класів 3 і більше непарність вже не допомагає, і ситуації неоднозначності все одно можуть виникати. Тому *i*-му сусіду приписуються ваги ω_i. Об'єкт відноситься до того класу, який набирає найбільші сумарні ваги серед *k* найближчих сусідів.

Якщо міра подібності об'єктів введена досить вдало, то схожі об'єкти частіше лежать в одному класі, ніж в різних. У цьому випадку межа між класами має досить просту форму, а класи утворюють компактно локалізовані області у просторі об'єктів.

Алгоритм EM (expectation-maximization)

EM-алгоритм – алгоритм, що використовується для знаходження оцінок максимальної схожості параметрів ймовірних моделей, у випадку, коли модель залежить від деяких прихованих змінних.

Кожна ітерація алгоритму складається з двох кроків.

На **E-кроці (expectation)** вираховується очікуване значення функції правдоподібності, при цьому приховані змінні розглядаються як спостережувані.

На **M-кроці (maximization)** вираховується оцінка максимальної схожості, таким чином, збільшується очікувана схожість, обчислена на E-кроці. Потім це значення використовується для E-кроку у наступній ітерації. Алгоритм виконується до його збіжності.

Алгоритм EM простий в реалізації, не чутливий до ізольованих об'єктів і швидко сходиться при вдалій ініціалізації. Однак він вимагає для ініціалізації кількості кластерів *k*, що передбачає наявність апріорних знань про дані. Крім того, при невдалій ініціалізації збіжність алгоритму може виявитися повільною або може бути отриманий неякісний результат.

Алгоритм PAM (partitioning around medoids)

PAM є модифікацією алгоритму *k*-means, алгоритмом *k*-медіани (*k*-medoids). Алгоритм менш чутливий до шумів і викидів даних, ніж алгоритм *k*-means, оскільки медіана менше піддається впливам викидів. PAM ефективний для невеликих наборів даних, його не бажано використовувати для великих наборів даних. Більш зрозумілі і прозорі результати кластеризації можуть бути отримані, якщо замість множини вихідних змінних використовувати якісь узагальнені змінні або критерії, які містять в стислому вигляді інформацію про зв'язки між змінними. Тобто виникає задача зниження розмірності даних. Вона може вирішуватися за допомогою різних методів, один з найбільш поширених – факторний аналіз. Факторний аналіз – це сукупність методів, орієнтованих на виявлення та аналіз прихованих залежностей між спостережуваними змінними. Один з методів факторного аналізу – метод головних компонент – заснований на припущенні про незалежність факторів один від одного.

На першому кроці факторного аналізу здійснюється стандартизація значень змінних. Факторний аналіз спирається на гіпотезу про те, що аналізовані змінні є непрямими проявами порівняно невеликого числа прихованих чинників.

Ієрархічні методи кластеризації

Загальна ідея методів даної групи полягає у послідовній ієрархічній декомпозиції множини об'єктів. У випадку агломеративного методу декомпозиції (знизу до верху) кожен об'єкт являє собою самостійний кластер. На кожній ітерації пари прилеглих кластерів послідовно об'єднуються у загальний кластер. Ітерації тривають до тих пір, поки всі об'єкти не будуть об'єднані в один кластер або поки не виконається деяка умова зупинки. Низхідний метод (зверху вниз) ґрунтується на тому, що на початковому етапі усі об'єкти об'єднані в єдиний кластер. На кожній ітерації кластер розділяється на більш дрібні до тих пір, поки кожен об'єкт не виявиться в окремому кластері або не буде виконано умову зупинки. Як умову зупинки можна використовувати граничне число кластерів, яке необхідно отримати, проте зазвичай використовується порогове значення відстані між кластерами.

Основна проблема ієрархічних методів полягає у складності визначення умови зупинки таким чином, щоб виділити "природні" кластери і в той же час не допустити їх розбиття. Ще одна проблема ієрархічних методів кластеризації полягає у виборі точки поділу або злиття кластерів. Цей вибір критичний, оскільки після поділу або злиття кластерів на кожному наступному кроці метод буде оперувати тільки новоствореними кластерами, тому невірний вибір точки злиття або поділу на будь-якому етапі може привести до неякісної кластеризації. Крім того, ієрархічні методи не можуть бути застосовані до великих наборів даних, тому рішення про поділ або злиття кластерів вимагає аналізу великої кількості об'єктів і кластерів, що веде до великої обчислювальної складності методу.

Алгоритм BIRCH

Завдяки узагальненому вигляду кластерів, швидкість кластеризації збільшується, алгоритм BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) при цьому володіє великим масштабуванням. У цьому алгоритмі реалізований двоетапний процес кластеризації (формування попереднього набору кластерів та застосування алгоритмів кластеризації).

Алгоритм WaveCluster

На початку роботи алгоритму WaveCluster дані узагальнюються шляхом накладення на простір даних багатовимірної решітки. На подальших кроках алгоритму аналізуються не окремі точки, а узагальнені характеристики точок, які потрапили в одну клітинку решітки. В результаті такого узагальнення необхідна інформація вміщується в оперативній пам'яті.

На наступних кроках для визначення кластерів алгоритм застосовує хвильове перетворення до узагальнених даних. Головні особливості WaveCluster:

- складність реалізації;
- алгоритм може виявляти кластери довільних форм;
- алгоритм не чутливий до шумів;
- алгоритм може бути застосований тільки до даних низької розмірності.

Алгоритм CLARA

Алгоритм CLARA (Clustering LARge Applications) «витягує» множину зразків з бази даних. Кластеризація застосовується до кожного із зразків, на виході алгоритму пропонується найкраща кластеризація. Для великих баз даних цей алгоритм ефективніший, ніж алгоритм PAM.

Ефективність алгоритму залежить від обраного в якості зразка набору даних. Хороша кластеризація на обраному наборі може не дати хорошу кластеризацію на всій множині даних.

Щільнісні методи кластеризації

Дана категорія методів розглядає кластери як ділянки простору даних з високою щільністю об'єктів, які розділені ділянками з низькою щільністю об'єктів. Щільнісні методи часто застосовуються для фільтрації шуму та виявлення кластерів довільної форми.

Алгоритм DBSCAN

Алгоритм DBSCAN (density-based spatial clustering of applications with noise) – один з перших алгоритмів кластеризації щільнісних методів.

В основі цього алгоритму лежить кілька визначень:

1. ε-околицею об'єкта називається околиця радіуса ε деякого об'єкту.
2. Кореневим об'єктом називається об'єкт, ε-околиця якого містить не менше деякого мінімального числа MinPts об'єктів.
3. Об'єкт p безпосередньо щільно-досяжний з об'єктом q, якщо p знаходиться в ε-околиці q і q є кореневим об'єктом.
4. Об'єкт p щільно-досяжний з об'єкта q при заданих ε і MinPts, якщо існує послідовність об'єктів p_1, ..., p_n, де $p_1 = q$ і $p_1 = p$, така що p_{1+n} безпосередньо щільно досяжний з p_i, $1 \leq i \leq n$.
5. Об'єкт p щільно-з'єднаний з об'єктом q при заданих ε і MinPts, якщо існує об'єкт o такий, що p і q щільно-досяжні з o.

Для пошуку кластерів алгоритм DBSCAN перевіряє ε-околицю кожного об'єкта. Якщо ε-околиця об'єкта p містить більше точок ніж MinPts, то створюється новий кластер з кореневим об'єктом p. Потім DBSCAN ітеративно збирає об'єкти безпосередньо щільно-досяжні з кореневих об'єктів, які можуть привести до об'єднання кількох щільно-досяжних кластерів. Процес завершується, коли ні до одного кластеру не може бути додано жодного нового об'єкта. Хоча, на відміну від методів розбиття, DBSCAN не вимагає заздалегідь вказувати число одержуваних кластерів, виникне потреба у вказівках значень параметрів ε і MinPts, які безпосередньо впливають на результат кластеризації. Оптимальні значення цих параметрів складно визначити, особливо для багатовимірних просторів даних.

Алгоритм OPTICS

OPTICS (ordering points to identify the clustering structure) – це алгоритм знаходження щільності на основі кластерів просторових даних. Його основна ідея схожа на DBSCAN, але він вирішує один з основних недоліків DBSCAN – проблему визначення значущих кластерів в наборах даних різної щільності. Для цього об'єкти набору даних повинні бути впорядковані (за лінійний час) так, що об'єкти, які просторово близькі, будуть сусідами в упорядкуванні.

Крім того, особлива відстань зберігається для кожної точки, яка являє собою щільність, яка повинна бути прийнятна для кластера, щоб обидві сусідні точки належали до тієї ж групи.

Властивості кластерів

Розглянемо приклад. Банк хоче надати своїм клієнтам пропозиції щодо надання кредитних карток. В даний час вони розглядають деталі кожного клієнта і на основі цієї інформації вирішують, яку пропозицію якому клієнту слід надати. У банку потенційно можуть бути мільйони клієнтів. Чи є сенс розглядати деталі кожного замовника окремо, а потім приймати рішення? Ручний процес займе величезну кількість часу. Що може зробити банк? Одним із варіантів є сегментація своїх клієнтів на різні групи (кластери). Наприклад, банк може групувати клієнтів на основі їх доходу:

Рис.1.2. Групування клієнтів на основі їх доходу

Банк може зробити три різні стратегії або пропозиції, по одній для кожної групи. Замість того, щоб створювати різні стратегії для окремих споживачів, потрібно скласти лише 3 стратегії. Це зменшить зусилля та час. Ці групи відомі як **кластери**, процес створення цих груп називається **кластеризацією**.

Кластеризація – це процес розподілу даних на групи (кластери) на основі шаблонів даних.

Наприклад, потрібно передбачити продажі (*Item_Outlet_Sales)* на основі даних *outlet_size, outlet_location_type* :

Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
Medium	Tier 1	Supermarket Type1	3735.1380
Medium	Tier 3	Supermarket Type2	443.4228
Medium	Tier 1	Supermarket Type1	2097.2700
NaN	Tier 3	Grocery Store	732.3800
High	Tier 3	Supermarket Type1	994.7052

Або потрібно передбачити, буде затверджена позика клієнту чи ні (*Loan_Status*) залежно від статі, сімейного стану, доходу клієнтів тощо:

Loan_ID	Gender	Married	ApplicantIncome	LoanAmount	Loan_Status
LP001002	Male	No	5849	130.0	Y
LP001003	Male	Yes	4583	128.0	N
LP001005	Male	Yes	3000	66.0	Y
LP001006	Male	Yes	2583	120.0	Y
LP001008	Male	No	6000	141.0	Y

Якщо ми маємо цільову змінну для прогнозування на основі заданого набору предикторів або незалежних змінних, ми використовуємо **контрольоване навчання моделі (Supervised Learning)**.

Також можуть бути ситуації, коли ми *не* маємо жодної цільової змінної, яку потрібно передбачити. Якщо ми маємо лише незалежні змінні і жодної цільової/залежної змінної, у таких задачах використовується **навчання без нагляду (Unsupervised learning).**

У кластеризації ми не маємо цілі прогнозування. Ми переглядаємо дані, а потім намагаємося об’єднати схожі спостереження та сформувати різні групи. Отже, це проблема контролю без нагляду (Unsupervised learning).

Властивості кластерів

Візьмемо той самий банк, який хоче сегментувати своїх клієнтів. Для спрощення, скажімо, банк хоче використовувати лише дохід та борг для сегментації клієнтів (рис.1.3).

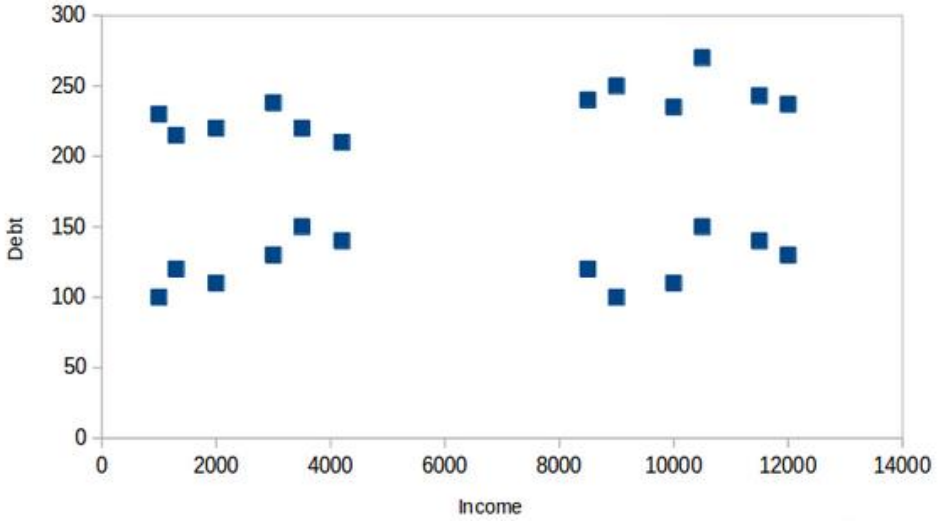


Рис.1.3. Графік розсіювання для візуалізації даних про клієнтів банку (вісь *X* відображає дохід клієнта, а вісь *Y* представляє суму боргу)

Клієнтів банку можна розділити на 4 різні кластери (рис.1.4):

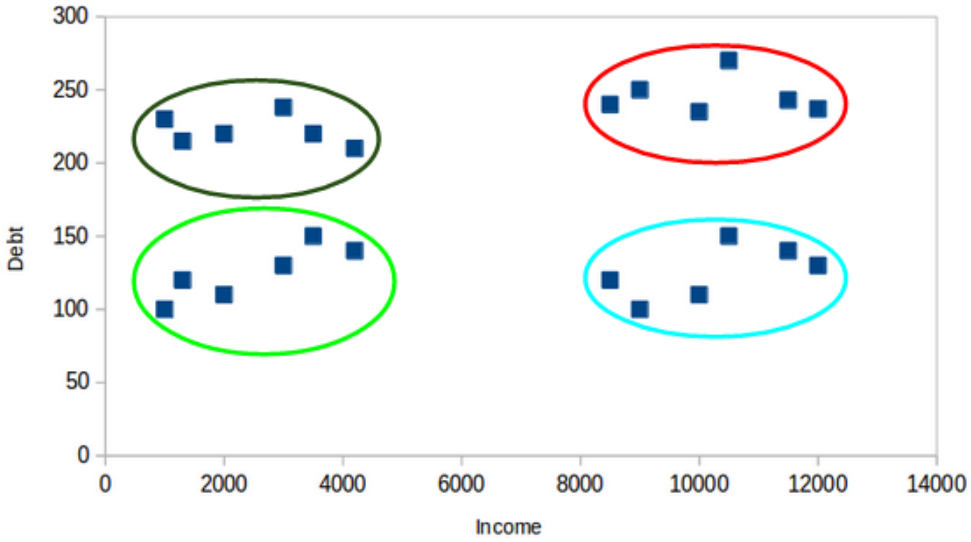


Рис.1.4. Групування клієнтів банку у кластери

Банк може надалі використовувати ці кластери для вироблення стратегій та надання знижок своїм клієнтам. Розглянемо властивості цих кластерів.

Властивість 1

Усі точки даних у кластері повинні бути подібними.

Якщо клієнти певного кластеру не схожі між собою, то їх вимоги можуть відрізнятися. Якщо банк дасть їм таку ж пропозицію, їм може не сподобатися, і їх інтерес до банку може зменшитися. Наявність подібних точок даних у межах одного кластера допомагає банку використовувати цільовий маркетинг.

Властивість 2

Точки даних з різних кластерів повинні бути максимально різними.

Візьмемо той самий приклад, щоб зрозуміти цю властивість:

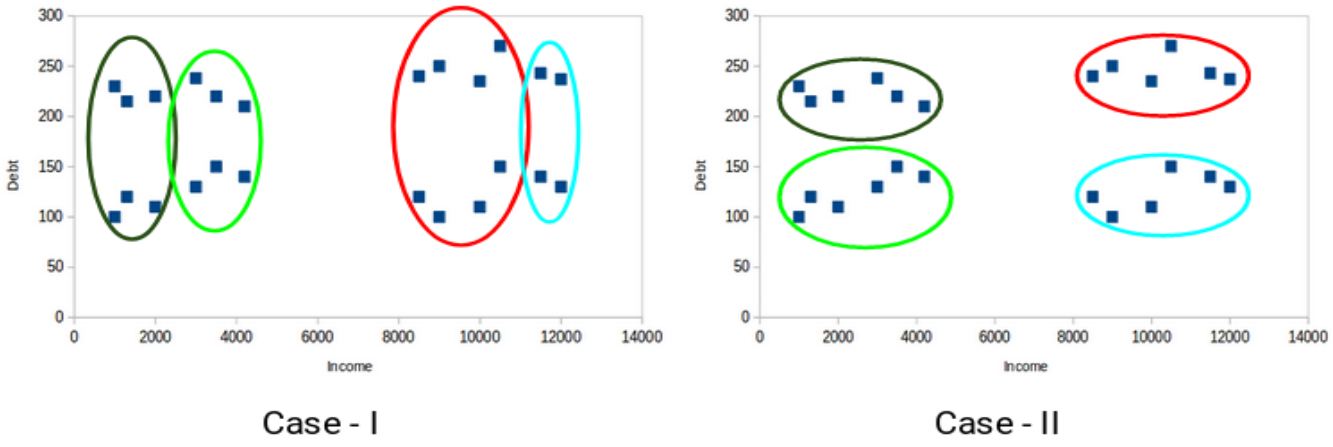


Рис.1.5. Приклади розбиття клієнтів банку на кластери (випадки I і II)

Який із цих випадків дасть кращі кластери? Розглянемо випадок I:

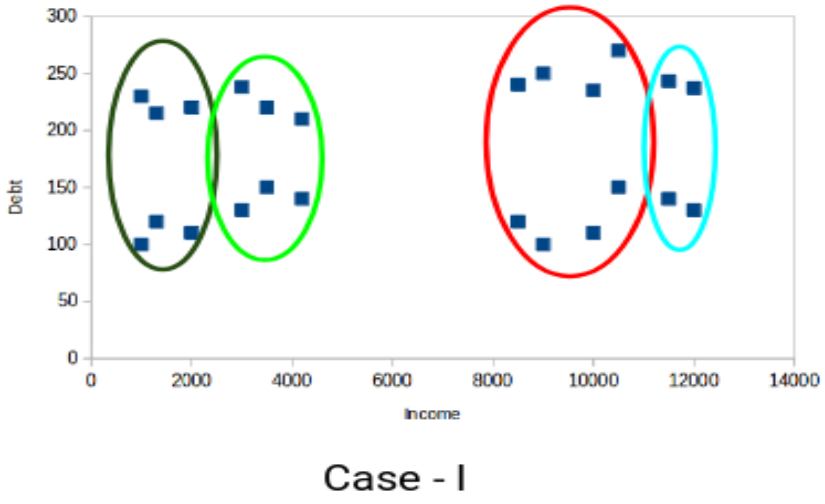


Рис.1.6. Приклад розбиття клієнтів банку на кластери (випадок I)

Клієнти в червоному та синьому кластерих досить схожі між собою. Чотири найкращі точки червоного кластера мають подібні властивості, як і два найкращих клієнта синього кластера. Вони мають високий дохід і велику вартість боргу. Тут ми згрупували їх по-різному. Розглянемо випадок II (рис.1.7).

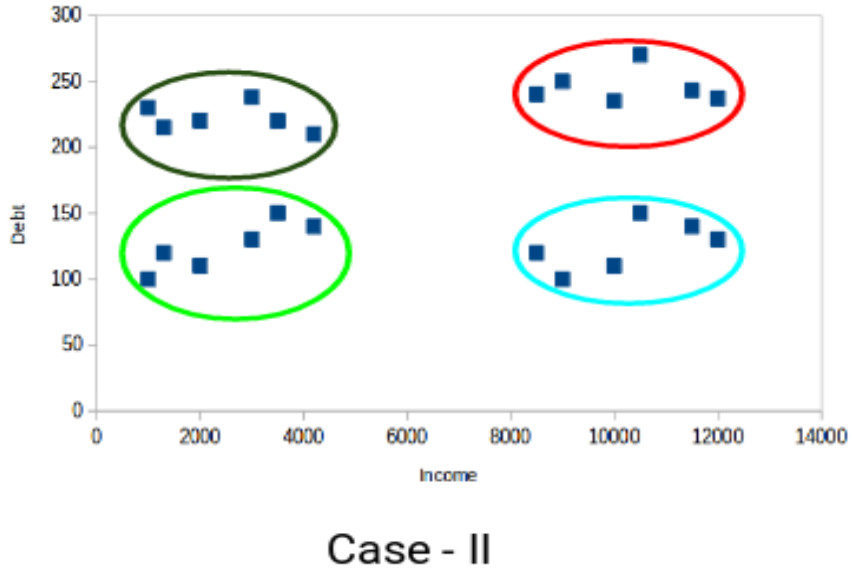


Рис.1.7. Приклад розбиття клієнтів банку на кластери (випадок II)

Точки в червоному кластері повністю відрізняються від клієнтів у синьому кластері. Усі клієнти червоного кластеру мають високий дохід і великий борг, а клієнти синього кластеру мають високий дохід і низьку вартість боргу. Очевидно, що в цьому випадку ми маємо кращу кластеризацію клієнтів. Отже, точки даних з різних кластерів повинні якомога більше відрізнятися один від одного, щоб мати більш значущі кластери.

Застосування кластеризації у реальних сценаріях

Сегментація клієнтів. Одним із найпоширеніших застосувань кластеризації є сегментація клієнтів. І це не лише банківська справа. Ця стратегія включає різні функції, включаючи телекомунікації, електронну комерцію, спорт, рекламу, продажі тощо.

Кластеризація документів. Скажімо, у нас є кілька документів, і нам потрібно згрупувати документи таким чином, щоб подібні документи знаходились в одних і тих же кластерах.

Сегментація зображень. Кластеризацію можна використовувати для виконання сегментації зображень. Тут ми намагаємось об’єднати подібні пікселі на зображенні разом. Ми можемо застосувати кластеризацію для створення кластерів, що мають подібні пікселі в одній групі.

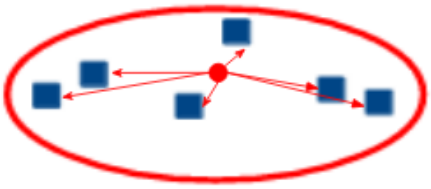
Рекомендаційні двигуни. Кластеризація може бути використана в механізмах рекомендацій. Наприклад, ми можемо рекомендувати пісні своїм друзям переглянувши пісні, які сподобалися окремій людині, а потім скористатися кластеризацією, щоб знайти схожі пісні та порекомендувати найбільш подібні пісні.

Метрики оцінки кластеризації

Візьмемо ще раз приклад сегментації клієнтів – ми маємо дані про дохід клієнта, його професію, стать, вік та багато іншого. Розглянемо деякі метрики оцінки якості наших кластерів.

Інерція

Інерція повідомляє нам, наскільки віддалені точки у кластері. За інерцією обчислюється сума відстаней усіх точок кластера від центроїда цього кластера. Ми обчислюємо це для всіх кластерів, і остаточне інерційне значення – це сума усіх цих відстаней. Ця відстань всередині кластерів відома як **внутрішньокластерна відстань**. Отже, інерція дає нам суму внутрішньокластерних відстаней:



Intra cluster distance

Рис.1.8. Внутрішньокластерна відстань

Ми хочемо, щоб точки в одному кластері були схожі між собою, отже, **відстань між ними повинна бути якомога меншою**.

Чим менше значення інерції, тим кращі кластери.

Індекс Данна

Інерція намагається мінімізувати внутрішньокластерну відстань, зробити більш компактні кластери. Якщо відстань між центроїдом кластера і точками у цьому кластері невелика, точки розташовані ближче одна до одної. Інерція гарантує, що перша властивість кластерів виконана.

Потрібно врахувати і другу властивість – різні кластери повинні якомога більше відрізнятися один від одного.

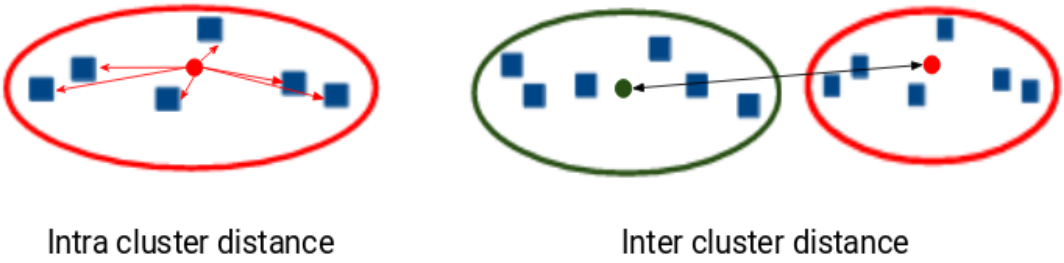


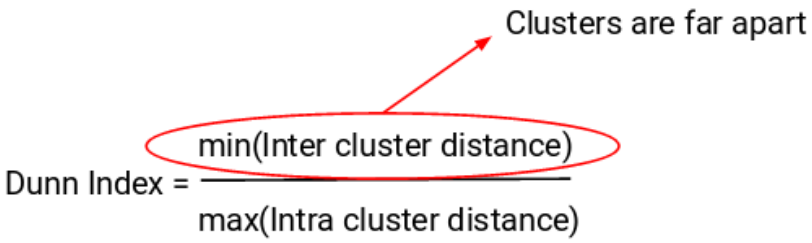
Рис.1.9. Міжкластерна відстань та відстань між центроїдами

Поряд з відстанню між центроїдом і точками, індекс Данна враховує відстань між двома кластерами. Ця відстань між центроїдами двох різних кластерів відома як **міжкластерна відстань**. Формула індексу Данна:

$$\text{Dunn Index} = \frac{\min(\text{Inter cluster distance})}{\max(\text{Intra cluster distance})}$$

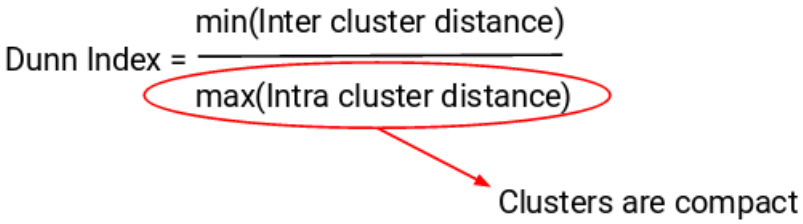
Індекс Данна – це відношення мінімуму міжкластерних відстаней та максимуму внутрішньокластерних відстаней.

Нам потрібно максимізувати індекс Данна. Чим більше значення індексу Данна, тим кращими будуть кластери.



Щоб максимізувати значення індексу Данна, чисельник повинен бути максимальним. Тут ми беремо мінімум відстаней між кластерами.

Отже, відстань навіть між найближчими кластерами повинна бути більшою, щоб переконатися, що кластери знаходяться далеко один від одного.



Знаменник дробу повинен бути мінімальним, щоб максимізувати індекс Данна. Ми беремо максимум внутрішньокластерних відстаней. Максимальна відстань між центроїдами кластера та точками повинна бути мінімальною, що забезпечить компактність кластерів.

Кластеризація *k*-means

Згадаймо першу властивість кластерів – точки всередині кластера повинні бути подібними одна до одної. Отже, **наша мета – мінімізувати відстань між точками всередині кластера.**

Алгоритм *k*-means намагається мінімізувати відстань точок в кластері з їх центроїдом. Це алгоритм, заснований на центроїді, або алгоритм, що базується на відстані, де ми обчислюємо відстані для призначення точки кластеру.

У *k*-means кожен кластер асоціюється з центроїдом.

Основною метою алгоритму *k*-means є мінімізація суми відстаней між точками та їх відповідним центроїдом кожного кластера.

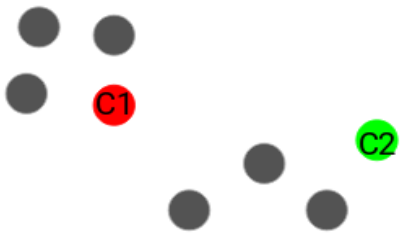
Розглянемо приклад, щоб зрозуміти, як працює *k*-means:

У нас є 8 точок, і ми хочемо застосувати *k*-means для створення кластерів для цих точок.

Крок 1. Вибір кількості кластерів *k*.

Крок 2. Вибір *k* випадкових точок з даних у якості центроїдів.

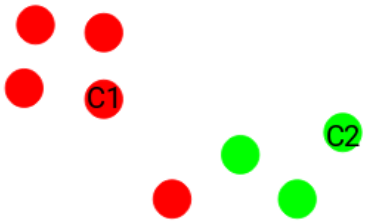
Ми випадково вибираємо центроїд для кожного кластера. Скажімо, ми хочемо мати 2 кластери, тому $k = 2$. Потім ми випадково вибираємо центроїди C1 та C2:



Тут червоні та зелені кола представляють центроїди цих кластерів.

Крок 3. Призначення точок найближчому центроїду кластера.

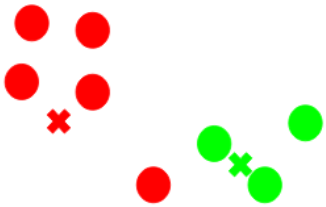
Після того, як ми ініціалізували центроїди, ми призначаємо кожну точку найближчому центроїду кластера:



Точки, які знаходяться ближче до червоної точки, присвоюються червоному скупченню, тоді як точки, які знаходяться ближче до зеленої точки, присвоюються зеленому скупченню.

Крок 4. Обчислення центроїдів новоутворених кластерів.

Тепер, як тільки ми призначили усі точки кожному кластеру, наступним кроком є обчислення центроїдів новоутворених кластерів:



Тут червоний і зелений хрестики – це нові центроїди.

Крок 5. Повторення кроків 3 і 4.

Потім ми повторюємо кроки 3 і 4.

Крок обчислення центроїда і присвоєння всіх точок кластеру на основі їх відстані від центроїда – це одна ітерація. Виникає питання – коли ми повинні зупинити цей процес?

Критерії зупинки кластеризації *k*-means

Існує три критерії зупинки алгоритму *k*-means:

1. Центроїди новоутворених кластерів не змінюються.
2. Точки залишаються у тому ж скупченні.
3. Досягнуто максимальної кількості ітерацій.

Ми можемо зупинити алгоритм, якщо центроїди новоутворених кластерів не змінюються. Навіть після декількох ітерацій, якщо отримано однакові центроїди для усіх кластерів, ми можемо сказати, що алгоритм не вивчає жодного нового шаблону, і це ознака для припинення процесу навчання.

Ще один чіткий знак того, що нам слід зупинити тренувальний процес, якщо точки залишаються в одному кластері навіть після тренування алгоритму для декількох ітерацій. Нарешті, ми можемо припинити навчання, якщо досягнуто максимальної кількості ітерацій. Припустимо, якщо ми встановили кількість ітерацій 100, процес повториться протягом 100 ітерацій перед зупинкою. Існують певні ситуації, коли цей алгоритм може не працювати.

Проблеми з алгоритмом кластеризації *k*-means

Однією із найпоширеніших проблем під час роботи з *k*-means, є те, що розмір кластерів різний (рис. 1.10):

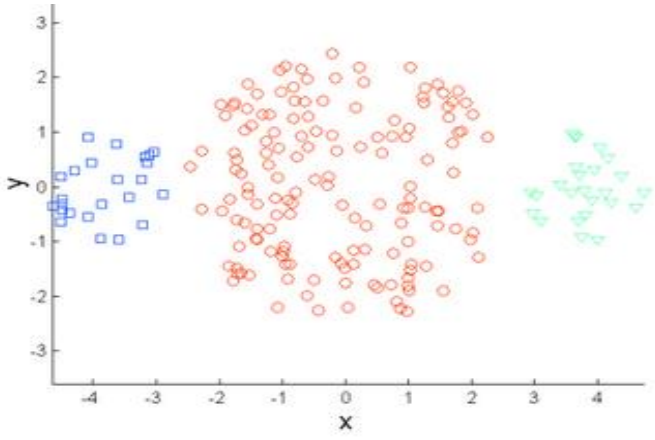


Рис.1.10. Розбиття точок на три кластери різного розміру

Лівий та правий кластери мають менший розмір порівняно з центральним кластером. Якщо ми застосуємо кластеризацію *k*-means до цих точок, результати будуть приблизно такими:

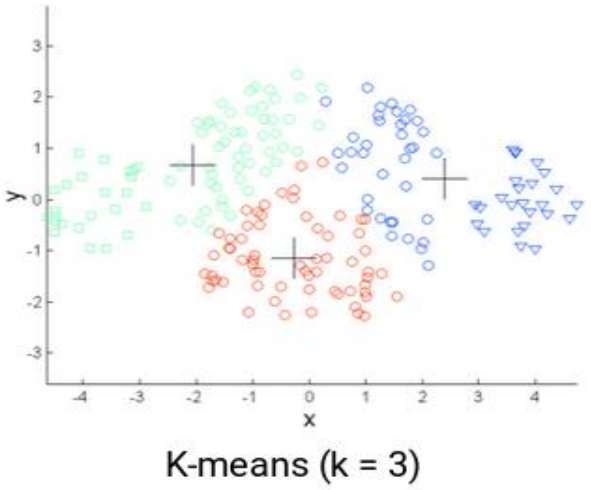


Рис.1.11. Застосування алгоритму *k*-means для кластерів різного розміру

Інша проблема з *k*-means полягає в тому, коли щільність вихідних точок різна (рис. 1.12). Тут точки в червоному скупченні розподілені, тоді як точки в інших кластерах тісно упаковані між собою. Ми бачимо, що компактні точки були призначені одному кластеру. Точки, що розповсюджені вільно, але були в одному кластері, були призначені різним кластерам.

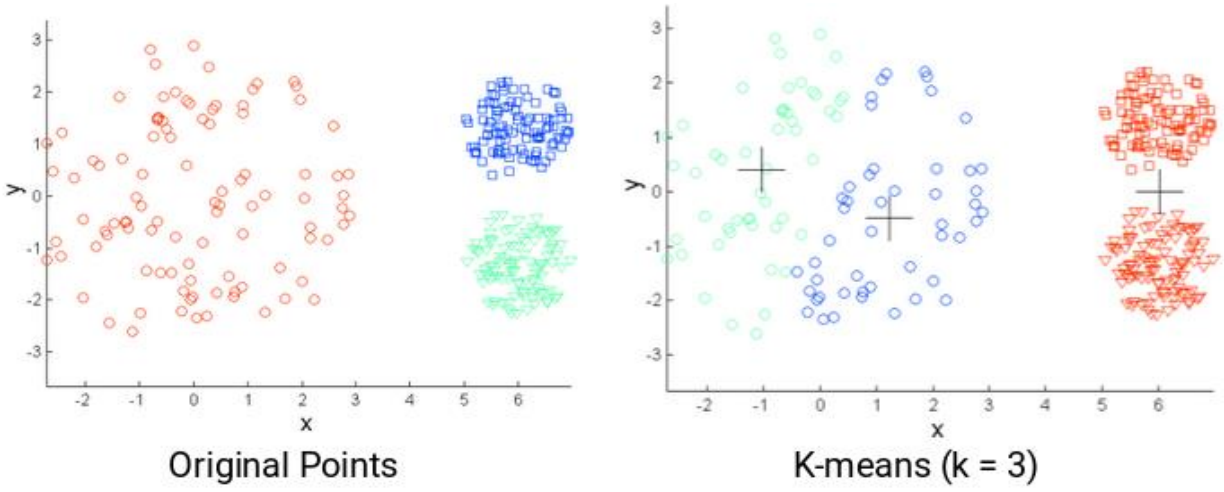


Рис.1.12. Випадок кластеризації, коли щільність вихідних точок різна

Одним із рішень є використання більшої кількості кластерів. У всіх вищезазначених сценаріях замість використання 3 кластерів, ми можемо використовувати більшу їх кількість.

Наприклад, встановлення $k = 10$ може призвести до більш значущих кластерів. Ми випадково ініціалізуємо центроїди. Це також потенційно проблематично, оскільки ми можемо отримувати щоразу різні кластери.

Для вирішення цієї проблеми випадкової ініціалізації існує алгоритм *k*-means++, який можна використовувати для вибору початкових значень або початкових кластерних центроїдів для *k*-means.

Алоритм *k*-means++ для вибору початкових кластерних центроїдів кластеризації *k*-means

У деяких випадках, якщо ініціалізація кластерів не є доречною, *k*-means може призвести до поганої кластеризації точок.

Алгоритм k-means++ визначає процедуру ініціалізації центрів кластера перед тим, як рухатися вперед за допомогою стандартного алгоритму кластеризації *k*-means.

Використовуючи алгоритм *k*-means++, ми оптимізуємо крок, коли ми випадковим чином вибираємо центроїд кластера. Ми, швидше за все, знайдемо рішення, яке буде конкурентоспроможним для оптимального рішення *k*-means, використовуючи ініціалізацію *k*-means++.

Етапи ініціалізації центроїдів за допомогою алгоритму *k*-means++:

1. Перший кластер вибирається навмання з точок даних, які ми хочемо групувати. Це схоже на *k*-means, але замість випадкового вибору всіх центроїдів, ми просто вибираємо один центроїд.

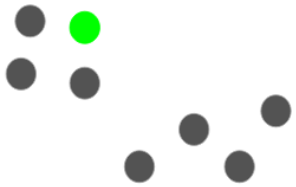
2. Далі ми обчислюємо відстань $D(x)$ кожної точки даних x від центру кластера, який вже був обраний.

3. Потім, вибираємо новий центр кластера з точок даних з ймовірністю x бути пропорційним $D^2(x)$.

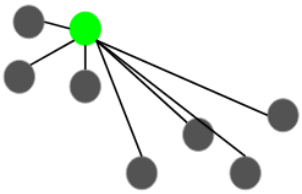
4. Потім ми повторюємо кроки 2 і 3, поки не буде обрано k кластерів.

Наприклад, у нас є такі точки, і ми хочемо створити 3 кластери:

Першим кроком є випадковий вибір точки даних як центроїда кластера:

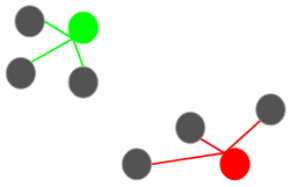


Наприклад, ми вибрали зелену точку як початковий центроїд. Тепер ми обчислимо відстань *D (x)* до кожної точки даних за допомогою цього центроїда:

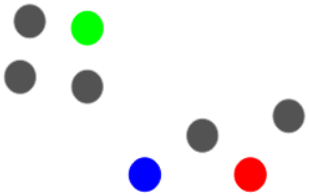


Наступним центроїдом буде той, у якого квадрат відстані $D^2(x)$ є найвіддаленішим від поточного центроїда:

У цьому випадку червоною точкою буде обрано наступний центроїд. Тепер, щоб вибрати останній центроїд, ми візьмемо відстань до кожної точки від найближчого центроїда, і точка, що має найбільший квадрат відстані, буде вибрана як наступний центроїд:



Виберемо останній центроїд:



Ми можемо продовжити алгоритм *k*-means після ініціалізації центроїдів. Використання *k*-means ++ для ініціалізації центроїдів має тенденцію до вдосконалення кластерів. Незважаючи на те, що це обчислювально дорого щодо випадкової ініціалізації, наступні ітерації *k*-means сходяться швидше.

Як правильно вибрати кількість кластерів у кластеризації *k*-means?

Одним з найпоширеніших сумнівів, що виникають під час роботи з *k*-means, є вибір потрібної кількості кластерів. Розглянемо техніку, яка допоможе нам вибрати правильне значення кластерів для алгоритму *k*-means.

Візьмемо вище розглянутий приклад сегментації клієнтів банку на основі їх доходу та суми боргу (рис.1.3). Ми можемо вибрати два кластери для розподілу клієнтів банку (рис.1.13). Усі клієнти з низьким рівнем доходу знаходяться в одному кластері, тоді як клієнти з високим рівнем доходу – у другому.

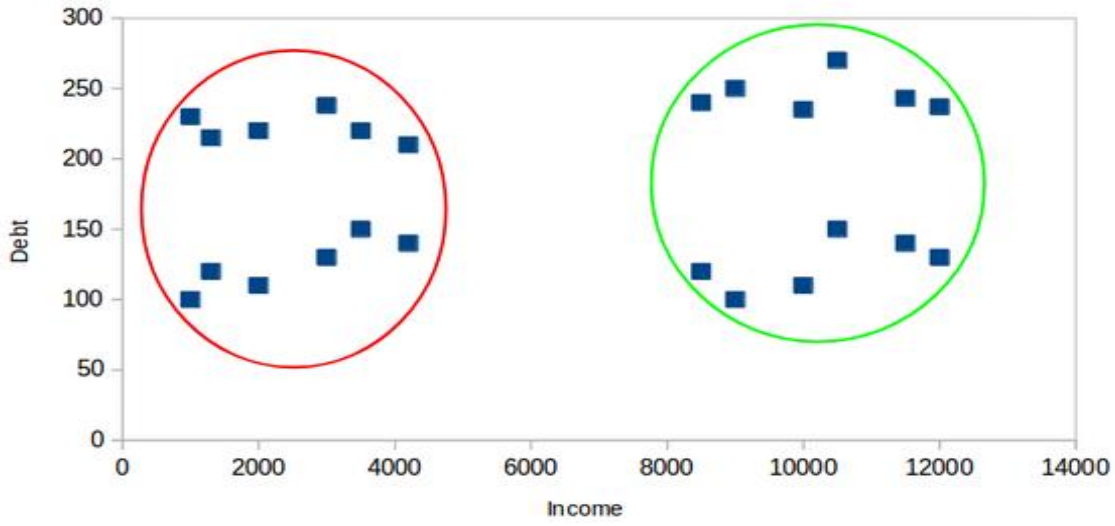


Рис.1.13. Розподіл клієнтів банку на 2 кластери

Ми також можемо розподілити клієнтів банку в 4 кластери:

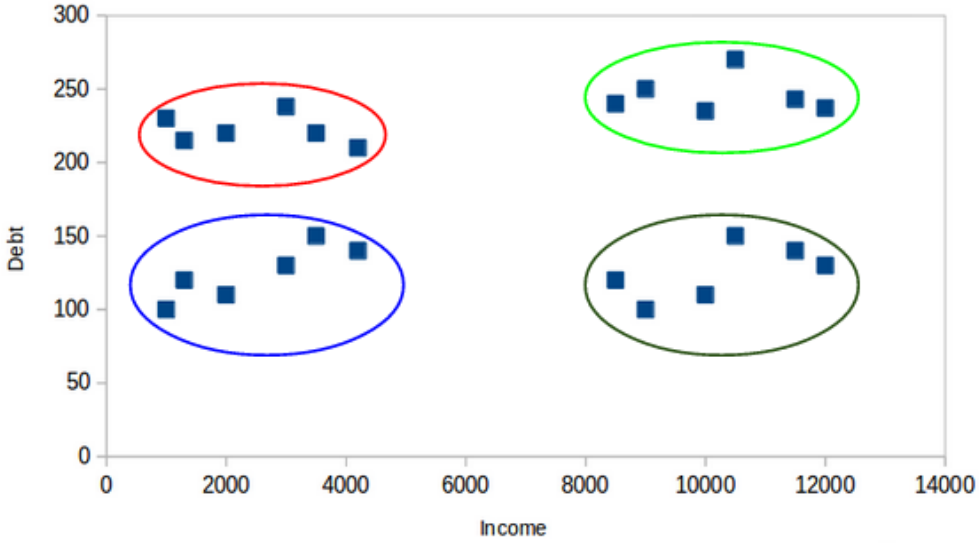


Рис.1.14. Розподіл клієнтів банку на 4 кластери

Тут один кластер може представляти клієнтів з низьким рівнем доходу та низьким боргом, інший кластер – це місце, де клієнти мають високий дохід та великий борг тощо. Також може бути 8 кластерів:

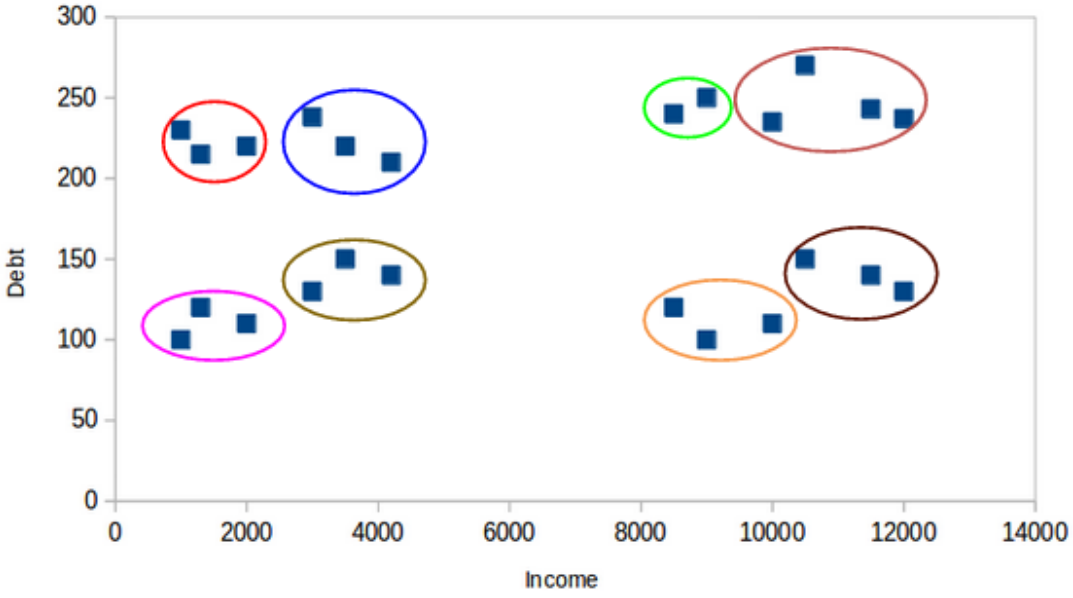


Рис.1.15. Розподіл клієнтів банку на 8 кластерів

Ми можемо обрати будь-яку кількість кластерів. Якою буде максимальна кількість можливих кластерів? Ми можемо призначити кожну точку окремому кластеру. Тобто максимально можлива кількість кластерів буде дорівнює кількості спостережень у наборі даних. У цьому випадку кластеризація не має сенсу. Як визначити оптимальну кількість кластерів?

Для цього потрібно побудувати графік, відомий як **крива ліктя**, де вісь *X* представлятиме кількість кластерів, а вісь *Y* буде метрикою оцінки. Скажімо, це інерція. Також можна обрати будь-який інший показник оцінки, такий як індекс Данна. Далі ми почнемо з невеликого значення кластера, скажімо 2. Ми навчаємо модель, використовуючи 2 кластери, обчислюємо інерцію для цієї моделі і побудуємо її на графіку. Наприклад, ми отримали значення інерції близько 1000:

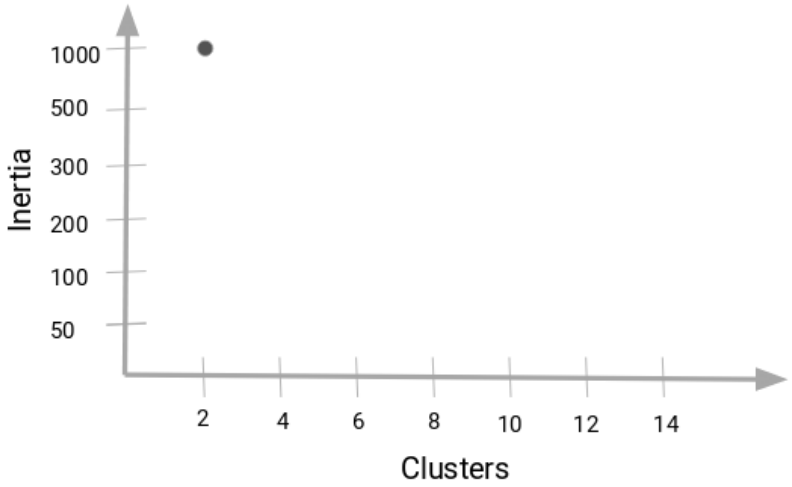


Рис.1.16. Побудова залежності значення інерції від кількості кластерів $k = 2$

Тепер збільшимо кількість кластерів, знову навчимо модель і побудуємо графік значення інерції:

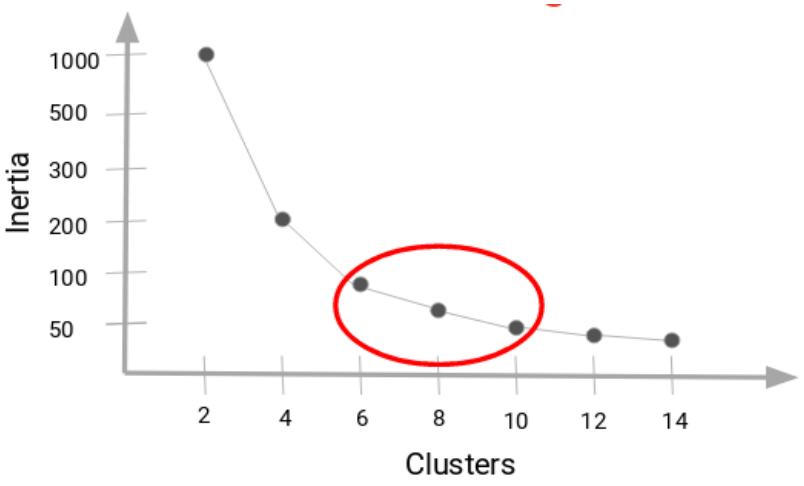


Рис.1.17. Побудова залежності значення інерції від кількості кластерів $k = 2, 4, 6, 8, 10, 12, 14$

Коли ми змінили значення кластера з 2 на 4, значення інерції дуже різко зменшилось. Значення інерції зменшується, і з часом стає постійним, оскільки ми збільшуємо кількість кластерів далі.

Значення кластера, де це зменшення значення інерції стає постійним, потрібно вибрати як оптимальне значення кластера для даних.

Тут ми можемо вибрати будь-яку кількість кластерів між 6 і 10. Ми можемо мати 7, 8 або навіть 9 кластерів. **Потрібно розглянути вартість обчислення, щоб визначити оптимальну кількість кластерів.** Якщо ми збільшимо кількість кластерів, вартість обчислень також зросте. Якщо у нас немає великих обчислювальних ресурсів, потрібно вибирати меншу кількість кластерів.

Для реалізації кластеризації *k*-means у Python використовуються наступні бібліотеки для машинного навчання.

NumPy – це основний пакет для наукових обчислень з Python. Він містить потужний N-вимірний об'єкт масиву та складні (трансляційні) функції.

Pandas – це бібліотека з ліцензією BSD з відкритим кодом, що забезпечує високопродуктивні, прості у використанні структури даних та засоби аналізу даних для мови програмування Python.

Matplotlib – це бібліотека для побудови графіків для мови програмування Python та її числового математичного розширення NumPy.

Завдання до комп’ютерного практикуму 1

У частині 1 потрібно виконати кластеризацію клієнтів банку для надання позики різного типу на основі даних про цих клієнтів, надати відповіді на поставлені питання у ході виконання програмного коду.

У частині 2 потрібно реалізувати алгоритми кластеризації *k*-means та *k*-means++ для ініціалізації центроїдів у Python, а також побудувати криву ліктя, щоб визначити, яка повинна бути потрібна кількість кластерів для вказаного набору даних. Потрібно надати відповіді на поставлені питання у ході виконання програмного коду та пояснити отриманий результат.

Частина 1 комп’ютерного практикуму 1

Потрібно завантажити набір даних про клієнтів банку для прийняття рішення надання відповідної позики даним клієнтам (файл **clustering.csv**).

Імпортуйте необхідні бібліотеки:

```
import pandas as pd
import numpy as np
import random as rd
import matplotlib.pyplot as plt
```

Прочитайте файл CSV і розгляньте перші п’ять рядків даних файлу:

```
data = pd.read_csv('clustering.csv')
data.head()
```

Який результат ви отримали?

Ми будемо брати з даних дві змінні – “LoanAmount” і “AppeantIncome”. Це полегшить візуалізацію кроків. Виберіть ці дві змінні та візуалізуйте точки даних:

```
X = data[["LoanAmount","ApplicantIncome"]]
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```

Який результат ви отримали?

Кроки 1 і 2 *k*-means – виберіть кількість кластерів (*k*=3) і виберіть випадковий центроїд для кожного кластера.

```
K=3
# Select random observation as centroids
Centroids = (X.sample(n=K))
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```

Який результат ви отримали?

Визначте умови реалізації алгоритму кластеризації:

Крок 3 - Призначте всі точки найближчому центроїду кластера.

Крок 4 - Обчисліть центроїди новоутворених кластерів.

Крок 5 - Повторіть кроки 3 і 4.

```
diff = 1
j=0
while(diff!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["ApplicantIncome"]-row_d["ApplicantIncome"])**2
            d2=(row_c["LoanAmount"]-row_d["LoanAmount"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1
    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new =
X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
    if j == 0:
        diff=1
        j=j+1
    else:        diff = (Centroids_new['LoanAmount'] - Centroids['LoanAmount']).sum() +
(Centroids_new['ApplicantIncome'] - Centroids['ApplicantIncome']).sum()
        print(diff.sum())
    Centroids = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
```

Який результат ви отримали?

Отримані значення можуть змінюватися кожного разу, коли ми запускаємо цей код. Ми припиняємо тренування моделі, коли центроїди не змінюються після двох ітерацій. Ми спочатку визначаємо *diff* як 1, так і всередині час циклу, ми обчислення цього *diff* як різницю між центроїдами у попередній та поточній ітерації. Коли ця різниця дорівнює 0, ми припиняємо навчання.

Візуалізуйте кластери та наведіть результат виконання програми.

```
color=['blue','green','cyan']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["ApplicantIncome"],data["LoanAmount"],c=color[k])
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('Income')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```

Поясніть отриманий результат виконання програми.

Частина 2 комп'ютерного практикуму 1

Розглянемо задачу сегментації оптових споживачів. Завантажте набір даних **Wholesale customers data.csv.** Потрібно сегментувати клієнтів оптового дистриб'ютора на основі їх щорічних витрат на різні категорії продуктів, таких як молоко, продукти харчування тощо. Імпортуйте необхідні бібліотеки:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
```

Прочитайте дані про перші п'ять рядків набору даних:

```
data=pd.read_csv("Wholesale customers data.csv")
data.head()
```

Який результат ви отримали?

Ми маємо деталі витрат клієнтів на різні продукти, такі як молоко, бакалія, заморожені продукти, миючі засоби тощо. Тепер потрібно сегментувати клієнтів на основі наданих деталей. Перш ніж це зробити, проаналізуйте статистичні дані:

```
data.describe()
```

Який результат ви отримали?

Змінні, такі як *Channel* та *Region*, мають низьку величину, тоді як такі, як *Fresh, Milk, Grocery* тощо, мають більшу величину.

Оскільки *k*-means – це алгоритм, заснований на відстані, ця різниця вимірності величин може створити проблему. Тому потрібно стандартизувати дані, тобто привести усі змінні до однакової величини:

```
# standardizing the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
    # statistics of scaled data
pd.DataFrame(data_scaled).describe()
```

Який результат ви отримали?

Створіть функцію *kmeans* і помістіть її в дані:

```
# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=2, init='k-means++')
# fitting the k means algorithm on scaled data
kmeans.fit(data_scaled)
```

Оцініть, наскільки добре сформовані кластери. Для цього обчисліть інерцію кластерів:

```
# inertia on the fitted data
kmeans.inertia_
```

Який результат ви отримали?

Спочатку підберіть декілька моделей *k*-means, і потім в кожній наступній моделі збільшіть кількість кластерів. Збережіть значення інерції кожної моделі, а потім побудуйте його для візуалізації результату:

```
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,20):
   kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
   kmeans.fit(data_scaled)
   SSE.append(kmeans.inertia_)
```

```
# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

Який результат ви отримали?

Визначте оптимальне значення кластера за отриманим графіком.

Дивлячись на криву ліктя, виберіть число кластерів (від 5 до 8).

Встановіть кількість кластерів та підберіть модель:

```
# k means using 5 clusters and k-means++ initialization
kmeans = KMeans(n_jobs = -1, n_clusters = 5, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)
```

Визначте кількість точок у кожному із сформованих вище кластерів:

```
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Який результат виконання програми? Скільки точок потрапило до кожного кластеру?Поясніть отриманий результат.

Вимоги до оформлення звіту

Звіт має включати:

1. Титульний аркуш.
2. Завдання на комп’ютерний практикум.
3. Хід роботи. Цей розділ складається з послідовного опису виконуваних кроків згідно інструкцій до комп’ютерного практикуму.
4. Висновки.

Питання для самоперевірки

1. Що таке кластеризація? Які властивості кластерів ви знаєте?
2. Кластеризація відноситься до задач машинного навчання без нагляду *(Unsupervised learning)* чи це контроловане навчання *(Supervised learning)*?
3. Які методи та алгоритми кластеризації ви знаєте?
4. Наведіть приклади ієрархічних, щільнісних та ітеративних методів кластеризації, їх переваги та недоліки.
5. Які застосування кластеризації в реальних сценаріях ви знаєте?
6. Які метрики оцінки кластеризації ви знаєте?
7. Що таке кластеризація K-Means? Як вибрати потрібну кількість кластерів у K-Means?
8. Для чого використовується алгоритм K-Means ++ ?
9. Як реалізувати алгоритм кластеризації K-Means та алгоритм K-Means ++ у Python?
10. Для чого використовується стандартизація даних?
11. Як реалізувати у Python стандартизацію даних?
12. Для чого використовуються пакети Python: pandas, numpy, matplotlib, random?
13. Які методи Python використовуються для прочитання файлу у форматі CSV та перегляду перших п’яти рядків даних?
14. Як реалізувати у Python ініціалізацію центроїдів кластера та та побудувати криву ліктя?

Рекомендована література

1. Clustering Algorithms // Електронний ресурс. Режим доступу: https://developers.google.com/machine-learning/clustering/clustering-algorithms
2. 10 Clustering Algorithms With Python // Електронний ресурс. Режим доступу: https://machinelearningmastery.com/clustering-algorithms-with-python/#:~:text=Cluster%20analysis%2C%20or%20clustering%2C%20is,or%20clusters%20in%20feature%20space

3. Pandas // Електронний ресурс. Режим доступу: https://pandas.pydata.org/
4. NumPy // Електронний ресурс. Режим доступу: https://numpy.org/
5. Matplotlib // Електронний ресурс. Режим доступу: https://matplotlib.org/

КОМП'ЮТЕРНИЙ ПРАКТИКУМ 2.
ПРОГНОЗУВАННЯ НА ОСНОВІ КЛАСИФІКАТОРА ДЕРЕВА РІШЕНЬ

Мета роботи*:* створити класифікатор дерева рішень, який працюватиме з набором даних про пасажирів у форматі csv, використовуючи бібліотеки Python: pandas, sklearn та застосування Graphviz.

Теоретичні відомості

Метод дерев рішень (decision trees) є одним із найбільш популярних методів розв'язання задач класифікації й прогнозування. Якщо залежна, тобто цільова змінна приймає дискретні значення, за допомогою методу дерева рішень розв'язується задача класифікації. Якщо ж залежна змінна приймає неперервні значення, то дерево рішень установлює залежність цієї змінної від незалежних змінних, тобто розв'язує задачу чисельного прогнозування [1].

У найбільш простому вигляді дерево рішень – це спосіб відображення правил у ієрархічній, послідовній структурі. Основа такої структури – відповіді «так» або «ні» на низку питань.

У результаті проходження від кореня дерева (іноді називається кореневою вершиною) до його вершини розв'язується задача класифікації, тобто вибирається один із класів. Метою побудови дерева рішень є визначення значення категоріальної залежної змінної. Основні елементи дерева рішень:

- корінь дерева;
- внутрішній вузол дерева або вузол перевірки;
- листок, кінцевий вузол дерева, вузол розв'язку або вершина.

У вузлах бінарних дерев розгалуження може відбуватися тільки у двох напрямках, тобто існує можливість тільки двох відповідей на поставлене питання («так» і «ні»). Бінарні дерева є найпростішим, частковим випадком дерев рішень. В інших випадках, відповідей і, відповідно, гілок дерева, що виходять із його внутрішнього вузла, може бути більше двох.

Розглянемо приклад. База даних, на основі даних якої повинне здійснюватися прогнозування, містить дані про клієнтів банку з атрибутами:

- вік,
- наявність нерухомості,
- освіта,
- середньомісячний дохід,
- чи повернув клієнт вчасно кредит.

Задача полягає у тому, щоб на підставі перерахованих даних (крім останнього атрибута) визначити, чи варто видавати кредит новому клієнтові. Така задача розв’язується у два етапи:

1. побудова класифікаційної моделі;
2. використання класифікаційної моделі.

На етапі побудови моделі будується дерево класифікації або створюється набір правил.

На етапі використання моделі побудоване дерево, або шлях від його кореня до однієї з вершин, що є набором правил для конкретного клієнта, використовується для відповіді на поставлене питання «Чи надавати кредит?».

Правилом є логічна конструкція, представлена у вигляді «якщо: то:». Задача «Чи надавати кредит клієнтові» є типовою задачею класифікації, і за допомогою дерев рішень одержують досить хороші варіанти її розв’язку.

Внутрішні вузли дерева (вік, наявність нерухомості, дохід і освіта) є атрибутами вище описаної бази даних. Ці атрибути називають прогнозуючими, або **атрибутами розщеплення (splitting attribute)**. Кінцеві вузли дерева, або листки, іменуються **мітками класу**, що є значеннями залежної категоріальної змінної: «видавати» або «не видавати» кредит. Кожна гілка дерева, що йде від

внутрішнього вузла, визначена предикатом розщеплення. Останній може відноситися лише до одного атрибуту розщеплення даного вузла.

Характерна риса предикатів розщеплення: кожний запис використовує унікальний шлях від кореня дерева лише до одного вузла-розв'язку. Об'єднана інформація про атрибути розщеплення й предикати розщеплення у вузлі називається **критерієм розщеплення (splitting criterion).**

Наприклад, критерій розщеплення «Яка освіта?», міг би мати два предикати розщеплення: освіта «вища» і «не вища». Тоді дерево рішень мало б інший вигляд. Отже, для цієї задачі (як і для будь-якої іншої) може бути побудована множина дерев рішень різної якості, з різною прогнозуючою точністю.

Якість побудованого дерева рішень залежить від правильного вибору критерію розщеплення. Метод дерев рішень часто називають «наївним» підходом. Але завдяки певній низці переваг, цей метод є одним із найбільш популярних для розв'язання задач класифікації.

Переваги класифікатора дерев рішень

Інтуїтивність дерев рішень. Класифікаційна модель, представлена у вигляді дерева рішень, є інтуїтивною і спрощує розуміння розв'язуваної задачі. Результат роботи алгоритмів створення дерев рішень, на відміну, наприклад, від нейронних мереж, що представляють собою «чорні ящики», легко інтерпретується користувачем. Ця властивість дерев рішень не тільки важлива при віднесенні до певного класу нового об'єкта, але й корисна при інтерпретації моделі класифікації в цілому. Дерево рішень дозволяє зрозуміти й пояснити, чому конкретний об'єкт відноситься до того або іншого класу.

Робота з базою даних. Дерева рішень дають можливість витягати правила з бази даних, використовуючи прості правила: наприклад, **якщо** Вік >35 і Дохід >200, **то** видати кредит. Розроблений ряд масштабованих алгоритмів, які можуть бути використані для побудови дерев рішень на надвеликих базах даних. Масштабованість тут означає, що із зростанням кількості записів бази даних, час, затрачуваний на навчання, тобто побудову дерев рішень, зростає лінійно. Приклади таких алгоритмів: SLIQ, SPRINT.

Дерева рішень дозволяють створювати класифікаційні моделі у тих галузях, де аналітикові досить складно формалізувати знання. На вхід алгоритму можна подавати всі існуючі атрибути, алгоритм сам вибере найбільш значимі серед них, і тільки вони будуть використані для побудови дерева.

У порівнянні, наприклад, з нейронними мережами, це значно полегшує користувачеві роботу, оскільки в нейронних мережах вибір кількості вхідних атрибутів суттєво впливає на час навчання.

Швидкий процес навчання. На побудову класифікаційних моделей за допомогою алгоритмів класифікатора дерев рішень потрібно значно менше часу, ніж, наприклад, на навчання нейронних мереж. Більшість алгоритмів створення дерев рішень мають можливість спеціальної обробки пропущених значень. Багато класичних статистичних методів, за допомогою яких розв'язуються задачі класифікації, можуть працювати лише із числовими даними, у той час як дерева рішень працюють і з числовими, і з категоріальними типами даних.

Багато статистичних методів є параметричними, і користувач повинен заздалегідь володіти певною інформацією, наприклад, знати тип моделі, мати гіпотезу про тип залежності між змінними, припускати, який вид розподілу мають дані. Дерева рішень, на відміну від таких методів, будують непараметричні моделі.

Отже, дерева рішень здатні розв'язувати такі задачі, у яких відсутня апріорна інформація про вид залежності між досліджуваними даними.

Створення дерева рішень

Розглянута нами задача класифікації відноситься до **контрольованого навчання (Supervised Learning)**, оскільки усі об'єкти тренувального набору даних заздалегідь віднесені до одного з визначених класів. Алгоритм створення дерева рішень складається з етапів побудови або **створення дерева (tree building)** і **скорочення дерева (tree pruning)**. У ході створення дерева вирішуються питання вибору критерію розщеплення й зупинки навчання (якщо це передбачено алгоритмом). У ході етапу скорочення дерева вирішується питання відсікання деяких його гілок.

Критерій розщеплення. Процес створення дерева відбувається зверху вниз, тобто є спадним. У ході процесу алгоритм повинен знайти такий критерій розщеплення (критерій розбивки), щоб розбити множину на підмножини, які б асоціювалися з даним вузлом перевірки. Кожний вузол перевірки повинен бути позначений певним атрибутом. Існує правило вибору атрибута: він повинен розбивати вихідну множину даних таким чином, щоб об'єкти підмножин, що одержуються у результаті цієї розбивки, були представниками одного класу або були максимально наближені до такої розбивки. Тобто кількість об'єктів з інших класів, так званих «домішок», у кожному класі зводиться до мінімуму.

Існують різні критерії розщеплення. Найбільш відомі – **міра ентропії та індекс Gini**. У деяких методах для вибору атрибута розщеплення використовується так звана міра інформативності підпросторів атрибутів, яка ґрунтується на ентропійному підході й відома як **«вимірювання інформаційного виграшу» (information gain measure)** або **вимірювання ентропії**. Інший критерій розщеплення реалізований в алгоритмі CART (Classification And Regression Trees) і називається **індексом Gini**. За допомогою цього індексу атрибут вибирається на підставі відстаней між розподілами класів. Якщо дана множина *T*, що включає зразки з *n* класів, індекс Gini, тобто *gini (T)*, визначається за формулою:

$$gini\ (T) = 1 - \sum_{j=1}^{n} p_j^2 \quad , \tag{2.1}$$

де *T* – поточний вузол, p_j – ймовірність класу *j* у вузлі *T*, *n* – кількість класів.

Кожний вузол бінарного дерева при розбивці має тільки двох нащадків, що називаються **дочірніми гілками**. Подальший поділ гілок залежить від того, чи багато вихідних даних описує дана гілка. На кожному кроці побудови дерева правило, сформоване у вузлі, ділить задану множину зразків на дві частини. Права його частина (гілка right) – це та частина множини, у якій правило виконується; ліва (гілка left) – та, для якої правило не виконується.

Функція оцінки якості розбивки, яка використовується для вибору оптимального правила, – індекс Gini. Дана оціночна функція заснована на ідеї зменшення рівня невизначеності у вузлі.

Допустимо, є вузол, і він розбитий на два класи. Максимальна невизначеність у вузлі буде досягнута при розбивці його на дві підмножини по 50 прикладів, а максимальна визначеність – при розбивці на 100 і 0 прикладів.

Велике дерево не означає, що воно «вдале». Чим більше окремих випадків описано у дереві рішень, тим менша кількість об’єктів потрапляє в кожний окремий випадок. Такі дерева називають «гіллястими» або «кущистими», вони складаються з невиправдано великої кількості вузлів і гілок, вихідна множина розбивається на велику кількість підмножин, що складаються із дуже малого числа об’єктів. У результаті «переповнення» таких дерев їх здатність до узагальнення зменшується, і побудовані моделі не можуть давати вірні відповіді. У процесі побудови дерева, щоб його розміри не стали надмірно великими, використовують спеціальні процедури, які дозволяють створювати оптимальні дерева, так звані дерева «вдалих розмірів».

Який розмір дерева може вважатися оптимальним? Дерево повинно ураховувати інформацію з досліджуваного набору даних, але одночасно воно повинно бути досить простим. Інакше кажучи, дерево повинно використовувати інформацію, що поліпшує якість моделі та ігнорувати ту інформацію, яка її не поліпшує. Тут існує дві можливі стратегії.

Перша полягає в нарощуванні дерева до певного розміру відповідно до параметрів, заданих користувачем. Визначення цих параметрів може ґрунтуватися на досвіді й інтуїції аналітика, а також на деяких «діагностичних повідомленнях» системи, що конструюють дерево рішень.

Друга стратегія полягає у використанні набору процедур, що визначають «вдалий розмір» дерева.

Процедури, які використовують для запобігання створення надмірно великих дерев, включають: скорочення дерева шляхом відсікання гілок; використання правил зупинки навчання. Не усі алгоритми при конструюванні дерева працюють за однією схемою. Деякі алгоритми включають два окремі послідовні етапи: побудова дерева і його скорочення; інші чергують ці етапи в процесі своєї роботи для запобігання нарощування внутрішніх вузлів.

Зупинка побудови дерева рішень

Правило зупинки повинне визначити, чи є розглянутий вузол внутрішнім вузлом (при цьому він буде розбиватися далі) або ж він є кінцевим вузлом, тобто вузлом розв'язком.

Зупинка – момент у процесі побудови дерева, коли слід припинити подальші розгалуження. Один із варіантів правил зупинки – «рання зупинка» (prepruning), вона визначає доцільність розбивки вузла. Перевага використання такого варіанта – зменшення часу на навчання моделі. Однак тут виникає ризик зниження точності класифікації. Тому рекомендується замість зупинки використовувати відсікання. Другий варіант зупинки навчання – обмеження глибини дерева. У цьому випадку побудова закінчується, якщо досягнута задана глибина. Ще один варіант зупинки – задання мінімальної кількості зразків, які будуть утримуватися в кінцевих вузлах дерева. При цьому варіанті розгалуження тривають до того моменту, поки всі кінцеві вузли дерева будуть містити не більш ніж задане число об'єктів.

Скорочення дерева або відсікання гілок. Вирішенням проблеми занадто гіллястого дерева є його скорочення шляхом відсікання (pruning) деяких гілок.

Якість класифікаційної моделі, побудованої за допомогою дерева рішень, характеризується двома основними ознаками: точністю розпізнавання й помилкою.

Точність розпізнавання розраховується як відношення об'єктів, правильно класифікованих у процесі навчання, до загальної кількості об'єктів набору даних, які брали участь у навчанні.

Помилка розраховується як відношення об'єктів, неправильно класифікованих у процесі навчання, до загальної кількості об'єктів набору даних, які брали участь у навчанні.

Відсікання гілок або заміну деяких гілок піддеревом слід проводити там, де ця процедура не призводить до зростання помилки. Процес проходить знизу вгору, тобто є висхідним. Це більш популярна процедура, ніж використання правил зупинки. Дерева, одержувані після відсікання деяких гілок, називають

усіченими. Якщо таке усічене дерево усе ще не є інтуїтивним і складне для розуміння, використовують витяг правил, які поєднують у набори для опису класів. Кожний шлях від кореня дерева до його вершини або листка дає одне правило. Умовами правила є перевірки на внутрішніх вузлах дерева.

Дерево рішень у графічному вигляді – це блок-схема, яка може допомогти прийняти рішення на основі попереднього досвіду. У наступному прикладі у нас є статистичні дані про коміків, їх рейтинг, національність, вік та досвід, а також рішення глядачів про бажання відвідати шоу відповідних коміків. Задача класифікації полягає у передбаченні бажання відвідати шоу глядачами (змінна *'Go'*), використовуючи попередній досвід на основі виділених атрибутів.

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

Далі наведено приклад програмної реалізації класифікатора дерева рішень для даної задачі з використанням мови програмування Python. В якості атрибутів (незалежних змінних) виступають змінні *'Age'*, *'Experience'*, *'Rank'*, *'Nationality'*, в якості залежної змінної виступає колонка *'Go'*.

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
df = pandas.read_csv("shows.csv")
```

```
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
features = ['Age', 'Experience', 'Rank', 'Nationality']
X = df[features]
y = df['Go']
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)
print(dtree.predict([[40, 10, 6, 1]]))
print("[1] means 'GO'")
print("[0] means 'NO'")
```

Для даного набору атрибутів [40, 10, 6, 1] ми отримаємо результат [0] («ні»).

Якщо змінити значення атрибуту *'Rank'* на 7, [40, 10, 7, 1] ми отримаємо результат [1] («піду»). Дерево рішень для даної задачі зображено на рис. 2.1.

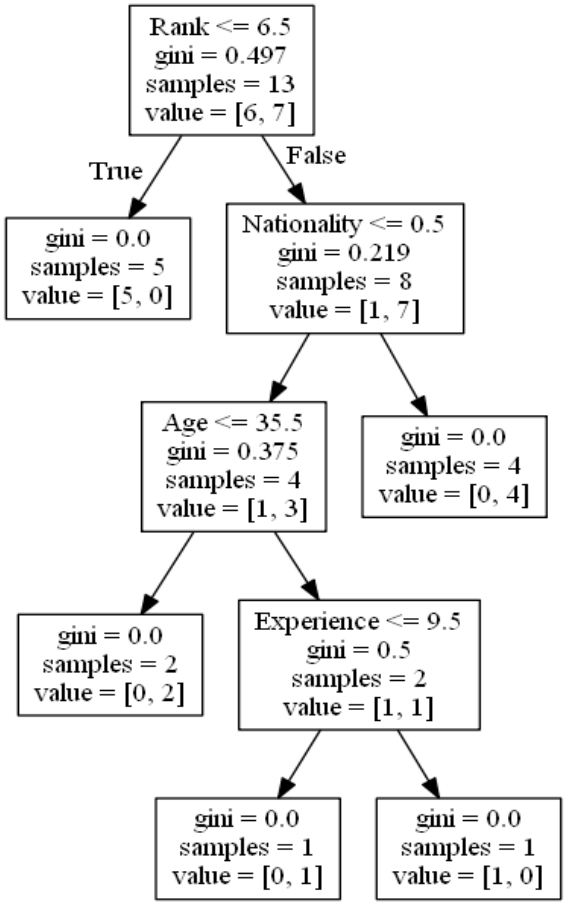


Рис. 2.1. Приклад побудови простого класифікатора дерева рішень [2]

Завдання до комп’ютерного практикуму 2

Потрібно створити класифікатор дерева рішень, який працюватиме з набором даних про пасажирів, які були на борту пасажирського лайнера «Титанік» під час його сумнозвісного рейсу.

У частині 1 потрібно створити класифікатор дерева рішень з використанням мови програмування Python. У частині 2 потрібно застосувати та оцінити модель дерева рішень, використовуючи відповідні технології Python. Потрібно надати відповіді на поставлені питання у ході виконання програмного коду та пояснити отриманий результат.

Частина 1 комп’ютерного практикуму 2

Створення класифікатора дерева рішень

Потрібно створити класифікатор дерева рішень, який буде навчатися на наборі даних із заданими мітками. Набір даних містить імена та дані кожного пасажира пасажирського лайнера «Титанік». Крім того, вказано деталі подорожі пасажирів. На основі цих даних потрібно побудувати дерево рішень, яке ілюструє фактори, які сприяли виживаності або смертності під час подорожі.

Таблиця 2.1. Змінні наборів даних про пасажирів та їх опис

Змінна	Опис
1. PassengerID	Унікальний ідентифікатор для кожного пасажира
2. Survival	Чи пасажир вижив? (0 = ні; 1 = так)
3. Pclass	Клас пасажирських квитків (1 = 1-й; 2 = 2-й; 3 = 3-й)
4. Name	Ім'я пасажира (прізвище та ім'я)
5. Gender	Чоловік чи жінка
6. Age	Вік у роках. Цілі числа зі значеннями float для дітей до 1 року.
7. SibSp	Кількість братів і сестер або подружжя на борту.
8. Parch	Кількість батьків або дітей на борту.
9. Ticket	Номер квитка.
10. Fare	Сума, сплачена за проїзд у британських фунтах.
11. Cabin	Номер кабіни.
12. Embarked	Порт посадки (C = Cherbourg; Q = Queenstown; S = Southampton)

За наведеними вище даними сформулюйте питання про фактори (атрибути), які сприяли тому, що пасажири вижили або загинули під час катастрофи Титаніка.

Крок 1. Створіть фрейм даних.

а) Імпортуйте pandas та файл csv.

Спочатку імпортуйте pandas і створіть фрейм даних з набору навчальних даних Titanic, який зберігається у файлі **titanic-train.csv**. Використовуйте метод **pd.read _csv()**.

```
#Code cell 1
#import pandas
import pandas as pd
#create a pandas dataframe called "training" from the titanic-train.csv file
training = pd.read_csv("./Data/titanic-train.csv")
```

б) Перевірте імпорт і прогляньте дані.

```
#Code cell 2
#verify the contents of the training dataframe using the pandas info() method.
#training.?
```

Чи відсутні значення у наборі даних? Скільки таких відсутніх значень?

```
#Code cell 3
#view the first few rows of the data
#
```

Крок 2. Підготуйте дані для побудови моделі дерева рішень.

а) Замініть рядкові дані числовими мітками.

Для створення дерев рішень потрібно використати бібліотеку scikit-learn. Модель дерева рішень, яку ми будемо використовувати, може обробляти лише числові дані. Значення змінної статі *Gender* необхідно перетворити в числові представлення: 0 буде використовуватися для позначення «чоловіків», а 1 – «жінок».

У програмному коді потрібно використати лямбда-вираз з методом фрейму даних **apply()**. Лямбда-вираз представляє функцію, яка використовує умовний оператор для заміни текстових значень у стовпцях відповідним

числовим значенням. Оператор лямбда можна інтерпретувати так: «якщо параметр toLabel дорівнює «male», повертається 0, якщо значення є чимось іншим, повертається 1». Метод **apply()** виконає цю функцію для значень у кожному рядку стовпця «Gender» фрейму даних.

```
#code cell 4
training["Gender"] = training["Gender"].apply(lambda toLabel: 0 if toLabel == 'male' else 1)
```

б) Переконайтеся, що змінна *Gender* була змінена.

Вихідні дані повинні показувати значення 0 або 1 для змінної *Gender* у наборі даних.

```
#code cell 5
#view the first few rows of the data again
```

в) Зверніться до відсутніх значень у наборі даних.

Результати методу **info()** вказують, скільки спостережень не мають значення віку.

Наведіть число відсутніх значень для атрибуту віку.

Для нашого аналізу значення віку є важливим. Ми повинні обробити ці відсутні значення. Ми можемо замінити ці відсутні значення віку середнім значенням віку для усього набору даних. Це робиться за допомогою методу **fillna()** у стовпці «Age» у наборі даних. Метод **fillna()** змінить вихідний фрейм даних за допомогою аргументу inplace = True .

```
#code cell 6
training["Age"].fillna(training["Age"].mean(), inplace=True)
```

г) Переконайтеся, що значення були замінені.

```
#code cell 7
#verify that the missing values for the age variable have been eliminated.
```

Яке значення було використано для заміни відсутніх значень віку?

```
#use code to answer the question above
```

Крок 3. Навчіть і оцініть модель класифікатора дерева рішень.

а) Створіть об'єкт масиву зі змінною, яка буде цільовою для моделі.

Метою моделі є класифікація пасажирів як здатних та не здатних вижити. Набір даних ідентифікує загиблих та вижившиx пасажирів. Модель класифікатора дерева рішень дізнається, які значення вхідних змінних, швидше за все, належать загиблим та вижившим пасажрам, а потім використає цю інформацію для класифікації пасажирів з унікального набору тестових даних.

```
#code cell 8
#create the array for the target values
y_target = training["Survived"].values
```

б) Створіть масив значень, які будуть вхідними для моделі.

Лише деякі функції даних корисні для створення дерева класифікатора. Ми створюємо список стовпців із даних, які ми хочемо, щоб класифікатор використовував як вхідні змінні, а потім створюємо масив, використовуючи назву стовпця з цієї змінної. Змінна *X_input* містить значення для всіх ознак, які модель використовуватиме, щоб дізнатися, як виконувати класифікацію. Після навчання моделі потрібно використати цю змінну, щоб призначити мітки набору тестових даних.

```
#code cell 9
columns = ["Fare", "Pclass", "Gender", "Age", "SibSp"]
#create the variable to hold the features that the classifier will use
X_input = training[list(columns)].values
```

в) Створіть модель класифікатора дерева рішень.

Імпортуйте модуль дерева рішень з бібліотеки машинного навчання sklearn. Створіть об'єкт класифікатора clf_train. Потім скористайтеся методом **fit()** об’єкта класифікатора зі змінними X_input та y_target як параметрами, щоб навчити модель.

```
#code cell 10
#import the tree module from the sklearn library
from sklearn import tree
#create clf_train as a decision tree classifier object
clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
#train the model using the fit() method of the decision tree object.
#Supply the method with the input variable X_input and the target variable y_tar
get
clf_train = clf_train.fit(X_input, y_target)
```

г) Оцініть модель.

Використовуйте метод **score()** об'єкта дерева рішень, щоб відобразити відсоткову точність призначень, зроблених класифікатором. Він приймає вхідні та цільові змінні як аргументи.

```
#code cell 11
clf_train.score(X_input,y_target)
```

Це значення оцінки вказує на те, що класифікації, зроблені моделлю, повинні бути правильними приблизно в 82% випадків.

Крок 4. Візуалізуйте дерево рішень.

а) Створіть вихідний проміжний файл.

Імпортуйте модуль sklearn.externals.six StringIO, який використовується для виведення характеристик дерева рішень у файл.

Створіть файл Graphviz, який дозволить експортувати результати класифікатора у формат, який можна перетворити на графіку.

```
#code cell 12
from sklearn.externals.six import StringIO
with open("./Data/titanic.dot", 'w') as f:
  f = tree.export_graphviz(clf_train, out_file=f, feature_names=columns)
```

б) Встановіть Graphviz.

Щоб візуалізувати дерево рішень, Graphviz необхідно встановити з терміналу. Інсталяція вимагає відповіді на підказку, що не можна зробити з коду блокнота. Використовуйте команду apt-get install graphviz з командного рядка терміналу, щоб інсталювати це програмне забезпечення.

в) Перетворіть проміжний файл у графічний.

Створений вище файл точок можна перетворити на файл формату .png за допомогою засобу візуалізації точок graphiz. Це команда оболонки, тому використовуйте ! перед цим, щоб запустити його з блокнота. Новий графічний файл titanic.png має з'явитися в каталозі, який містить цей блокнот.

```
#code cell 13
#run the Graphviz dot command to convert the .dot file to .png
!dot -Tpng ./Data/titanic.dot -o ./Data/titanic.png
```

г) Покажіть зображення.

Тепер ми імпортуємо модуль Image з бібліотеки IPython.display. Це дозволить нам відкрити та відобразити зовнішній графічний файл на сторінці блокнота. Функція Image використовується для відображення файлу з ім'ям файлу .png як аргументом.

```
#code cell 14
#import the Image module from the Ipython.display libary
from IPython.display import Image

#display the decison tree graphic
Image("./Data/titanic.png")
```

д) Інтерпретуйте дерево рішень.

З дерева рішень ми бачимо кілька речей. По-перше, в корені дерева знаходиться змінна *Gender*, що вказує на те, що вона є єдиним найважливішим фактором у створенні класифікації. Гілки ліворуч призначені для статі = 0 (чоловічої статі). Кожен корінь і проміжний вузол містить коефіцієнт рішення, ентропію та кількість пасажирів, які відповідають критерію у цій точці дерева.

Виходячи з даних кореневого вузла, скільки спостережень складають набір навчальних даних?

Виходячи з даних наступного рівня, скільки пасажирів були чоловіками, а скільки жінками?

На третьому рівні, у крайньому правому куті, ми бачимо, що 415 пасажирів були чоловіками і заплатили за проїзд менше 26,2686. Нарешті, листові вузли для цього проміжного вузла вказують, що 15 з цих пасажирів були молодше 13,5 років, а інші 400 були старшими за цей вік.

Елементи у масиві значень вказують на виживання. Перше значення – це кількість людей, які загинули, а друге – кількість тих, хто вижив за кожним критерієм.

Виходячи з даних кореневого вузла, скільки людей з даної вибірки померли та скільки вижили?

Ентропія є мірою шуму (невизначеності) у рішенні. Наприклад, у вузлах, у яких рішення призводить до однакових значень у масиві значень виживання, ентропія досягає максимально можливого значення, яке становить 1,0. Це

означає, що модель не змогла остаточно прийняти рішення щодо класифікації на основі вхідних змінних. Для значень дуже низької ентропії рішення було набагато більш чітким, а різниця в кількості вцілілих і жертв набагато вища.

Що характеризує групу, яка мала найбільше смертей за кількістю?

У якій групі вижило найбільше пасажирів?

Частина 2 комп'ютерного практикуму 2

Застосування та оцінка моделі класифікатора дерева рішень

У цій частині ми будемо використовувати результати моделі навченого дерева рішень, щоб позначити немаркований набір даних про пасажирів Титаніка. Дерево рішень оцінює особливості кожного спостереження та позначає спостереження, пасажир вижив (мітка = 1) або загинув (мітка = 0).

Крок 5. Імпортуйте та підготуйте дані.

На цьому кроці ви імпортуєте та підготуєте дані для аналізу.

а) Імпортуйте дані.

Назвіть фрейм даних «testing» та імпортуйте файл **titanic-test.csv**.

```
#code cell 15
#import the file into the 'testing' dataframe.
testing = pd.read_csv("./Data/titanic-test.csv")
```

Скільки записів у наборі даних?

Яких важливих змінних значень немає, скільки відсутніх?

б) Використовуйте лямбда-вираз, щоб замінити значення *"male"* та *"female"* на 0 для чоловіків і 1 для жінок.

```
#code cell 16
#replace the Gender labels in the testing dataframe
# Hint: look at code cell 4
```

в) Замініть пропущені значення віку на середнє значення віку.

```
#code cell 17
#Use the fillna method of the testing dataframe column "Age"
#to replace missing values with the mean of the age values.
      testing["Age"].fillna(testing["Age"].mean(), inplace=True)
```

г) Переконайтеся, що значення були замінені.

Переконайтеся, що пропущені значення заповнено, а мітки статі *Gender* мають значення 0 і 1.

```
#code cell 18
#verify the data preparation steps. Enter and run both the info and head
#methods from here, by entering and running one and then the other.
```

Крок 6. Позначте набір даних тестування.

На цьому кроці ви застосуєте навчену модель класифікатора дерева рішень до набору даних тестування.

а) Створіть масив вхідних змінних із набору даних тестування.

```
#code cell 19
#create the variable X_input to hold the features that the classifier will use
X_input = testing[list(columns)].values
```

б) Застосуйте модель до набору даних тестування.

Використовуйте метод **predict()** об'єкта clf_train, який був навчений позначати спостереження в наборі даних тестування найбільш вірогідною класифікацією виживання. Надайте масив вхідних змінних із набору даних тестування як параметр для цього методу.

```
#code cell 20
#apply the model to the testing data and store the result in a pandas dataframe.
#Use X_input as the argurment for the predict() method of the clf_train classifi
er object
target_labels = clf_train.predict(X_input)
#convert the target array into a pandas dataframe using the pd.DataFrame() metho
d and target as argument
target_labels = pd.DataFrame({'Est_Survival':target_labels, 'Name':testing['Name
']})
#display the first few rows of the data set
```

в) Оцініть точність передбачуваних міток.

Основну істину щодо виживання кожного пасажира можна знайти у файлі під назвою **all_data.csv**. Щоб вибрати лише пасажирів, які містяться в наборі даних тестування, ми об'єднуємо кадр даних target_labels і фрейм даних all_data у полі імені **Name**. Потім ми порівнюємо оцінку мітки з базовим кадром даних істини та обчислюємо точність вивченої моделі.

```
#code cell 21
#import the numpy library as np
import numpy as np
# Load data for all passengers in the variable all_data
all_data = pd.read_csv("./Data/titanic_all.csv")
# Merging using the field Name as key, selects only the rows of the two datasets
that refer to the same passenger
testing_results = pd.merge(target_labels, all_data[['Name','Survived']], on=['Na
me'])
# Compute the accuracy as a ratio of matching observations to total osbervations
. Store this in in the variable acc.
acc = np.sum(testing_results['Est_Survival'] == testing_results['Survived']) / f
loat(len(testing_results))
# Print the result
```

Оцінка моделі класифікатора дерева рішень

Бібліотека sklearn містить модуль, який можна використовувати для оцінки точності моделі дерева рішень.

Метод **train_test_split()** розділить спостереження у всьому наборі даних на два випадково вибрані масиви спостережень, які складають набори даних для навчання та тестування. Після підгонки моделі до навчальних даних, навчену модель можна оцінити та порівняти точність передбачення як для навчального, так і для тестового наборів даних. Бажано, щоб дві оцінки були близькими, але точність тестового набору даних зазвичай нижча, ніж для набору навчальних даних.

Крок 7. Імпортуйте дані.

Цього разу ми імпортуємо дані з файлу csv, але вказуємо стовпці, які ми хочемо відображати у фреймі даних. Ми зробимо це, передавши подібний до масиву список імен стовпців до параметра usecols методу **read_csv()**. Використаємо такі стовпці: *'Survived', 'Fare', 'Pclass', 'Gender', 'Age', and 'SibSP'*. Кожен має бути в лапках, а список має бути у квадратних дужках.

Назвіть цей фрейм даних all_data.

```
#code cell 22
#import the titanic_all.csv file into a dataframe called all_data. Specify the l
ist of columns to import.
all_data = pd.read_csv("./Data/titanic_all.csv", usecols=['Survived','Pclass','G
ender','Age','SibSp','Fare'])
#View info for the new dataframe
```

Скільки записів у наборі даних?

Яких важливих значень змінних немає, скільки всього відсутніх значень?

Крок 8. Підготуйте дані.

а) Видаліть рядки " male" і " female" і замініть їх на 0 і 1 відповідно.

```
#code cell 23
#Label the gender variable with 0 and 1
```

б) Замініть пропущені значення віку на середнє значення віку всіх пасажирів з набору даних.

```
#code cell 24
#replace missing Age values with the mean age
#display the first few rows of the data set
```

Крок 9. Створіть вхідні та вихідні змінні для даних навчання та тестування.

Бібліотека sklearn включає модулі, які допомагають у виборі моделі. Ми імпортуємо з sklearn.model_selection метод **train_test_split()**. Цей метод автоматично розділить весь набір даних, повернувши загалом чотири масиви чисел, два для функцій (тест і перевірка) і два для міток (тест і перевірка).

Один з параметрів методу визначає частку спостережень, які потрібно використовувати для тестування та навчання. Інший параметр визначає початкове значення, яке використовуватиметься для рандомізації віднесення спостереження до тестування або навчання. Це використовується для того, щоб інший користувач міг повторити вашу роботу, отримуючи ті самі призначення спостережень до наборів даних. Синтаксис методу такий:

```train_test_ split(input_X, target_y, test_size=0,4, random_state=0)```

Тобто 40% даних буде використано для тестування. Випадкове початкове значення встановлюється на 0.

Метод повертає чотири значення. Ці значення є вхідними змінними для даних навчання та тестування та цільовими змінними для даних навчання та тестування в такому порядку.

а) Призначте вхідні та вихідні змінні та згенеруйте масиви.
```

```
#code cell 25
#Import train_test_split() from the sklearn.model_selection libary
from sklearn.cross_validation import train_test_split
#create the input and target variables as uppercase X and lowercase y. Reuse the
columns variable.
X = all_data[list(columns)].values
y = all_data["Survived"].values
#generate the four testing and training data arrays with the train_test_split()
method
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.40, random_stat
e=0)
```

б) Навчіть модель і пристосуйте її до даних тестування.

Модель навчатиметься, використовуючи лише навчальні дані, вибрані функцією train_test_split.

```
#code cell 26
#create the training decision tree object
clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
#fit the training model using the input and target variables
clf_train = clf_train.fit(X_train, y_train)
```

в) Порівняйте моделі, оцінивши кожну.

Використовуйте метод **score()** кожного об’єкта дерева рішень, щоб створити оцінки.

```
#code cell 27
#score the model on the two datasets and store the scores in variables. Convert
the scores to strings using str()
train_score = str(clf_train.score(X_train,y_train))
test_score = str(clf_train.score(X_test,y_test))
#output the values in a test string
print('Training score = '+ train_score+' Testing score = '+test_score)
```

Який результат ви отримали?

Порівняйте оцінки для навченої моделі як за тестовими, так і з валідаційними (перевірочними) даними.

Оцінка точності тесту близька, але нижча, ніж оцінка для навчальних даних. Це пояснюється тим, що зазвичай модель має тенденцію переповнювати навчальні дані, тому оцінка тесту є кращою оцінкою того, як модель здатна узагальнювати поза навчальними даними.

Вимоги до оформлення звіту

Звіт має включати:

1. Титульний аркуш.
2. Завдання на комп'ютерний практикум.
3. Хід роботи. Цей розділ складається з послідовного опису виконуваних кроків згідно інструкцій до комп'ютерного практикуму.
4. Висновки.

Питання для самоперевірки

1. Дайте визначення методу дерев рішень (decision trees). Для чого він використовується?
2. Як будуються та використовуються бінарні дерева? Наведіть приклад.
3. Які існують переваги методу дерева рішень?
4. Які існують критерії розщеплення дерева рішень?
5. Які існують варіанти зупинки навчання дерева рішень?
6. Які інструменти Python використовуються для побудови дерева рішень?
7. Як реалізовується оцінка моделі класифікатора дерева рішень?

Рекомендована література

1. Методи дерев рішень, класифікації та прогнозування // Електронний ресурс. https://moodle.znu.edu.ua/pluginfile.php?file=/486136/mod_resource/content/1/%d0%9b%d0%b5%d0%ba%d1%86%d1%96%d1%8f%209.pdf
2. Decision Tree // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/python_ml_decision_tree.asp
3. Decision Tree // Електронний ресурс. Режим доступу: https://scikit-learn.org/stable/modules/tree.html
4. What is a decision tree diagram // Електронний ресурс. Режим доступу: https://www.lucidchart.com/pages/decision-tree

КОМП'ЮТЕРНИЙ ПРАКТИКУМ 3.
ВИКОРИСТАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ CNN ДЛЯ ЗАДАЧІ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

Мета роботи*:* використати бібліотеки Python для завантаження, дослідження та аналізу набору даних, змінити розмір та масштаб зображень, перетворити мітки даних у вектори для подальшого кодування, розділити дані на набори для навчання та тестування моделі; побудувати модель нейронної мережі CNN: навчити та оцінити побудовану модель, побудувати графіки точності та втрат; оцінити нову модель та порівняти результати; зробити прогнози щодо даних випробувань, перетворити ймовірності у мітки класів і побудувати тестові зразки, які модель класифікує правильно, візуалізувати звіт про класифікацію даних.

Теоретичні відомості

Штучні нейронні мережі є моделями нейронної структури мозку, який здатен сприймати, обробляти, зберігати та продукувати інформацію, що представлена образами. Особливістю мозку також є навчання та самонавчання на власному досвіді. Адаптивні системи на основі штучних нейронних мереж дозволяють з успіхом вирішувати проблеми розпізнавання образів, виконання прогнозів, оптимізації, асоціативної пам'яті, керування та інші інтелектуальні завдання, не використовуючи традиційного програмування.

Усі штучні нейромережі конструюються з базового формуючого блоку – **штучного нейрону**, що моделює основні функції природного нейрона. При функціонуванні штучний нейрон одночасно отримує багато вхідних сигналів. Кожен вхід має власний коефіцієнт, що називається **синаптичною вагою** і моделює різноманітні синаптичні з'єднання біологічних нейронів. Ваговий коефіцієнт збільшує або зменшує значення входу, тому, ваги суттєвого входу підсилюються і, навпаки, вага несуттєвого входу примусово зменшується, що визначає інтенсивність вхідного сигналу. Ваги можуть змінюватись відповідно до навчальних прикладів, топології мережі та навчальних правил.

Вхідні сигнали x_n помножені на вагові коефіцієнти з'єднання w_n додаються (блок суматора), проходять через передатну функцію та генерують результат (рис.3.1).

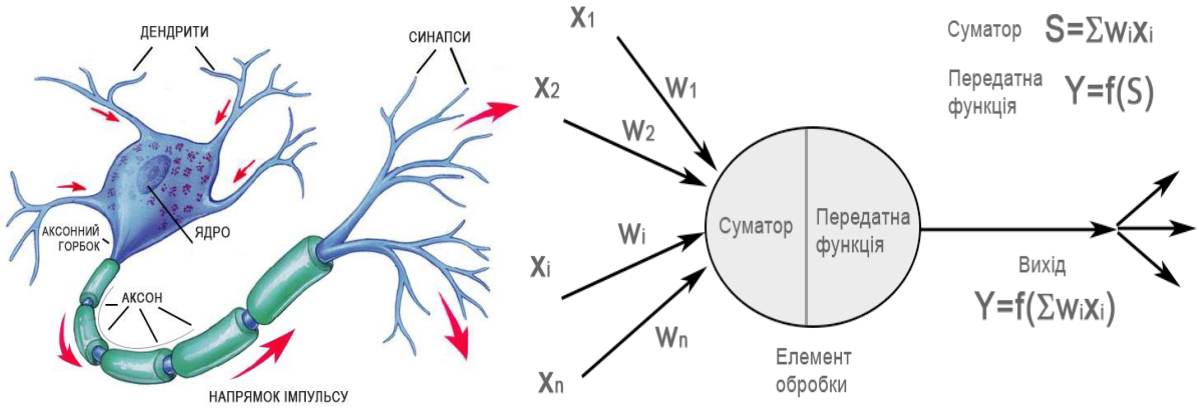


Рис. 3.1. Базовий штучний нейрон

Поглиблені знання щодо будови біологічного нейрона, як ефективного перетворюючого інструмента, можна розглядати як джерело базових ідей та концепцій по створенню нових типів нейромереж. В наявних на цей час нейронних мережах штучні нейрони мають значно більше можливостей, ніж базовий нейрон, що описано вище. На рис. 3.2 зображено детальну схему штучного нейрону.

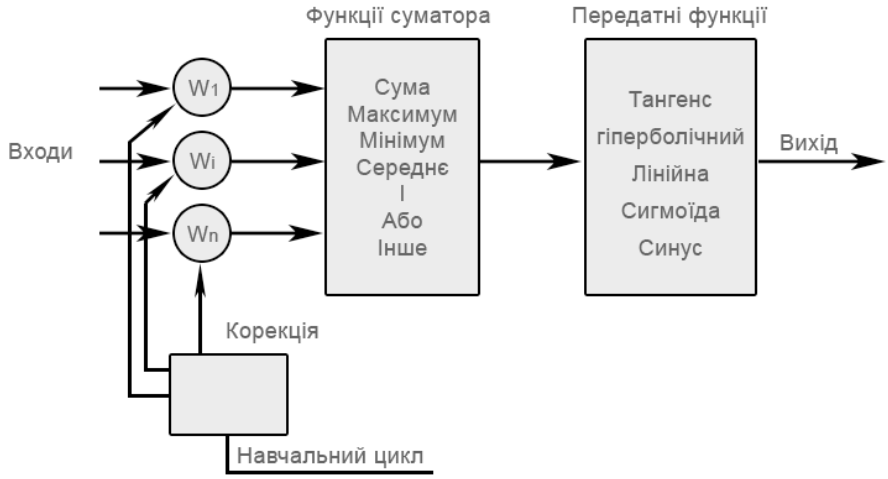


Рис. 3.2. Модель штучного нейрону

Модифіковані входи (зважені синаптичними коефіцієнтами) надходять до блока суматора, де, можна не лише додавати входи, а й виконати інші операції, наприклад, обрати середнє, найбільше чи найменше значення входів, здійснити логічні операції AND чи OR, або виконати інші функції.

Іноді до блоку суматора додається функції активації, яка дозволяє функції суматора зміщуватися в часі. Значення, що спродуковано в блоці суматора надсилається до передатної функції, яка за допомогою певного алгоритму, обмежує вихід в певному діапазоні, наприклад, [0 ÷ 1] чи [-1÷1]. В існуючих нейромережах в якості передатних функцій можуть бути використані сигмоїда, синус, гіперболічний тангенс та ін. (рис. 3.3).

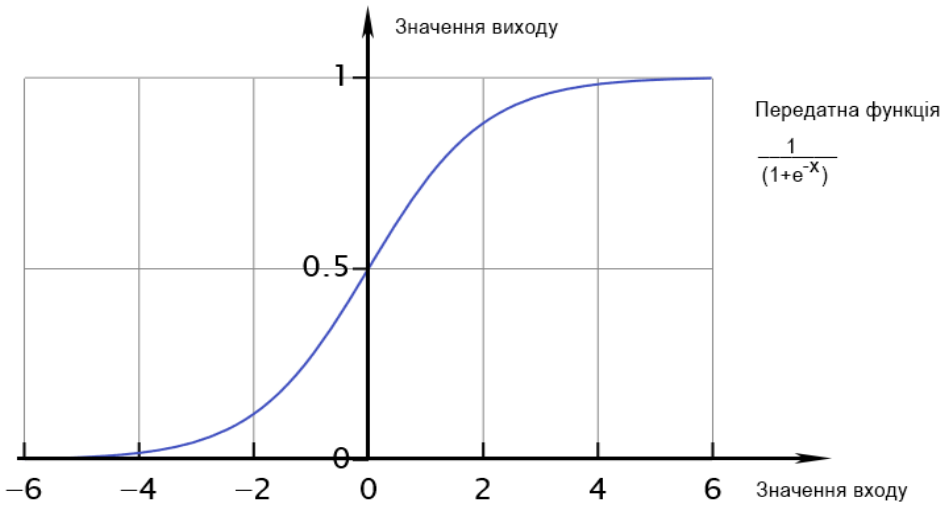


Рис. 3.3. Сигмоїдна передатна функція

Результат передатної функції надходить до входів інших нейронів або до зовнішнього з'єднання, відповідно до структури нейромережі.

Компоненти штучного нейрона

Незалежно від розташування та функціонального призначення, штучні нейрони мають спільні компоненти. Нижче представлено сім основних компонентів.

Компонента 1. Нормалізація входів

Нормалізація вхідних даних – це процес, при якому всі вхідні дані проходять процес "вирівнювання", тобто приведення до певного інтервалу, наприклад [0,1]. Нормалізація дозволяє значно підвищити швидкість збіжності алгоритму навчання нейронної мережі, оскільки не можна порівнювати величини різних порядків. Якщо не провести нормалізацію, то вхідні дані будуть надавати додатковий вплив на нейрон, що може привести до невірних рішень.

Компонента 2. Вагові коефіцієнти

До нейрону одночасно надходять сигнали від багатьох входів. Кожен вхід змінює своє значення відповідно до коефіцієнта синаптичної ваги. Ваги суттєвого входу підсилюються і навпаки вага несуттєвого входу примусово зменшується, що визначає інтенсивність вхідного сигналу. Ваги можуть змінюватись відповідно до алгоритму навчання. Після проходження навчання ваги фіксуються і в подальшому використовуються.

Компонента 3. Функція суматора

Першим кроком дії нейрону є обчислення зваженої суми всіх входів. Математично, вхідні сигнали та відповідні їм ваги представлено векторами (x_{10}, x_{20} ... x_{n0}) та (w_{10}, w_{20} . . . w_{n0}). Добуток цих векторів є загальним вхідним сигналом.

Спрощеною функцією суматора є множення кожної компоненти вектора x на відповідну компоненту вектора w: вхід$_1$ = x_1 * w_1, вхід$_2$ = x_2 * w_2, і знаходження суми всіх добутків: вхід$_1$ + вхід$_2$ + . . . + вхід$_n$. Результатом є не багатоелементний вектор, а єдине число.

Функція суматора може бути складнішою, наприклад, вибір мінімуму, максимуму, середнього арифметичного, добутку або інший нормалізуючий алгоритм. Вхідні сигнали та вагові коефіцієнти можуть комбінуватись багатьма способами перед надходженням до передатної функції. Особливі алгоритми для комбінування входів нейронів визначаються обраними мережною архітектурою та парадигмою (алгоритмом навчання).

В деяких нейромережах функції суматора виконують додаткову обробку, так звану функцію активації, яка зміщує вихід функції суматора відносно часу.

Компонента 4. Передатна функція

Результат функції суматора перетворюється у вихідний сигнал через алгоритмічний процес відомий як передатна функція. У передатній функції для визначення виходу нейрона загальна сума порівнюється з деяким порогом. Якщо сума є більшою за значення порогу, елемент обробки генерує сигнал, в противному випадку сигнал не генерується або генерується гальмуючий сигнал.

Компонента 5. Вихідна функція (змагання)

За аналогією з біологічним нейроном, кожний штучний нейрон має один вихідний сигнал, який передається до сотень інших нейронів. Переважно, вихід є прямо пропорційним до результату передатної функції. В деяких мережних топологіях результати передатної функції змінюються для створення змагання між сусідніми нейронами. Нейронам дозволяється змагатися між собою, блокуючи дії нейронів, що мають слабший сигнал.

Змагання (конкуренція) може відбуватись між нейронами, які знаходяться на одному або різних прошарках. По-перше, конкуренція визначає, який штучний нейрон буде активним і забезпечить вихідний сигнал. По-друге, конкуруючі виходи допомагають визначити, який нейрон буде брати участь у процесі навчання.

Компонента 6. Функція похибки та поширюване назад значення

У більшості мереж, що застосовують контрольоване навчання обчислюється різниця між продукованим та бажаним виходом. Похибка відхилення (біжуча похибка) обробляється функцією похибки відповідно до заданої мережної архітектури. Після проходження всіх прошарків поточна похибка поширюється назад до попереднього прошарку, алгоритм навчання корегує вагові коефіцієнти, що враховується в наступному циклі навчання.

Компонента 7. Функція навчання

Метою функції навчання є налаштування вагових коефіцієнтів на входах кожного елемента обробки відповідно до певного алгоритму навчання для досягнення бажаного результату.

Існує два типи навчання: контрольоване та неконтрольоване. Контрольоване навчання вимагає навчальної множини даних або вчителя, що ранжирує ефективність результатів мережі. У випадку неконтрольованого навчання система самоорганізовується за внутрішнім критерієм, закладеним в алгоритм навчання.

Навчання нейронних мереж

Оригінальність нейронних мереж, як аналога біологічного мозку, полягає у здібності до навчання за прикладами, що складають навчальну множину. Процес навчання нейромереж розглядається як налаштування архітектури та вагових коефіцієнтів синаптичних зв'язків відповідно до даних навчальної множини так, щоб ефективно вирішити поставлену задачу.

Після визначення архітектури нейронної мережі її потрібно «навчити», тобто підібрати такі значення її ваг, щоб вона працювала належним чином. Процес навчання нейронної мережі полягає в налаштуванні її внутрішніх параметрів під конкретну задачу (рис.3.4).

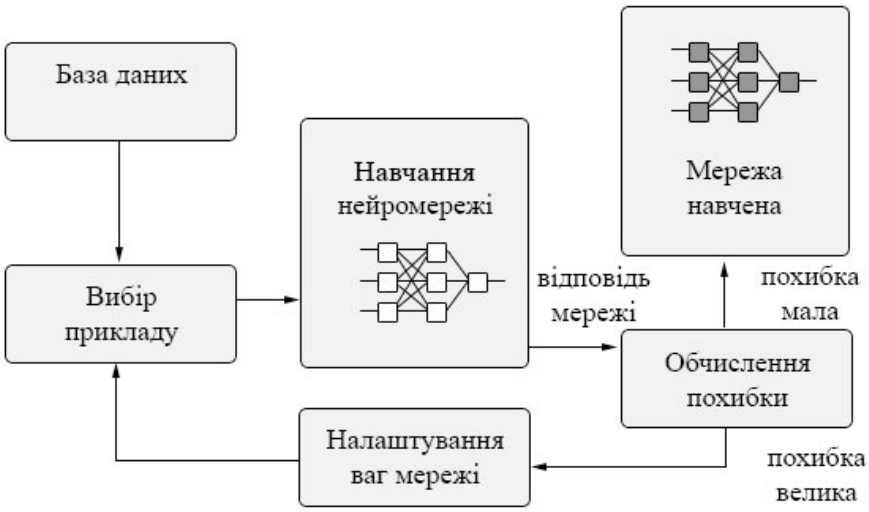


Рис. 3.4. Процес навчання нейронної мережі

Алгоритм роботи нейронної мережі є ітеративним, його кроки називають епохами або циклами.

Епоха – одна ітерація в процесі навчання, що містить пред'явлення всіх прикладів з навчальної множини і, можливо, перевірку якості навчання на контрольній множині.

Процес навчання здійснюється на **навчальній вибірці (множині)**. Навчальна вибірка містить набір даних про предметну область (об’єкт, явище, процес), що поділяється на вхідні значення і відповідні їм вихідні значення. Наприклад, після пред'явлення зображення літери "А" на вхід нейронної мережі, вона видає деяку відповідь, не обов'язково вірну.

У навчальній множині присутня вірна (бажана) відповідь, і сенсом навчання є те, щоб на виході нейронної мережі з міткою "А" рівень сигналу був максимальний. В ході навчання нейронна мережа знаходить певні залежності вихідних полів від вхідних. Зазвичай, в якості бажаного виходу в задачі класифікації беруть набір (1, 0, 0, ...), де 1 стоїть на виході з міткою "А", а 0 – на усіх інших виходах. Обчислюючи різницю між бажаною і реальною відповідями мережі, утворюється похибка, що надалі обробляється відповідною функцією. Функція похибки – це цільова функція, що дозволяє оцінити якість роботи нейронної мережі і потребує мінімізації в процесі керованого навчання нейронної мережі.

Алгоритмом навчання є набір формул, який дозволяє за функцією похибки обчислити необхідні зміни для вагових коефіцієнтів зв’язків нейронної мережі.

Одну літеру (або різні зображення однієї літери) можна пред'являти нейронній мережі багато разів. У цьому сенсі навчання швидше нагадує повторення вправ в спорті – тренування. Після багаторазового пред'явлення прикладів вагові коефіцієнти зв’язків нейронної мережі стабілізуються, причому нейронна мережа надає правильні відповіді на всі (або майже всі) приклади з навчальної множини. У такому випадку говорять, що "нейронна мережа вивчила всі приклади", "нейронна мережа навчена", або "нейронна мережа натренована".

Які вхідні поля (ознаки) необхідно використовувати? Спочатку здійснюється евристичний вибір, далі кількість входів може бути змінено. Складність може викликати питання про кількість прикладів в наборі даних. Вся інформація, яку нейронна мережа має про завдання, міститься в наборі прикладів. Тому, якість навчання нейронної мережі безпосередньо залежить від кількості прикладів в навчальній вибірці, а також від того, наскільки повно ці приклади описують це завдання. Так, наприклад, безглуздо використовувати нейронну мережу для передбачення фінансової кризи, якщо в навчальній вибірці не представлено жодної кризи. Для повноцінного навчання нейронної мережі потрібна репрезентативна і доволі велика вибірка з сотні (а краще тисячі) прикладів. Кількість необхідних прикладів залежить від складності розв'язуваної

задачі. При збільшенні кількості ознак кількість прикладів зростає нелінійно, ця проблема носить назву "прокляття розмірності". За недостатньої кількості даних рекомендується використовувати лінійну модель. Розробник має надати можливості вибору кількості прошарків у мережі і кількості нейронів в кожному прошарку. Далі необхідно призначити такі значення ваг і зсувів, які зможуть мінімізувати похибку рішення. Від якості навчання нейронної мережі залежить її здатність вирішувати поставлені перед нею завдання.

Перенавчання нейронної мережі

При навчанні нейронних мереж часто виникає проблема перенавчання (**overfitting**). Перенавчання, або надмірно близька підгонка – надлишкова точна відповідність нейронної мережі до конкретного набору навчальних прикладів, при якому мережа втрачає здатність до узагальнення. Перенавчання виникає у разі занадто довгого навчання, недостатньої кількості навчальних прикладів або занадто складної структури нейронної мережі.

Перенавчання пов'язано з тим, що вибір навчальної множини є випадковим. З перших кроків навчання відбувається зменшення похибки. На наступних кроках з метою зменшення похибки (цільової функції) параметри підлаштовуються під особливості навчальної множини. Однак при цьому відбувається "підлаштування" не під загальні закономірності ряду, а під особливості його частини – навчальної підмножини. При цьому точність прогнозу зменшується.

Один з варіантів боротьби з перенавчанням мережі - розділення навчальної вибірки на дві множини (навчальну і тестову). На навчальній множині відбувається навчання нейронної мережі. На тестовій множині здійснюється перевірка побудованої моделі. Ці множини не повинні перетинатися.

Глибоке навчання (deep learning) – це сукупність алгоритмів, що використовуються у машинному навчанні для моделювання абстракцій високого рівня даних за допомогою модельних архітектур, які складаються з множини нелінійних перетворень.

Це підхід, який використовується для побудови та навчання нейронних мереж. Алгоритм вважається глибоким, якщо вхідні дані передаються через низку перетворень, перш ніж вони стають вихідними. Спостереження (наприклад, зображення) може бути представлено багатьма способами, такими як вектор значень яскравості для пікселів [1].

Специфічним типом глибокої нейронної мережі є згортка, яку називають **CNN (convolutional neural network)** або **ConvNet**. Це штучна нейронна мережа з прямим зв’язком. Нейронні мережі з прямим зв’язком (feed-forward) також називають багатошаровими персептронами (MLP, multi-layer perceptrons). Моделі називаються «feed-forward», оскільки інформація протікає прямо через модель. Немає з'єднань зворотного зв'язку, в яких виходи моделі повертаються в саму себе.

CNN натхненні біологічною зоровою корою людини. У корі є невеликі ділянки клітин, чутливі до певних ділянок поля зору. Ця ідея була розширена з експериментом, проведеним Хубелем і Візелем у 1962 році. У цьому експерименті дослідники показали, що деякі окремі нейрони в мозку активувалися або запускалися лише за наявності країв певної орієнтації, наприклад, вертикальних або горизонтальних країв.

Деякі нейрони спрацьовують, коли вони потрапляють на вертикальні краї, а деякі – коли показують горизонтальний край. Х'юбель і Візель виявили, що всі ці нейрони були добре упорядковані в стовпчастий спосіб і разом вони здатні створювати візуальне сприйняття. Згорткові нейронні мережі були одними з найвпливовіших нововведень у сфері комп’ютерного зору. Ці нейронні мережі виявилися успішними в багатьох різних реальних дослідженнях:

- класифікація зображень, виявлення об'єктів, сегментація, розпізнавання обличчя;
- автомобілі з автономним керуванням, які використовують системи комп’ютерного зору на базі CNN;
- класифікація кристалічної структури за допомогою згорткової нейронної мережі тощо.

У 2012 році Алекс Крижевський використовував згорткові нейронні мережі, щоб виграти конкурс ImageNet, зменшивши помилку класифікації з 26% до 15%. Конкурс ImageNet Large Scale Visual Recognition Challenge (ILSVRC) розпочався в 2010 році, у цьому конкурсі дослідницькі групи оцінюють свої алгоритми на заданому наборі даних і змагаються за досягнення більшої точності для кількох задач візуального розпізнавання.

Цей час часто називають «третьою хвилею нейронних мереж». Дві інші хвилі були в 1940-1960-х і в 1970-1980-х роках. Рис. 3.5. показує, зображення подається як вхідні дані в мережу, яка проходить через численні згортки, підвибірку, повністю підключений шар і, нарешті, виводить результат.

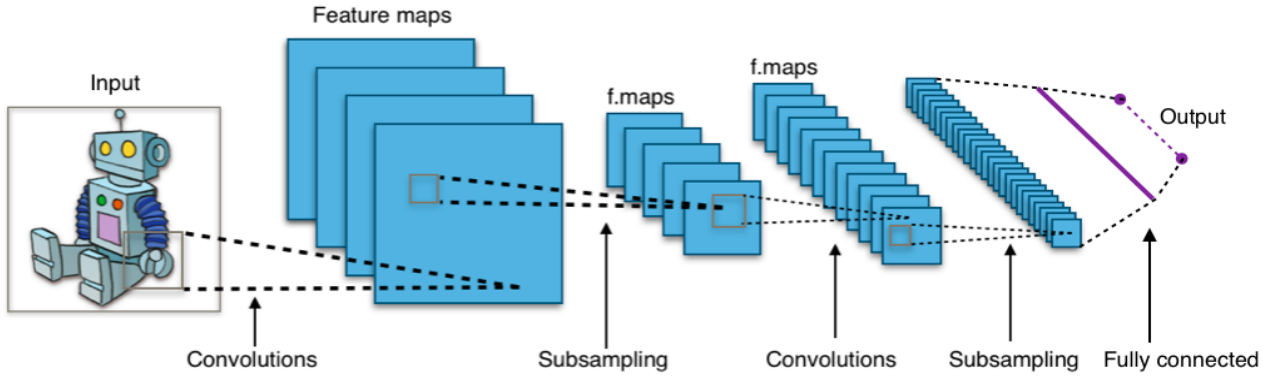


Рис.3.5. Обробка вхідного зображення за допомогою CNN

Шар згортки обчислює вихід нейронів, які підключені до локальних областей або сприйнятливих полів на вході, кожен з яких обчислює добуток між їх вагами та невеликим сприйнятливим полем, з яким вони підключені у вхідному обсязі. Кожне обчислення призводить до вилучення карти об’єктів із вхідного зображення. Наприклад, якщо зображення представлене у вигляді матриці значень 5x5, обирається матриця 3x3, це вікно або ядро розміром 3x3 пересувається навколо зображення. У кожній позиції цієї матриці перемножається значення вікна 3x3 на значення на зображенні, які в даний момент покриваються вікном. В результаті отримується одне число, яке представляє усі значення у цьому вікні зображень. Цей шар використовується для фільтрації при переміщенні вікна над зображенням. Фільтри помножуються на значення, виведені за допомогою згортки.

Метою підвибірки є отримання вхідного представлення шляхом зменшення його розмірів, що допомагає зменшити перенавчання моделі (overfitting). Одним із прийомів підвибірки є **максимальне об'єднання (max pooling)**. За допомогою цієї техніки обирається найвище значення пікселя з області залежно від її розміру. Максимальний пул приймає найбільше значення з вікна зображення, яке в даний момент охоплює ядро. Наприклад, шар із максимальним об'єднанням розміром 2x2, який вибирає максимальне значення інтенсивності пікселів із області 2x2. Ядро або вікно і переміщається над зображенням. Функція, яка застосовується до ядра, і вікно зображення не є лінійними.

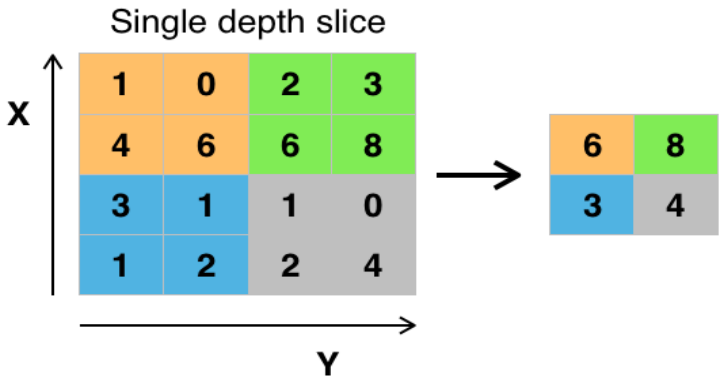


Рис.3.6. Max-Pooling [5]

Метою повнозв'язаного шару є вирівнювання високорівневих функцій, які вивчаються згортковими шарами та об'єднання усіх функцій. Цей шар передає вихід на вихідний шар, де використовується класифікатор softmax або сигмоїда для прогнозування мітки вхідного класу.

Згорткова нейронна мережа утворює клас штучних нейронних мереж, який став домінуючим у різних задачах комп'ютерного зору. CNN розроблено для автоматичного та адаптивного вивчення просторової ієрархії функцій за допомогою кількох будівельних блоків, таких як шари згортки, шари об'єднання та повністю пов'язані шари.

Перші два, шари згортки та об'єднання, виконують вилучення ознак, тоді як третій, повністю пов'язаний шар, відображає вилучені об'єкти в кінцевий результат, наприклад, класифікацію. Рівень згортки відіграє ключову роль у CNN та складається з набору математичних операцій, таких як згортка (спеціалізований тип лінійної операції).

У цифрових зображеннях значення пікселів зберігаються в двовимірній (2D) сітці, тобто в масиві чисел (рис. 3.7), і невелика сітка параметрів, що називається **ядром**, оптимізований екстрактор функцій, застосовується до кожної позиції зображення, що робить CNN високоефективними для обробки зображень, оскільки функція може зустрічатися в будь-якому місці зображення.

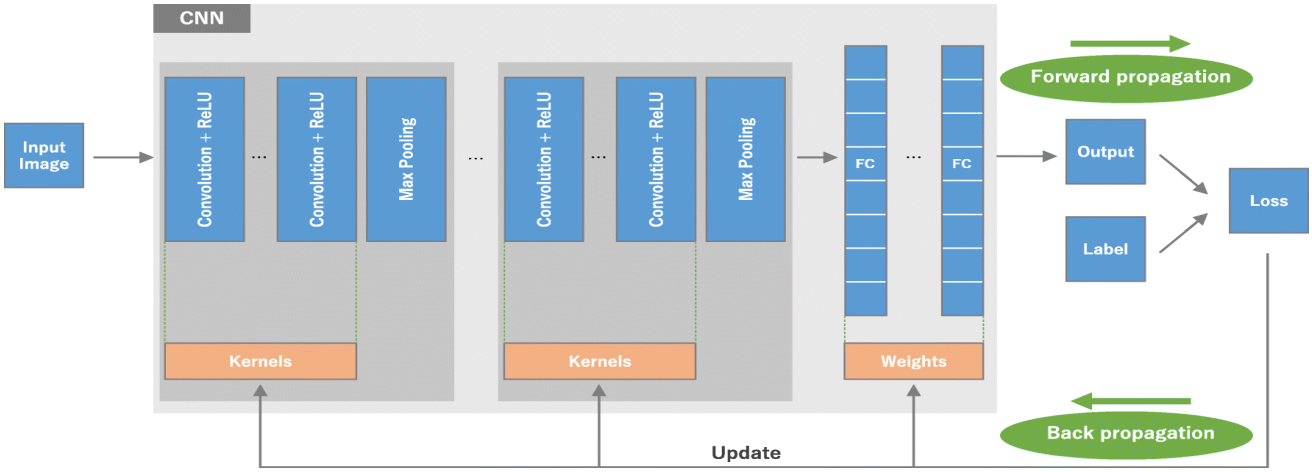


Рис. 3.7. Архітектура згорткової нейронної мережі та процес її навчання [5]

CNN складається з стеку кількох будівельних блоків: **шарів згортки, шарів об'єднання (наприклад, максимальне об'єднання) та шарів повного зв'язку (FC, fully connected)**. Продуктивність моделі для певних ядер і ваг розраховується з функцією втрат шляхом прямого поширення на навчальному наборі даних, а параметри, які можна вивчати, тобто ядра та ваги, оновлюються відповідно до значення втрат за допомогою зворотного поширення за допомогою алгоритму оптимізації градієнтного спуску. Оскільки один шар передає свій вихід на наступний шар, витягнуті об'єкти можуть ієрархічно та поступово ставати більш складними. Процес оптимізації таких параметрів, як ядра, називається **навчанням**, виконується таким чином, щоб мінімізувати різницю між виходами та основними мітками істини за допомогою алгоритму оптимізації.

Згортка

Згортка — це спеціалізований тип лінійної операції, що використовується для вилучення ознак, де невеликий масив чисел, який називається **ядром**, застосовується до вхідних даних, що є масивом чисел, який називається **тензором**.

Поелементний добуток між кожним елементом ядра та вхідним тензором обчислюється у кожному місці тензора і підсумовується, щоб отримати вихідне значення у відповідній позиції вихідного тензора, яке називається **картою ознак (feature map)** (рис. 3.8. а–в). Ця процедура повторюється із застосуванням кількох ядер для формування довільної кількості карт ознак, які представляють різні характеристики вхідних тензорів.

Таким чином, різні ядра можна розглядати як екстрактори різних ознак. Двома ключовими гіперпараметрами, які визначають операцію згортки, є розмір і кількість ядер. Перший зазвичай дорівнює 3×3, але іноді 5×5 або 7×7. Останній є довільним і визначає глибину вихідних карт об’єктів.

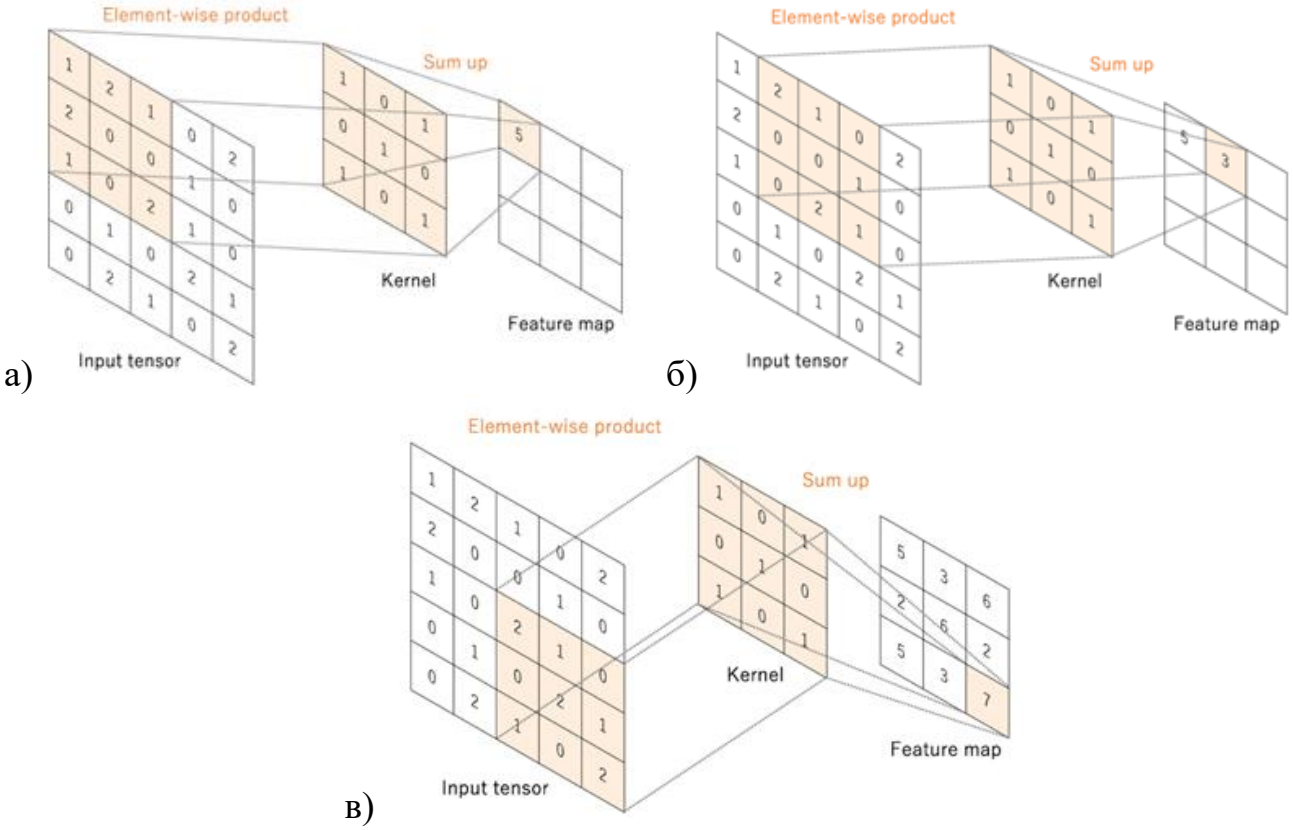


Рис. 3.8. Приклад операції згортки з розміром ядра 3 × 3, без заповнення та кроком 1 [5]

Операція згортки не дозволяє центру кожного ядра перекривати крайній елемент вхідного тензора і зменшує висоту та ширину вихідної карти об’єктів порівняно з вхідним тензором. Заповнення, як правило, нульове заповнення, є методом вирішення цієї проблеми, коли рядки та стовпці нулів додаються з кожної сторони вхідного тензора, щоб умістити центр ядра на крайньому зовнішньому елементі та зберегти площину після операції згортки (рис. 3.9).

Сучасні архітектури CNN зазвичай використовують нульове заповнення, щоб зберегти розміри в площині та застосувати більше шарів. Без нульового заповнення кожна наступна карта ознак буде зменшуватися після операції згортки.

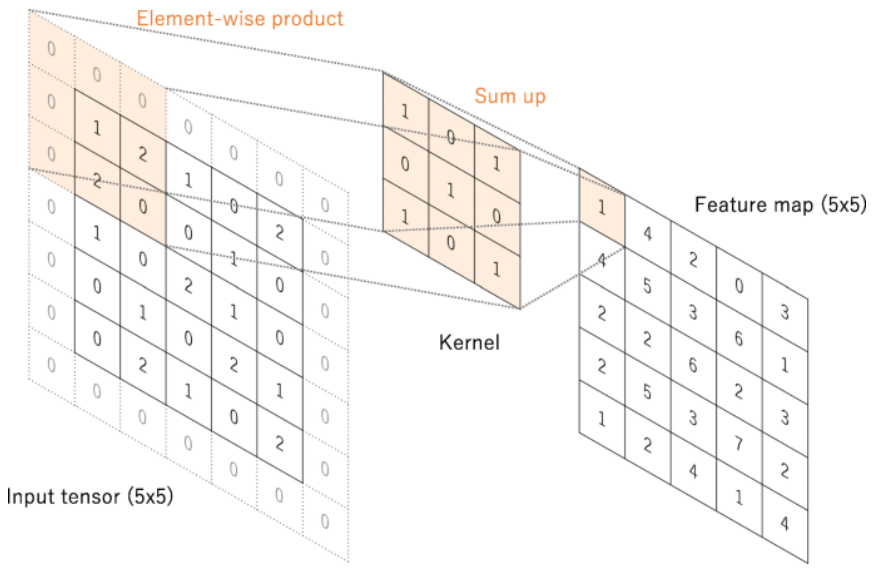


Рис. 3.9. Приклад операції згортки з нульовим заповненням для збереження розмірів в площині, вхідний розмір 5 × 5 зберігається у вихідній карті об’єктів, розмір ядра і крок встановлені як 3 × 3 і 1 відповідно [5]

Відстань між двома послідовними положеннями ядра називається **кроком**, який також визначає операцію згортки. Загальний вибір кроку – 1; однак крок, більший за 1, іноді використовується для досягнення зниження дискретизації карт ознак. Альтернативною технікою виконання зниження дискретизації є операція об'єднання.

Ключовою особливістю операції згортки є розподіл ваги: ядра є спільними для всіх позицій зображення. Розподіл ваги створює такі характеристики операцій згортки:

1. дозволяючи шаблонам локальних ознак, витягнутим за допомогою трансляції ядра, бути інваріантними, оскільки ядра переміщуються по всіх позиціях зображення та виявляють вивчені локальні шаблони;

2. вивчення просторової ієрархії шаблонів ознак шляхом зменшення дискретизації в у поєднанні з операцією об’єднання, що призводить до захоплення більшого поля зору та підвищення ефективності моделі за рахунок зменшення кількості параметрів для вивчення.

Процес навчання моделі CNN для шару згортки полягає у визначенні ядер, які найкраще працюють для заданого набору навчальних даних. Ядра є єдиними параметрами, які автоматично вивчаються під час процесу навчання у шарі згортки; розмір ядер, кількість ядер, заповнення та крок є гіперпараметрами, які необхідно встановити перед початком процесу навчання (табл. 3.1).

Таблиця 3.1. Список параметрів і гіперпараметрів у згортковій нейронній мережі (CNN)

	Параметри	**Гіперпараметри**
Шар згортки	Ядра	Розмір ядра, кількість ядер, крок, функція активації
Об’єднуючий шар	Жодного	Метод об’єднання, розмір фільтра, крок, заповнення
Повністю пов'язаний шар	Ваги	Кількість ваг, функція активації
Інші		Архітектура моделі, оптимізатор, швидкість навчання, функція втрат, розмір міні-пакету, епохи, регуляризація, ініціалізація ваги, поділ набору даних

Параметр – це змінна, яка автоматично оптимізується під час процесу навчання, а **гіперпараметр** – це змінна, яку потрібно встановити заздалегідь.

Нелінійна функція активації

Вихідні дані лінійної операції, наприклад згортки, потім передаються через нелінійну функцію активації. Хоча гладкі нелінійні функції, такі як сигмоїда або гіперболічний тангенс (tanh), використовувалися раніше, найпоширенішою нелінійною функцією активації, яка використовується зараз, є **ReLU (rectified linear unit):** ***f(x) = max(0, x)*** (рис. 3.10)

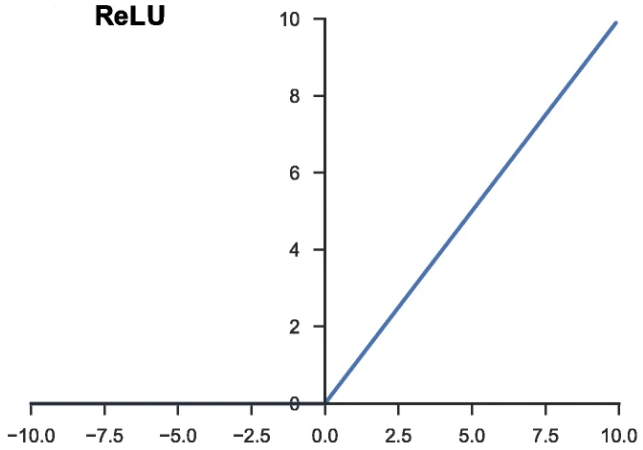


Рис. 3.10. Функція активації ReLU (rectified linear unit) [5]

Об’єднуючий шар

Рівень об’єднання забезпечує операцію зниження дискретизації, яка зменшує розмірність у площині карт ознак, щоб ввести інваріантність трансляції до невеликих зсувів і спотворень, а також зменшити кількість наступних параметрів, які можна вивчати. У жодному з шарів об’єднання немає параметрів, які можна вивчати, тоді як розмір фільтра, крок і заповнення є гіперпараметрами в операціях об’єднання, подібними до операцій згортки.

Максимальне об'єднання

Найпопулярнішою формою операції об’єднання є максимальний пул, який витягує діянки з вхідних карт об’єктів, виводить максимальне значення у кожному патчі та відкидає всі інші значення (рис. 3.11). На практиці зазвичай використовується максимальне об’єднання з фільтром розміром 2×2 з кроком 2. Це зменшує розмірність у площині карт об’єктів у 2 рази. На відміну від висоти та ширини, розмір глибини карт об’єктів залишається незмінним.

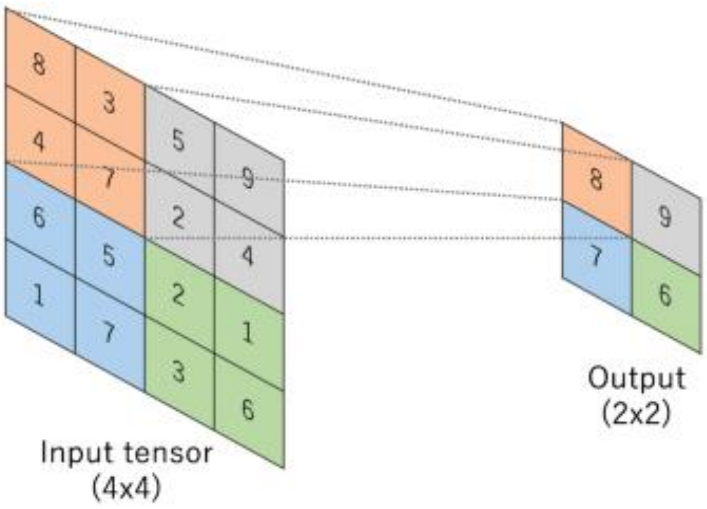


Рис. 3.11. Приклад операції максимального об'єднання, розмір фільтра 2×2, без заповнення, який витягує 2×2 патчі з вхідних тензорів, виводить максимальне значення в кожному патчі та відкидає всі інші значення [5]

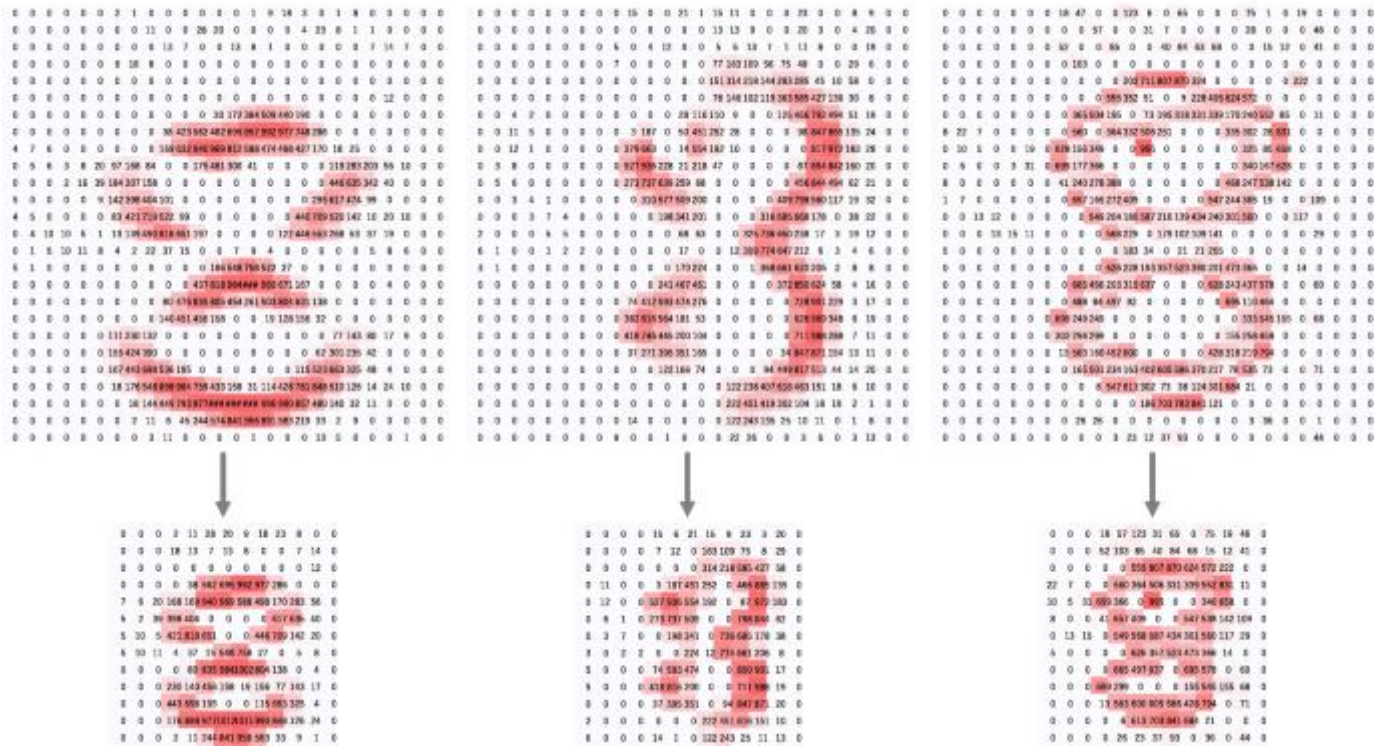


Рис. 3.12. Приклади операції максимального об'єднання для зображень, які зменшуються в 2 рази, з 26×26 до 13×13 [5]

Глобальне середнє об’єднання

Глобальне середнє об’єднання виконує екстремальний тип зниження дискретизації, коли карта об’єктів розміром висота×ширина зменшується до масиву 1×1, взявши середнє значення всіх елементів у кожній карті об’єктів, тоді як глибина карт об’єктів зберігається. Ця операція зазвичай застосовується лише один раз перед повністю з’єднаними шарами. Переваги застосування глобального середнього пулу полягають у зменшенні кількості параметрів, які можна вивчати та дозволяє CNN приймати вхідні дані змінного розміру.

Повністю пов'язаний шар

Вихідні карти характеристик остаточного шару згортки або об’єднання зазвичай згладжуються, тобто перетворюються в одновимірний (1D) масив чисел (або вектор) і з’єднуються з одним або кількома повністю пов’язаними шарами, також відомими як щільні шари, в якому кожен вхід пов’язаний з кожним виходом за допомогою ваги. Після того, як об’єкти, виділені шарами згортки та зменшені шарами об’єднання, створені, вони відображаються підмножиною повністю пов’язаних шарів на кінцеві вихідні дані мережі, наприклад,

ймовірності для кожного класу в завданнях класифікації. Останній повністю підключений шар має таку ж кількість вихідних вузлів, як і кількість класів. За кожним повністю підключеним шаром слідує нелінійна функція, така як ReLU.

Функція активації останнього шару

Функція активації, застосована до останнього повністю підключеного шару, зазвичай відрізняється від інших. Відповідно до кожного завдання необхідно вибрати відповідну функцію активації. Функція активації softmax застосовується до задач мультикласової класифікації, є функцією, яка нормалізує вихідні реальні значення з останнього повністю підключеного шару до ймовірностей цільового класу, де кожне значення знаходиться в діапазоні від 0 до 1, а сума всіх значень дорівнює 1.

Навчання нейронної мережі CNN

Навчання нейронної мережі CNN – це процес пошуку ядер у шарах згортки та ваг у повністю зв’язаних шарах, що мінімізує відмінності між вихідними передбаченнями та заданими основними мітками істинності в наборі навчальних даних.

Алгоритм зворотного поширення помилки (англ. backpropagation) – це метод, який використовується для навчання нейронних мереж, де важливу роль відіграють функція втрат і алгоритм оптимізації градієнтного спуску. Продуктивність моделі для певних ядер і ваг розраховується функцією втрат шляхом прямого поширення на навчальному наборі даних, а параметри, які можна вивчати, а саме ядра та ваги, оновлюються відповідно до значення втрат за допомогою алгоритму зворотного поширення та градієнтного спуску.

Функція втрат (Loss)

Функція втрат, яку також називають функцією вартості, вимірює сумісність між вихідними передбаченнями нейронної мережі через пряме поширення та заданими мітками істинності. Зазвичай функцією втрат для багатокласової класифікації є перехресна ентропія, тоді як середня квадратична помилка зазвичай застосовується до регресії для неперервних значень змінних. Тип функції втрат є одним із гіперпараметрів і потребує визначення відповідно до поставлених завдань.

Градієнтний спуск (Gradient descent)

Градієнтний спуск – це алгоритм оптимізації, який ітеративно оновлює доступні для навчання параметри, тобто ядра та ваги мережі, щоб мінімізувати втрати. Градієнт функції втрат дає напрямок, у якому функція має найбільшу швидкість зростання, і кожен параметр, який можна вивчати, оновлюється в негативному напрямку градієнта з довільним розміром кроку, визначеним на основі гіперпараметра α, який називається **швидкістю навчання** (рис. 3.13). Математично градієнт є частинною похідною від втрати по відношенню до кожного параметра, оновлення параметра обчислюється таким чином:

$$w := w - \alpha * \frac{\partial L}{\partial w}, \qquad (3.1)$$

де w означає кожен параметр, який можна вивчати, α означає швидкість навчання, а L означає функцію втрат.

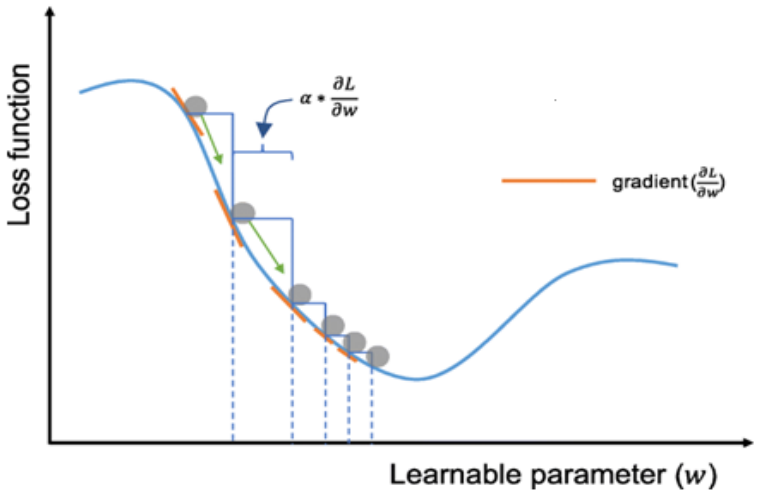


Рис. 3.13. Градієнтний спуск [5]

Швидкість навчання є одним із найважливіших гіперпараметрів, які необхідно встановити перед початком навчання. З таких причин, як обмеження пам’яті, градієнти функції втрат щодо параметрів обчислюються за допомогою підмножини навчального набору даних, що називається міні-пакетом, і застосовуються до оновлення параметрів. Цей метод **називається градієнтним спуском міні-партії**, який також часто називають **стохастичним градієнтним спуском (stochastic gradient descent, SGD)**, а розмір міні-партії також є гіперпараметром. Також використовується багато вдосконалень алгоритму градієнтного спуску, таких як SGD з імпульсом, RMSprop, Adam.

Мітки даних (Labels)

Дані та основні позначки істини є найважливішими компонентами в дослідженнях із застосуванням глибокого навчання або інших методів машинного навчання. Для успішного проєкту глибокого навчання обов'язковим є ретельний збір даних і базових міток істини, за допомогою яких можна тренувати й тестувати модель, але отримання високоякісних позначених даних може бути дорогим і тривалим. Доступні дані, як правило, поділяються на три набори: навчальний, валідаційний і тестовий набір, хоча є деякі варіанти, наприклад, перехресна перевірка.

Для навчання мережі використовується навчальний набір, де значення втрат обчислюються за допомогою прямого поширення, а параметри, які можна вивчати, оновлюються за допомогою зворотного поширення.

Набір перевірки використовується для оцінки моделі під час процесу навчання, точного налаштування гіперпараметрів і виконання вибору моделі. Тестовий набір в ідеалі використовується лише один раз в самому кінці проекту, щоб оцінити продуктивність моделі, яка була точно налаштована та обрана у процесі навчання з наборами навчання та перевірки. Потрібні окремі набори перевірки та тестування, оскільки навчання моделі завжди передбачає тонке налаштування її гіперпараметрів і виконання вибору моделі. Оскільки цей процес виконується на основі продуктивності набору перевірки, деяка інформація про цей набір перевірки просочується в саму модель, тобто переналаштовується на набір перевірки, навіть якщо модель ніколи безпосередньо не навчається на ньому для параметрів, які можна вивчати. З цієї причини гарантується, що модель із точно налаштованими гіперпараметрами у наборі перевірки буде добре працювати на цьому самому наборі перевірки.

Перенавчання моделі (Overfitting)

Перенавчання відноситься до ситуації, коли модель вивчає статистичні закономірності, характерні для навчального набору, тобто в кінцевому підсумку запам'ятовує невідповідний шум замість того, щоб вивчати сигнал, і, отже, працює менш добре на наступному новому наборі даних.

Це одна з головних проблем машинного навчання, оскільки перенавчену модель неможливо узагальнити на нові дані, які ми ніколи не бачили.

У цьому сенсі тестовий набір відіграє ключову роль у належній оцінці продуктивності моделей машинного навчання. Звичайна перевірка на розпізнавання перенавчання навчальним даним полягає в моніторингу втрат і точності на наборах навчання та перевірки. Якщо модель добре працює на навчальному наборі порівняно з набором перевірки, то, ймовірно, модель була переповнена навчальними даними.

Найкраще рішення для зменшення перенавчання – отримати більше даних про тренування. Модель, навчена на більшому наборі даних, як правило, дає кращі резутати. Інші рішення включають регуляризацію зі зменшенням ваги, нормалізацію пакетів і збільшення даних, а також зниження архітектурної складності.

Dropout — це методика регуляризації, коли випадково вибрані активації встановлюються в 0 під час навчання, так, що модель стає менш чутливою до певних ваг у мережі [2].

Зниження ваги, яке також називають L2 регуляризацією, зменшує перенавчання, штрафуючи ваги моделі, щоб ваги приймали лише невеликі значення. Пакетна нормалізація адаптивно нормалізує вхідні значення наступного шару, пом’якшуючи ризик перенавчання, покращуючи градієнтний потік через мережу, забезпечуючи більш високу швидкість навчання та зменшуючи залежність від ініціалізації [3]. Збільшення даних також ефективне для зменшення перенавчання, що є процесом модифікації навчальних даних за допомогою випадкових перетворень, так що модель не бачитиме однакових вхідних даних під час навчальної ітерації [4]. Звіт про продуктивність остаточної моделі на окремому (невидимому) наборі тестів, в ідеалі – на зовнішніх наборах даних перевірки, якщо це можливо, є вирішальним для перевірки узагальнюваності моделі.

Набір даних Fashion-MNIST

Набір даних Fashion-MNIST — це набір даних зображень Zalando із зображеннями у відтінках сірого розміром 28x28, це каталог 70 000 товарів із 10 категорій та 7 000 зображень у кожній категорії. Навчальний набір містить 60 000 зображень, тестовий набір – 10 000 зображень.

Fashion-MNIST схожий на набір даних MNIST, який використовується для класифікації рукописних цифр. Це означає, що розміри зображення, навчальні та тестові розділи подібні до набору даних MNIST. Набір даних Fashion-MNIST можна завантажити за допомогою спеціальних модулів TensorFlow і Keras. Набір даних зображень Fashion-MNIST також знаходиться за посиланням https://arxiv.org/abs/1708.07747

Завдання до комп'ютерного практикуму 3

Потрібно використати бібліотеки мови програмування Python для завантаження, дослідження та аналізу набору даних Fashion-MNIST для задачі класифікації зображень. Після цього потрібно провести попередню обробку даних: змінити розміри, масштабувати, перетворити мітки у вектори кодування та розділити дані на навчальні та тестові набори. Зробивши все це, потрібно побудувати модель нейронної мережі CNN. Далі потрібно компілювати, навчити та оцінити побудовану модель, візуалізуючи точність і графіки втрат.

Потрібно забезпечити уникнення перенавчання моделі, переглянути свою початкову модель і повторно навчити її. Потрібно оцінити нову модель і порівняти результати обох моделей. Зробіть прогнози на основі даних тесту, перетворіть ймовірності у мітки класів і побудуйте кілька тестових зразків, які модель правильно та неправильно класифікувала. Нарешті, потрібно візуалізувати звіт про класифікацію, який дасть вам більш глибоку інформацію про те, який клас був (не)правильно класифікований моделлю. Дайте відповіді на всі питання, поставені у ході виконання практикуму.

Завантажте дані

Keras постачається з бібліотекою під назвою **datasets**, яку можна використовувати для завантаження наборів даних «із коробки»: ви завантажуєте дані з сервера і прискорюєте процес, оскільки вам більше не потрібно завантажувати дані на свій комп'ютер.

Зображення тренувального та тестувального наборів даних разом із мітками завантажуються та зберігаються у змінних *train_X*, *train_Y*, *test_X*, *test_Y*, відповідно.

```
from keras.datasets import fashion_mnist
(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()
```

Проаналізуйте дані

Проаналізуте, як виглядають зображення у наборі даних. Навіть якщо ви вже знаєте розміри зображень, все одно варто докладати зусиль, щоб проаналізувати їх програмно: можливо, доведеться змінити масштаб пікселів та розмір зображень.

```
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

Який результат ви отримали?

Які зображення є даними для навчання та які є тестовими даними?

Який розмір навчальних та тестових зразків?

Знайдіть унікальні номери міток тренувального набору.

```
classes = np.unique(train_Y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

Який результат ви отримали?

Є десять вихідних класів від 0 до 9.

Прогляньте зображення у наборі даних:

```
plt.figure(figsize=[5,5])
# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))
# Display the first image in testing data
```

```
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

Який результат ви отримали?

Результати двох вищевказаних графіків виглядають як ботильйони, і цьому класу присвоєно мітку класу 9. Аналогічно, інші продукти будуть мати різні мітки, але подібні продукти будуть мати однакові мітки. Це означає, що усі 7000 зображень ботильйонів будуть мати мітку класу 9.

Попередня обробка даних

Як ви могли бачити на графіку вище, у наборі даних використовуються зображення у відтінках сірого і мають значення пікселів у діапазоні від 0 до 255. Крім того, ці зображення мають розмір 28x28. Вам потрібно попередньо обробити дані, перш ніж ви вводити їх у модель.

Як перший крок, перетворіть кожне зображення тренувального і тестового набору розміром 28x28 у матрицю розміром 28x28x1, яка подається в мережу.

```
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X.shape, test_X.shape
```

Який результат ви отримали?

Дані зараз у форматі int8, тому перед подачею їх у мережу вам потрібно перетворити їх тип на float32, а також змінити масштаб значень пікселів у діапазоні від 0 до 1 включно.

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
```

Тепер вам потрібно перетворити мітки класів у вектор одноразового кодування. При одноразовому кодуванні ви перетворюєте категоріальні дані у вектор чисел. Причина, чому ви перетворюєте категоріальні дані в кодування, полягає в тому, що алгоритми машинного навчання не можуть працювати з

категоріальними даними безпосередньо. Ви створюєте один логічний стовпець для кожної категорії або класу. Тільки один із цих стовпців міг приймати значення 1 для кожного зразка. Звідси і термін одноразове кодування.

Одне кодування буде вектором-рядком, і для кожного зображення воно матиме розмір 1x10. Тут важливо зазначити, що вектор складається з усіх нулів, крім класу, який він представляє, і для цього це 1. Наприклад, зображення ботильйонів, яке ви отримали вище, має мітку 9, тому для всіх зображень ботинок один вектор кодування буде [0 0 0 0 0 0 0 0 1 0].

Перетворіть навчальні та тестові мітки у вектори кодування:

```
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)
# Display the change for category label using one-hot encoding
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

Який результат ви отримали?

У машинному навчанні або будь-якій задачі, що стосується даних, ви повинні правильно розділити дані. Щоб модель добре узагальнювала дані, ви повинні розділит навчальні дані на дві частини, одну призначену для навчання, а іншу для перевірки. У цьому випадку ви навчатимете модель на 80% навчальних даних і перевіряєте її на 20% решти даних. Це також допоможе зменшити перенавчанання, оскільки ви будете перевіряти модель на даних, які вона не побачила б на етапі навчання, що допоможе підвищити ефективність тестування.

```
from sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label = train_test_split(train_X,
train_Y_one_hot, test_size=0.2, random_state=13)
```

Перевірте форму набору для навчання та перевірки.

```
train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
```

Який результат ви отримали?

Нейронна мережа

Зображення мають розмір 28x28. Ви перетворюєте матрицю зображень у масив, змінюєте її масштаб між 0 і 1, змінюєте її так, щоб вона мала розмір 28x 28 x1, і передаєте її як вхідні дані в мережу.

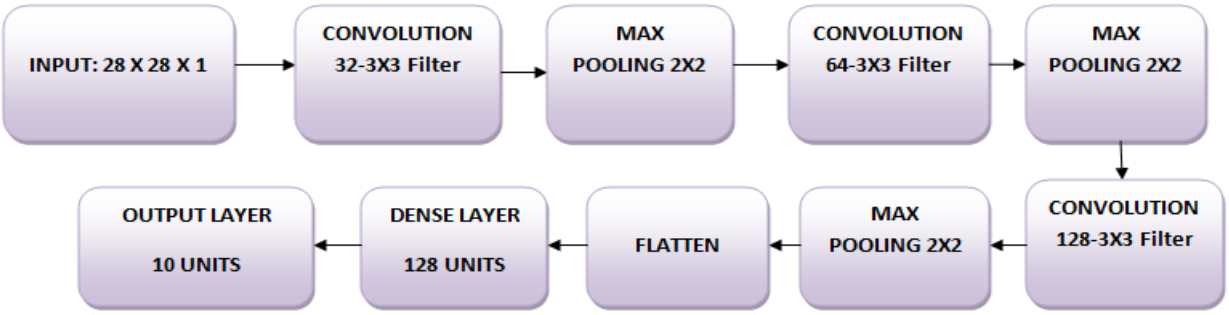
Потрібно використовувати три згорткові шари:

Перший шар матиме 32- 3x3 фільтри,

Другий шар матиме 64- 3x3 фільтри

Третій шар матиме фільтри 128-3x3.

Крім того, є три шари максимального об'єднання розміром 2x2.



Змоделюйте дані

Спочатку імпортуйте всі необхідні модулі, необхідні для навчання моделі.

```
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
```

Ви будете використовувати пакетний розмір 64, використовуючи більший розмір пакету 128 або 256, також краще, все залежить від пам'яті. Це робить значний внесок у визначення параметрів навчання та впливає на точність передбачення. Навчайте мережу протягом 20 епох.

```
batch_size = 64
epochs = 20
num_classes = 10
```

Архітектура нейронної мережі

У Keras можна складати шари, додаючи потрібний шар один за одним. Спочатку ви додасте перший згортковий шар за допомогою Conv2D().

Зауважте, що ви використовуєте цю функцію, оскільки працюєте із зображеннями. Далі потрібно додати функцію активації Leaky ReLU, яка допомагає мережі вивчати межі нелінійних рішень. Оскільки у вас є десять різних класів, вам знадобиться нелінійна межа рішення, яка могла б розділяти ці десять класів, які не є лінійно розділеними.

Функція активації ReLU часто використовується в архітектурах нейронних мереж, а точніше, в згорткових мережах, де вона виявилася більш ефективною, ніж широко використовувана логістична сигмовидна функція.

Станом на 2017 рік ця функція активації була найпопулярнішою для глибоких нейронних мереж. Функція ReLU дозволяє обмежити активацію нульовим порогом. Проте під час навчання підрозділи ReLU можуть «загинути». Це може статися, коли через нейрон ReLU проходить великий градієнт: це може призвести до оновлення ваг таким чином, що нейрон більше ніколи не активується у жодній точці даних. Якщо це станеться, то градієнт, що протікає через одиницю, назавжди дорівнюватиме нулю з цієї точки. Leaky ReLU намагаються вирішити це: функція не буде нульовою, а матиме невеликий негативний нахил.

Далі додайте шар максимального об’єднання за допомогою MaxPooling2D() і так далі. Останній шар – це щільний шар, який має функцію активації softmax з 10 одиницями, яка необхідна для цієї задачі класифікації з багатьма класами.

```
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3,3),
activation='linear',input_shape=(28,28,1),padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
```

```
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

Компіляція моделі

Після створення моделі компілюйте її за допомогою оптимізатора Adam, одного з найпопулярніших алгоритмів оптимізації. Вкажіть тип втрат, який є категоріальною перехресною ентропією, що використовується для багатокласової класифікації, також можна використовувати двійкову перехресну ентропію як функцію втрат. Вкажіть метрику як точність, яку хочете проаналізувати під час навчання моделі.

```
fashion_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
```

Вконайте візуалізацію шарів, які було створено на вищезгаданому кроці за допомогою функції підсумовування. Це покаже деякі параметри (ваги та зміщення) у кожному шарі, а також загальні параметри моделі.

```
fashion_model.summary()
```

Який результат ви отримали?

Тренування моделі

Настав час тренувати модель з функцією Keras **fit().** Модель тренується протягом 20 епох. Функція **fit()** поверне об’єкт історії; зберігаючи результат цієї функції в fashion_train, ви можете використовувати його пізніше, щоб побудувати графіки точності та функції втрат між навчанням і перевіркою, що допоможе візуально проаналізувати продуктивність моделі.

```
fashion_train = fashion_model.fit(train_X, train_label,
batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,
valid_label))
```

Який результат ви отримали?

Ви навчали модель на наборі даних fashion-MNIST протягом 20 епох, і, спостерігаючи за точністю навчання та втратами, можна сказати, що модель добре попрацювала, оскільки після 20 епох точність навчання становить близько 99%, а втрата навчання досить низька.

Однак схоже, що модель перенавчена, оскільки втрата підтвердження становить 0,4396, а точність перевірки становить 92%. Перенавчання означає, що мережа дуже добре запам’ятала навчальні дані, але не гарантує, що вона працюватиме з невидимими даними, і саме тому існує різниця в точності навчання та перевірки. Модель можна зробити ефективнішою, додавши шар Dropout в мережу і залишивши всі інші шари без змін. Але спочатку потрібно оцінити продуктивність моделі на тестовому наборі, перш ніж зробити висновок.

Оцінка моделі на тестовому наборі

```
test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=0)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

Який результат ви отримали?

Чи ці результати хороші?

Розгляньте модель оцінки в перспективі та побудуйте графіки точності та втрат між даними навчання та перевірки:

```
accuracy = fashion_train.history['acc']
val_accuracy = fashion_train.history['val_acc']
loss = fashion_train.history['loss']
val_loss = fashion_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
```

```
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Який результат ви отримали?

З отриманих двох графіків можна побачити, що точність підтвердження майже стагнувала через 4-5 епохи і рідко збільшувалася в певні епохи.

Спочатку точність перевірки лінійно збільшувалася зі втратами, але потім не сильно зросла. Втрата перевірки показує, що це ознака перенавчання, подібно до точності перевірки, лінійно зменшувалася, але через 4-5 епох почала збільшуватися. Це означає, що модель спробувала запам’ятати дані і досягла успіху. Маючи це на увазі, настав час ввести деяку кількість випадів у нашу модель і подивитися, чи допоможе це зменшити перенавчання.

Додавання Dropout до нейронної мережі

Ви можете додати вимикаючий шар, щоб певною мірою подолати проблему перенавчання. Dropout випадковим чином вимикає частину нейронів під час навчального процесу, зменшуючи залежність від навчального набору на деяку кількість даних. Скільки фракцій нейронів потрібно вимкнути, визначається гіперпараметром, який можна налаштувати. Таким чином, вимкнення деяких нейронів не дозволить мережі запам’ятовувати навчальні дані, оскільки не всі нейрони будуть активними одночасно, а неактивні нейрони не зможуть нічому навчитися.

Тож знову створіть, компілюйте та навчайте мережу, але цього разу з використанням Dropout. І запустіть його протягом 20 епох із розміром партії 64.

```
batch_size = 64
epochs = 20
num_classes = 10
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3,
3),activation='linear',padding='same',input_shape=(28,28,1)))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
fashion_model.summary()
```

Який результат ви отримали?

```
fashion_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
fashion_train_dropout = fashion_model.fit(train_X, train_label,
batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,
valid_label))
```

Який результат ви отримали?

Збережіть модель, щоб безпосередньо завантажувати її і не тренувати її знову протягом 20 епох. Таким чином, ви можете завантажити модель пізніше, якщо вона вам знадобиться, і змінити архітектуру; крім того, ви можете почати процес навчання на цій збереженій моделі. Завжди корисно зберегти модель – і навіть її вагу, оскільки це економить ваш час.

Зауважте, що ви також можете зберігати модель після кожної епохи, щоб у випадку, якщо виникла якась проблема, яка зупиняє навчання в епоху, вам не довелось починати навчання з початку.

```
fashion_model.save("fashion_model_dropout.h5py")
```

Оцінка моделі на тестовому наборі

Оцініть нову модель і подивіться, як вона працює.

```
test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=1)
```

Який результат ви отримали?

```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

Який результат ви отримали?

Додавання Dropout в модель повинно спрацювати, хоча точність тесту значно не покращилася, але втрата тесту зменшилася порівняно з попередніми результатами. В останній раз побудуйте графіки точності та втрат між даними навчання та перевірки.

```
accuracy = fashion_train_dropout.history['acc']
val_accuracy = fashion_train_dropout.history['val_acc']
loss = fashion_train_dropout.history['loss']
val_loss = fashion_train_dropout.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
```

```
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Який результат ви отримали?

Ви можете побачити, що втрата перевірки та точність перевірки синхронізовані з втратою та точністю навчання. Незважаючи на те, що втрата перевірки та лінія точності не є лінійною, вона показує, що модель не перенавчена: втрата перевірки зменшується, а не збільшується, і немає великого розриву між навчанням і точністю перевірки.

Таким чином, здатність моделі до узагальнення стала набагато кращою, оскільки втрата як тестового набору, так і набору перевірки була лише трохи більше в порівнянні з втратою навчання.

Прогнозування міток

```
predicted_classes = fashion_model.predict(test_X)
```

Оскільки отримані прогнози є значеннями з плаваючою крапкою, порівняти передбачені мітки з істинними тестовими мітками буде неможливо. Таким чином, потрібно заокруглити результат, який перетворить значення з плаваючою крапкою в ціле число. Потрібно використати np.argmax(), щоб вибрати номер індексу, який має вище значення в рядку.

Наприклад, припустимо, що для одного тестового зображення передбачено значення 0 1 0 0 0 0 0 0 0 0, вихід для цього має бути міткою класу 1.

```
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
predicted_classes.shape, test_Y.shape
```

Який результат ви отримали?

```
correct = np.where(predicted_classes==test_Y)[0]
print "Found %d correct labels" % len(correct)
for i, correct in enumerate(correct[:9]):
plt.subplot(3,3,i+1)
plt.imshow(test_X[correct].reshape(28,28), cmap='gray',
interpolation='none')
plt.title("Predicted {}, Class {}".format(predicted_classes[correct],
test_Y[correct]))
plt.tight_layout()
```

Який результат ви отримали?

```
incorrect = np.where(predicted_classes!=test_Y)[0]
print "Found %d incorrect labels" % len(incorrect)
for i, incorrect in enumerate(incorrect[:9]):
plt.subplot(3,3,i+1)
plt.imshow(test_X[incorrect].reshape(28,28), cmap='gray',
interpolation='none')
plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect],
test_Y[incorrect]))
plt.tight_layout()
```

Який результат ви отримали?

Подивившись на кілька зображень, ми не можемо бути впевнені, чому модель не в змозі правильно класифікувати наведені вище зображення, але різноманітність подібних шаблонів, наявних у кількох класах, впливає на продуктивність класифікатора CNN. Наприклад, зображення 5 і 6 належать до різних класів, але схожі, наприклад, піджак або, можливо, сорочка з довгим рукавом.

Звіт про класифікацію

Звіт про класифікацію допоможе більш детально визначити неправильно класифіковані класи. Ми можемо спостерігати, для якого класу модель показала погані результати з наведених десяти класів.

```
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i in range(num_classes)]
print(classification_report(test_Y, predicted_classes,
target_names=target_names))
```

Поясніть отриманий результат.

Ви можете помітити, що класифікатор недостатньо ефективний для класу 6 як щодо точності, так і щодо запам’ятовування. Для класу 0 і класу 2 класифікатору не вистачає точності. Крім того, для класу 4 класифікатору не вистачає точності та запам’ятовування.

Вимоги до оформлення звіту

Звіт має включати:

1. Титульний аркуш.
2. Завдання на комп’ютерний практикум.
3. Хід роботи. Цей розділ складається з послідовного опису виконуваних кроків згідно інструкцій до комп’ютерного практикуму.
4. Висновки.

Питання для самоперевірки

1. Назвіть основні складові базового штучного нейрону, які функції вони виконують?
2. Яким чином працює блок суматора, які функції він може використовувати?
3. Що означає термін "глибоке навчання"?
4. Дайте визначення згортковим нейронним мережам.
5. У яких реальних дослідженнях дослідженнях використовуються мережі CNN?

6. Яка архітектура згорткової нейронної мережі?
7. Що таке згортка нейронної мережі CNN?
8. Що таке параметри і гіперпараметри згорткової нейронної мережі?
9. Які функції активації ви знаєте?
10. Поясніть операцію максимального об'єднання в мережі CNN.
11. Як відбувається навчання нейронної мережі CNN?
12. Для чого використовуються алгоритм зворотного поширення та метод градієнтного спуску?
13. Що таке перенавчання моделі (Overfitting)? Як можна вирішити дану проблему?
14. Як реалізувати у Python навчання та тестування згорткової нейронної мережі?

Рекомендована література

1. LeCun Y., Bengio Y., Hinton G. (2015) Deep learning. Nature 521:436–444.
2. Hinton G.E., Srivastava N., Krizhevsky A., Sutskever I., Salakhutdinov R.R. (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv. Available online at: https://arxiv.org/pdf/1207.0580.pdf
3. Ioffe S., Szegedy C. (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv. Available online at: https://arxiv.org/pdf/1502.03167.pdf
4. Zhong Z., Zheng L., Kang G., Li S., Yang Y. (2017) Random erasing data augmentation. arXiv. Available online at: https://arxiv.org/pdf/1708.04896.pdf
5. Convolutional neural networks: an overview and application in radiology. Available online at: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9
6. A Beginner's Guide to Neural Networks and Deep Learning. Available online at: https://wiki.pathmind.com/neural-network
7. Neural Networks (NN). Available online at: https://www.w3schools.com/ai/ai_neural_networks.asp