Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка засобів гібридної візуалізації, класифікації, кластеризації та регресії в середовищі Wolfram Mathematica з використанням мови програмування Python

Виконав: студент 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) (прізвище та ініціали)

Керівник (підпис) Бойко І.В. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) . (прізвище та ініціали)

Тернопіль 2023

АНОТАЦІЯ

Атестаційна робота магістра на тему «Розробка засобів гібридної візуалізації, класифікації, кластеризації та регресії в середовищі Wolfram Mathematica з використанням мови програмування Python», Войтович Ростислав Вікторович.

Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–61, Тернопіль, 2023. С. – 106, рис. – 61, табл. – 2, додат. – 1, бібліог. – 18.

Метою атестаційної роботи магістра є розробка прямих та комбінованих методів візуалізації зображень, класифікації, кластеризації і регресії для масивів даних отримуваних в результаті аналізу складних систем та їх структури.

Технології розробки: Python, Wolfram Mathematica, Wolfram Cloud, Wolfram data analyst, MathCodeC++.

Технології для тестування: Python Unit testing, WolframUnit.

Розроблені методи спрямовані для фахівців з науки про дані та дата інжинірингу й дозволяють значно оптимізувати й збільшити ефективність їхньої роботи шляхом гібридного застосування в складних програних системах вбудованих процедур та функцій з близькоспоріднених мов Python та Wolfram Mathematica.

Ключові слова: гібридна візуалізація, кластеризація даних, інтерполяція даних, машинне навчання, регресія даних, Python, Wolfram Mathematica.

ABSTRACT

Master's certification work on the topic «Development of hybrid visualization, classification, clustering and regression tools in the Wolfram Mathematica environment using the Python programming language», Voytovych Rostyslav Viktorovych.

Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Department of Software Engineering, SPm-61 Academic Group, Ternopil, 2023. P. – 106, fig. – 61, table. – 2, append. – 1, bibliog. – 18.

The goal of the master's certification work is to develop direct and combined methods for image visualization, classification, clustering and regression for data sets obtained as a result of the analysis of complex systems and their structure..

Development technologies: Python, Wolfram Mathematica, Wolfram Cloud, Wolfram data analyst, MathCodeC++.

Technologies for testing: Python Unit testing, WolframUnit.

The developed methods are aimed at specialists in data science and data engineering and can significantly optimize and increase the efficiency of their work through the hybrid use of built-in procedures and functions from closely related programming languages Python and Wolfram Mathematica in complex software systems.

Keywords: hybrid visualization, data clustering, data interpolation, machine learning, data regression, Python, Wolfram Mathematica.

ЗМІСТ

1. ВСТУП. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ПРОБЛЕМИ

Комп'ютерний зір — це область штучного інтелекту, яка дозволяє розуміти зміст цифрових зображень, як-от фотографії. В даний час машинне навчання робить вражаючі успіхи у вирішенні завдань, пов'язаних із комп'ютерним зором, обіцяючи подальші вражаючі успіхи.

Говорячи про комп'ютерний зір, часто з'являються два типи підходів у розробці методів: (i) довідкові методології, написані експертами, які часто є вченими, орієнтованими на студентів та практиків, та (ii) методології, орієнтовані на програмування (тобто книги-ігри), написані експертами. які часто є розробниками та інженерами та призначені для використання практиками-програмістами як допоміжні матеріали. У той час як перші в основному зосереджуються на загальних методах та теорії (математика), а не на практичних аспектах проблем та застосуванні методів (код), другі фокусуються головним чином на методах та практичних проблемах вирішення проблем, де основна увага приділяється розміщенню прикладу коду та стандартних бібліотек.

Хоча методології, засновані на прикладних програмах, коротко описують методи з відповідною теорією (математика), вони, ймовірно, не привертають себе до уваги в плані використання як основного довідника. У цьому відношенні д-р Джейсон Браунлі, спеціаліст з машинного навчання, який вчить розробників, як отримувати результати за допомогою сучасних методів машинного навчання за допомогою практичних посібників, рекомендує роботу Річарда Селіски (2010; Комп'ютерний зір: алгоритми та програми), оскільки вона забезпечує короткий, цілеспрямований та підхід до розуміння того, що таке комп'ютерний зір, доповнюючи методи короткою теорією, не надаючи уваги значним подробицям. Програмістам він пропонує книгу Яна Еріка Солема (2012; Programming Computer Vision with Python), оскільки вона фокусується на реальних методах комп'ютерного зору з використанням стандартних (або досить близьких) бібліотек

Python. Це відмінна відправна точка для тих, хто займається новими технологіями щодо комп'ютерного зору.

Дана дипломна робота має своє цілю стати проміжною ланкою між згаданими двома типами методологій. З одного боку нами буде показано безліч різних методів, разом з великою кількістю прикладів щодо їх реалізації. Для кожного методу ми будемо зупинятися на детальному обговоренні математики, що лежить в основі різних методів. Коди алгоритмів дано як у Python, так і у формі Mathematica. У дипломній роботі версія Mathematica 11 сумісна з Python, більшість кодів об'єднані в гібридні коди, проте в для компіляції коду, що розміщений в додатках може використовуватися остання версія Mathematica 12, завантажена на Researchgate. Mathematica - неймовірно потужна платформа з цікавою та інтелектуальною мовою, але вона дорога і має закритий вихідний код. Python — зручна та потужна мова, що користується широкою підтримкою спільноти розробників. З того часу, як існують ці дві мови, люди намагалися зв'язати їх разом, щоб можна було використати інтегровані переваги обох мов і це є одним з ключових практичних рішень дипломної роботи.

Структура дипломної роботи є такою. Перша її частина присвячена методам зменшення розмірів візуальних об'єктів, де крім стандартних методів; подано аналіз незалежних компонентів, автоматичне кодування і фрактальний стиск. У другому розділі обговорюються методи класифікації, які включають класифікацію опорних векторів. Тут також досліджуються різні методи кластеризації. Продемонстровано як ієрархічна працює кластеризація, просторова кластеризація з шумом на основі щільності та спектральна кластеризація.. Виконано аналіз застосування нейронних мереж у комп'ютерному зору. Крім стандартних типів мереж, розроблено мережі глибокого навчання та згорткові мережі. Розвинуті методи порівнюються та кваліфікуються з різних практичних точок зору.

2. ЗАСОБИ ВІЗУАЛІЗАЦІЇ В PYTHON ТА MATHEMATICA.

2.1. Комп'ютерний зір та машинне навчання

Комп'ютерний зір (також відомий як машинний зір; Jain et al., 1995), міждисциплінарна область, яка у широкому значенні є підобластю штучного інтелекту та машинного навчання, має однією зі своїх цілей вилучення корисної інформації з зображень. Таким чином, основна проблема комп'ютерного зору полягає в тому, щоб спробувати зрозуміти, тобто побачити структуру реального світу із заданого набору зображень за допомогою спеціалізованих методів і загальних алгоритмів навчання (наприклад, Hartley and Zisserman 2003 [1-5]). Його застосування добре описано в Jähne and Haubecker (2000) [5-10], де воно знаходить застосування, наприклад, при захопленні руху людини (Moeslund and Granum 2001). Завдяки безлічі безпілотних літальних апаратів (БПЛА) або дронів (див. Awange, 2018; Awange та Kiema, 2019 [11-16]), комп'ютерний зір зміцнює свій авторитет у галузі БПЛА завдяки своїм інтелектуальним можливостям (Al-Kaff et al., 2018 [16-18]). Існує безліч публікацій з комп'ютерного зору, наприклад, алгоритмів обробки зображень (наприклад, Parker 2011, Al-Kaff et al., 2018 [17]), розпізнавання образів/мов у комп'ютерному зору (наприклад, Chen 2015), витягу ознак (Nixon and Aguado) 2012 [12, 15, 18]) та ін.

Зі свого боку, машинне навчання (ML) – це використання комп'ютерів статистичних методів для вивчення конкретних та складних завдань на основі заданих даних, які поділені на вивчені та певні класи (Anantrasirichai., 2018, 2019 [2-9]). Вони широко використовувалися, наприклад, для вивчення зсувів (Yilmaz, 2010), рослинності (Brown et al., 2008) [11], землетрусів (Adeli and Panakkat, 2009) [12], класифікації земної поверхні (Li, 2014) [13] та для класифікації вулканічних деформацій (Anantrasirichai., 2018, 2019), Ларі та ін. (2016) [13-16] добре пояснюють його застосування.

Зазвичай внесок комп'ютерного зору переважно групується у дві категорії, які фокусуються на методах та теорії, а не на практиці, та на основі

програмування, які фокусуються на методах та практичності вирішення проблем. Навряд чи знайдеться книга, яка б спробувала об'єднати ці два поняття; тобто методи/теорія, з одного боку, та методи/практичність (тобто код) з іншого боку. В даній дипломній роботі ми намагаємось заповнити цю прогалину, розглядаючи комп'ютерний зір як проблему машинного навчання (Рис. 1) та ігноруючи все, що ми знаємо про створення зображення. Наприклад, ми не використовуємо детальне розуміння перспективної проекції.

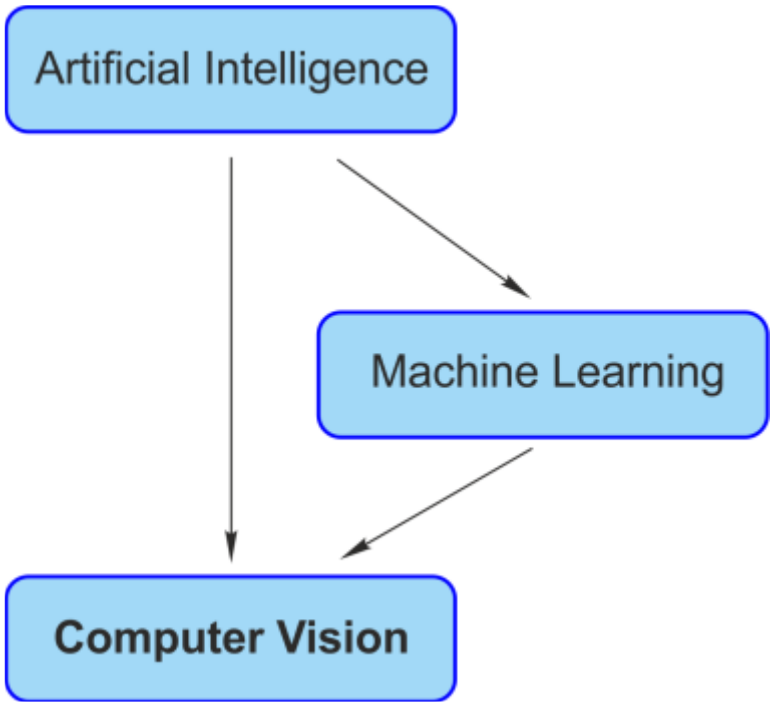


Рис. 2.1. Зв'язок між комп'ютерним зором, штучним інтелектом та машинним навчанням

Загалом ланцюжок обробки зображень містить п'ять різних завдань: повторна обробка, скорочення даних, сегментація, розпізнавання об'єктів та розуміння зображень. Методи оптимізації використовуються як набір допоміжних інструментів, доступних на всіх етапах ланцюжка обробки зображень, Рис. 2.1.

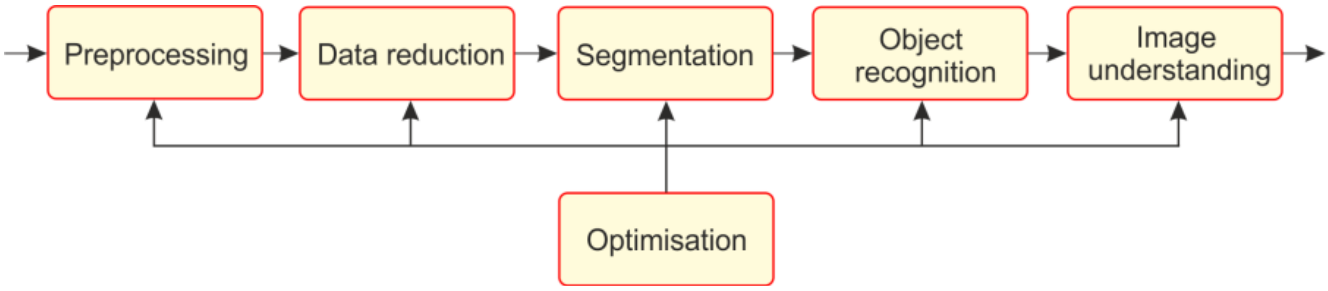


Рис. 2.2. Ланцюжок обробки зображень, що містить п'ять різних завдань.

Багато популярних програм комп'ютерного зору включають спроби розпізнавати об'єкти на фотографіях; наприклад:

1. Класифікація об'єктів. Яка широка категорія об'єктів зображена на цій фотографії?
2. Ідентифікація об'єкта. Який тип об'єкта зображено на цій фотографії?
3. Перевірка об'єкта: чи об'єкт знаходиться на фотографії?
4. Виявлення об'єктів. Де розташовані об'єкти на фотографії?
5. Виявлення орієнтирів об'єкта. Якими є ключові точки об'єкта на фотографії?
6. Сегментація об'єкта. Які пікселі належать об'єкту на зображенні?
7. Розпізнавання об'єктів. Які об'єкти зображені на цій фотографії та де вони?

Давайте розглянемо кілька прикладів, коли методи машинного навчання застосовуються на вирішення цих проблем комп'ютерного зору.

Приклад 1 (сегментація як кластеризація)

Сегментація – це будь-яка операція, яка розбиває зображення на області, узгоджені за певним критерієм. Одним із прикладів є поділ різних текстур. Нижче наведено зображення, де показані бактерії. Тепер нам хотілося б видалити фон, щоб отримати чітку інформацію про розмір та форму бактерій.

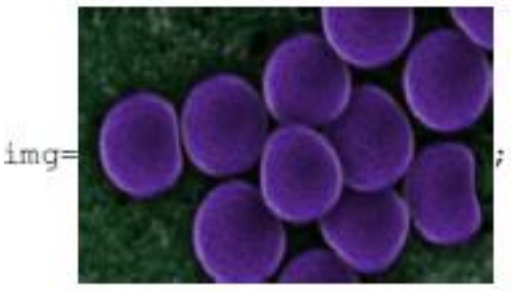


Рис. 2.3. Початкове зображення

```
RemoveBackground[img, {"Foreground", "Uniform"}]
```

Рис. 2.4. Зображення бактерій після видалення фону

Приклад 2 (Розпізнавання об'єктів як класифікація)

Виявлення та розпізнавання об'єктів визначають положення, а можливо, також орієнтацію та масштаб конкретних об'єктів на зображенні та класифікують їх. На зображенні нижче ми можемо отримати інформацію про тип об'єкта зображення (автомобіля) та можливу ймовірність підкласів.

```
img=
```

```
;
```

Рис. 2.5. Початкове зображення

```
⇒ data=ImageIdentifyn[img, car(WORD), 10, "Probability"]
⇐ <|convertible → 0.725076,saloon → 0.150854,coupe → 0.0760313,
    station wagon → 0.0414005,hatchback → 0.00342106,
    limousine → 0.00153371,automobile → 1.|>
```

Приклад 3 (розпізнавання зображень як виявлення орієнтирів)

Ключові точки зображення можуть характеризувати розташування основних елементів зображення. Цю інформацію можна використовувати для подальших операцій обробки зображень, таких як перетворення зображень, класифікація та кластеризація. Розглянемо зображення на Рис. 2.6 нижче.

Рис. 2.6 Початкове зображення

Знайдемо перші тридцять найважливіших ключових точок зображення.

```
HighlightImage[img, ImageKeypoints[img, "MaxFeatures" .-> 30]]
```

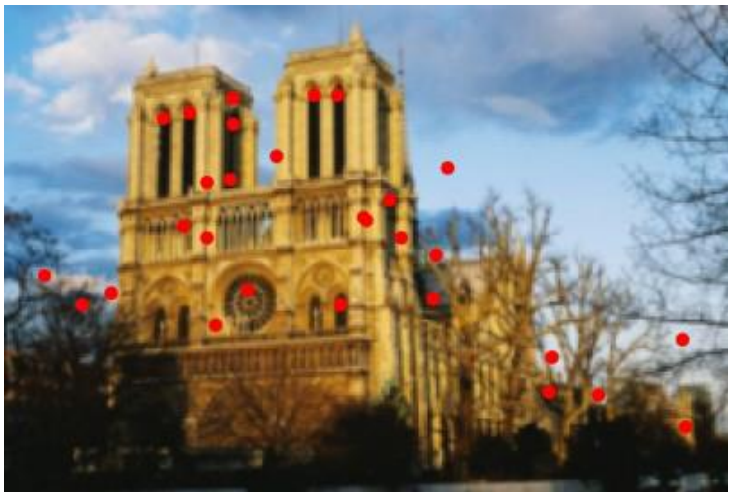


Рис. 2.7. Перші тридцять найважливіших ключових точок.

2.2. Python і Mathematica як засоби для гібридної візуалізації

Python в даний час, безсумнівно, є найпопулярнішою мовою для проектів з науки про дані, в той час як Wolfram є швидше нішевою мовою в цій сфері. Отже,

Python, ймовірно, добре відомий практикуючим програмістам в порівнянні з Mathematica. Враховуючи, що мова Wolfram Language, що широко використовується в академічних колах (особливо у фізиці, математиці та фінансовій аналітиці), існує вже понад 30 років, вона насправді старша за R і Python.

Загальний принцип мови Wolfram Language полягає у тому, що кожна функція має дуже високий рівень та максимально автоматизована. Наприклад, функція Classify[ ] автоматично вибирає метод для користувача. Однак користувач може встановити його вручну, наприклад, «Метод ➔ «RandomForest». Функція нейронної мережі, що використовується в Mathematica, використовує MxNet як серверну частину і аналогічна використанню Keras в Python, хоча її приємніше переглядати. Загалом функції машинного навчання (ML) у Wolfram нагадують чорну скриньку, хоча є й функції нижчого рівня. Тому не слід сліпо вірити, що автоматичні рішення, які надаються функціями прогнозування та класифікації, є єдиними оптимальними рішеннями. Найчастіше вони далекі від цього і в кращому разі дають базові рішення, на які можна покластися. Потім можна використовувати функції нижчого рівня для створення власного індивідуального рішення машинного навчання за допомогою Wolfram або Python. У Mathematica дуже хороша система документації з усіма вбудованими функціями. Також сама документація є в блокнотах, щоб можна було швидко спробувати щось прямо всередині документації. Документація в Mathematica дійсно хороша, але у Python набагато більша спільнота з широкою мережею підтримки, тому дуже ймовірно, що можна знайти відповідь на цю проблему. Крім того, можна багато чого навчитися на таких сайтах, як Kaggle.

Отже, поговоримо про слона в кімнаті: ціну. Mathematica не безкоштовна, насправді вона досить дорога. Оскільки Mathematica з самого початку постачається з усіма функціями, немає необхідності купувати додаткові набори інструментів, як у Matlab. Тепер ми представляємо деякі пункти для обох мов у довільному порядку.

Wolfram Mathematica

- переклад на природну мову
- зіставлення із зразком є потужним і помітним, наприклад, в оголошенні функції
- інтерактивна та дуже хороша документація
- послідовна
- символічна, у функцію можна передати все (має багато переваг, але ускладнює налагодження)
- більш просунуті ноутбуки
- ніяких віртуальних середовищ та залежностей
- працює однаково у всіх ОС
- у більшості випадків є лише один очевидний спосіб зробити щось, наприклад, побудова сюжету.
- динамічні та керуючі функції для більшої інтерактивності
- вбудовані знання
- індекси починаються з 1
- миттєвий інтерфейс прикладного програмування (API) (правда, тільки у Wolfram Cloud або у власному Wolfram Enterprise Cloud) - важко знайти роботу/важко набрати людей, які знають Wolfram

Python

- ближче до сучасного рівня техніки
- коди легше читати та підтримувати
- повідомлення налагодження зазвичай корисніші
- безкоштовний дистрибутив
- безліч можливостей для розгортання навченої моделі
- безліч онлайн-курсів, подкастів та інших ресурсів
- Використання google-colab або Kaggle для вивчення машинного навчання без локального графічного процесора.
- Pandas простіше використовувати, ніж "Набір даних" у системі Mathematica.
- більша спільнота, отже, легша підтримка.

Вивчення іншої мови зазвичай є корисним для загального розуміння програмування. Ми тут використовуємо мову Wolfram для швидкого прототипування ідей і часто вигадуємо цікаві комбінації даних або проектування функцій, маючи вбудовані знання мови Wolfram. Крім того, швидка маніпуляція допомагає краще зрозуміти проблему та дані.

У Mathematica лише одним рядком можна розгорнути нашу модель у вигляді інтерфейсу прикладного програмування (API) або веб-програми, але тільки в інфраструктурі Wolfram, яка може не вписатися у вашу інфраструктуру чи політику. Крім того, високорівневі функції Classify та Predict надто обмежені, і навіть стандартні алгоритми навчання scikit перевершують їх.

Загалом ми сподіваємося, що обидві мови надихають одна одну, оскільки блокнот Jupyter безперечно був натхненний Mathematica. З іншого боку, Wolfram матиме важке майбутнє, якщо вони продовжать намагатися робити все самостійно і прив'язувати користувачів до своєї інфраструктури. Тому поєднання двох мов у майбутньому буде дедалі плідніше.

3. КЛАСИФІКАЦІЯ. РОЗРОБКА МЕТОДІВ КЛАСТЕРИЗАЦІЇ. РОЗРОБКА МЕТОДІВ РЕГРЕСІЇ

3.1. Класифікація найближчих сусідів

Класифікація KNearest Neighbours - це найпростіший алгоритм, заснований виключно на даних, який можна використовувати як задач класифікації, так задач регресії. У літературі з геолого-геофізичних наук вони відомі як багатокутники Вороного, а у чисельному моделюванні — як осередки Діріхле (Мюллер та Гвідо, 2017).

Щоб використовувати його для класифікації, існуючий набір прикладів даних позначається як навчальний набір, де для всіх даних відомий клас, до якого належить кожна частина даних. Щоразу, коли дається новий фрагмент даних без мітки, він порівнюється з позначеним існуючим фрагментом даних, а потім беруться найбільш схожі (найближчі заданою мірою) фрагменти даних (найближчі сусіди) і розглядаються їх мітки. Ми дивимося на *k* найбільш схожих фрагментів даних нашого відомого набору даних; звідси і береться *k* (*k* - ціле число, зазвичай менше 20). Нарешті, голосування більшості береться з *k* найбільш схожих фрагментів даних, де більшість стає новим класом, присвоєним даним, які нас попросили класифікувати, див. Рис. 3.1.

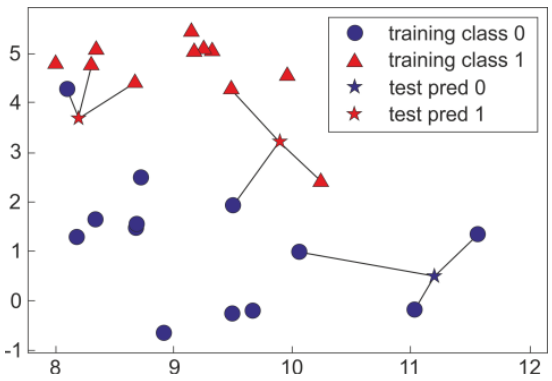


Рис. 3.1. Принцип класифікації найближчих сусідів у разі двох класів та використання k = 3 сусідів, згідно теорії Мюллера та Гвідо

Зазвичай *k*, який залежить від даних і присутності структур даних, є гіперпараметром, який слід налаштовувати адаптивно. Найпоширенішим підходом до пошуку оптимального значення *k* є використання процедури перехресної перевірки. При n-кратної перехресної перевірки вихідний набір навчальних даних розбивається на n підмножин. В окремому випадку (залишити одне спостереження) з вихідного набору даних як точка перевірки вибирається одне спостереження. Один з n-підмножини використовується як дані перевірки для тестування моделі, а інші *n*-1 підмножини використовуються як дані навчання. Помилка перевірки розраховується шляхом повторення процесу *n* разів, а усереднені помилки перевірки використовуються як єдина оцінка помилки перехресної перевірки для заданого параметра *k*. Процедура повторюється для різних значень параметра *k*, причому в якості оптимальної (*k*) вибирається модель, що забезпечує найменшу помилку перехресної перевірки.

Перевагами алгоритму є висока точність, нечутливість до випадіння значень та відсутність спеціальних припущень щодо даних. Однак це вимагає великих обчислювальних витрат та великої кількості пам'яті.

Невеликий набір даних

Спочатку розглянемо невеликий набір даних із пакета Python **mglearn**. Для Mathematica ми можемо записати дані у файл ASCII. Щоб використовувати Python у Mathematica, ми запускаємо сеанс Python,

```
⇒ session=
    StartExternalSession[<|"System" → "Python",
     "Version" → "3.5.4","Executable" →
      "C:\Users\Ben\AppData\Local\Programs\Python\Python35\
        python.exe"|>]//Quiet

⇐ ExternalSessionObject[
    SummaryPanel[ System: Python   EvaluationCount: None
                  UUID: 7b76966c2-ebb7-4ee8-bb25-02322d1456a8 ]]
```

```
import mglearn
import numpy as np
X, y = mglearn.datasets.make_forge()
np.savetxt('G:\\dataX.txt',X,fmt='%.5e')
```

Елементи даних, що підлягають класифікації, представлені двовимірними векторами.

```
⇒ X=Import["G:\\dataX.txt","Table"];
```

Однак, якщо файл короткий, ми не записуємо дані у файл. Позначення елементів таке:

```
> y
⇐ {1,0,1,0,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,1,0,0,0,0,1,0}
⇒ y=%;
```

Це означає, що ми маємо два класи. Візуалізуємо дані, див. Рис. 3.2.

```
⇒ class1=Pick[X,y,0];
⇒ class2=Pick[X,y,1];
⇒ p0=ListPlot[{class1,class2},PlotStyle → {Green,Red},Frame → True,
    Axes → None,PlotMarkers → {Automatic,Medium},AspectRatio → 1]
```

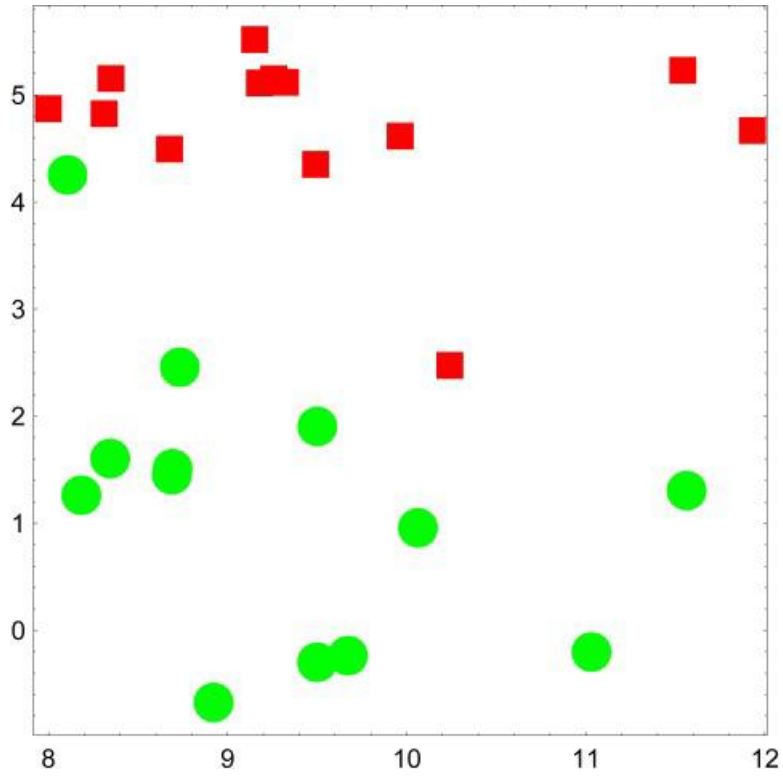


Рис. 3.2. Малий набір даних

Далі ми проведемо класифікацію за допомогою Mathematica.

Ми готуємо дані для Mathematica як Element→Label,

```
⇒ trainingData=Thread[X → y];
```

Спочатку ми використовуємо сусіда $k = 1$ з методом вичерпного пошуку з усього набору даних («Scan»).

```
⇒ c1=Classify[trainingData,Method → {"NearestNeighbors",
    "NeighborsNumber" → 1,"NearestMethod" → "Scan"}];
```

Результат можна побачити на Рис. 3.3.

```
⇒ p1=Show[{DensityPlot[c1[{u,v}],{u,7.5,12},{v,-1,6},
    ColorFunction → "CMYKColors"],p0}]
```

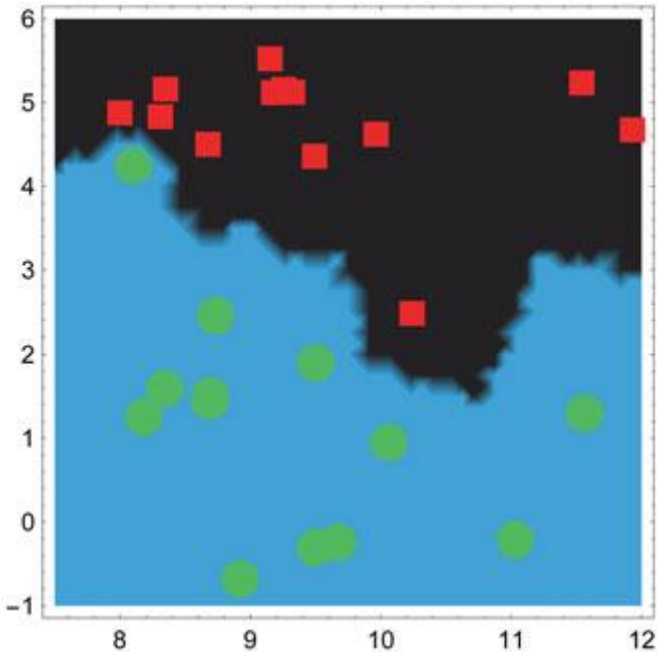


Рис. 3.3. Класифікація з $k = 1$

Дослідимо вплив кількості сусідів на якість класифікації. Розглянемо $k = 3$ і $k = 9$ сусідів.

```
⇒ c3=Classify[trainingData,Method → {"NearestNeighbors",
    "NeighborsNumber" → 3,"NearestMethod" → "Scan"}];
⇒ c9=Classify[trainingData,Method → {"NearestNeighbors",
    "NeighborsNumber" → 9,"NearestMethod" → "Scan"}];
⇒ p3=Show[{DensityPlot[c3[{u,v}],{u,7.5,12},{v,-1,6},
    ColorFunction → "CMYKColors"],p0}];
⇒ p9=Show[{DensityPlot[c9[{u,v}],{u,7.5,12},{v,-1,6},
    ColorFunction → "CMYKColors"],p0}];
⇒ GraphicsGrid[{{p3,p9}}]
```

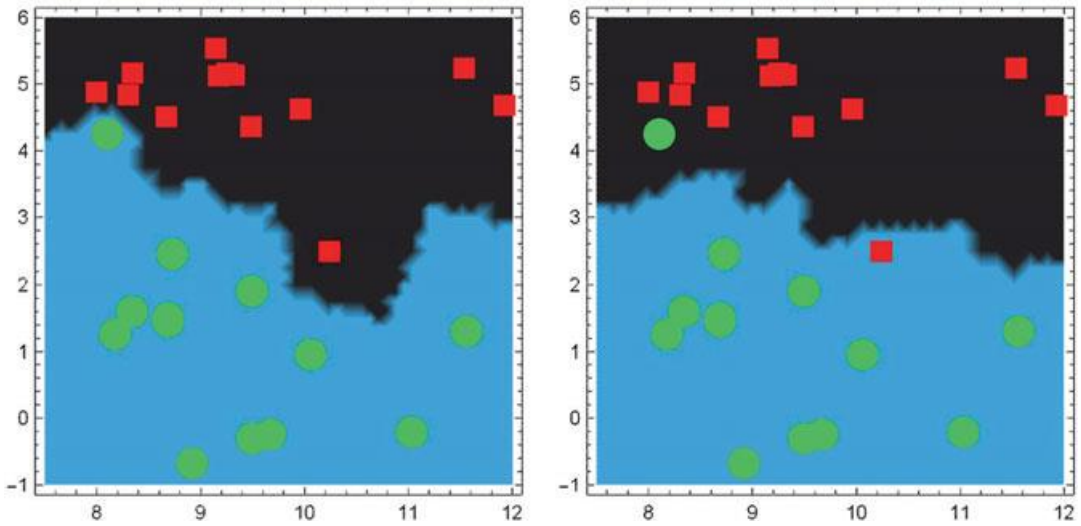


Рис. 3.4. Класифікація через найближчого сусіда $k = 3$ і $k = 9$

Ми можемо порівняти мітки елементів даних із мітками, наданими методом, використовуючи різну кількість сусідів. Найкращий результат дається при $k = 1$.

```
⇒ y1=Map[c1[#]&,X]
⇐ {1,0,1,0,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,1,0,0,0,0,1,0}

⇒ y3=Map[c3[#]&,X]
⇐ {1,0,1,0,1,1,1,0,1,1,1,0,0,0,1,1,1,0,0,1,0,0,0,0,1,0}

⇒ y9=Map[c9[#]&,X]
⇐ {1,0,1,0,1,1,1,0,1,1,1,0,0,0,1,1,1,0,0,1,0,0,0,0,1,0}
```

Норма помилок така:

```
⇒ Norm[y-y1]
⇐ 0

⇒ Norm[y-y3]
⇐ √2

⇒ Norm[y-y9]
⇐ √2
```

Цей результат свідчить про те, що k=1 є найкращим рішенням. Однак третій варіант (k=9) є більш надійним, оскільки різні елементи даних знаходяться далеко від межі двох класів, тому ця конфігурація не така чутлива до помилок вимірювання векторів ознак.

Давайте використаємо Python для k=1,

```
# from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors=1).fit(X,y)
prediction=clf.predict(X)
prediction
```

```
⇐ {1,0,1,0,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,1,0,0,0,0,1,0}
⇒ yP1=%;
```

Потім обчислюючи вектори помилок маємо:

```
⇒ Norm[y-yP1]
⇐ 0
```

Ми отримали той самий результат. У цьому прикладі використовувався лише навчальний набір. Тепер давайте скористаємося тестовим набором, щоб перевірити здатність нашого класифікатора до узагальнення. Це можна легко зробити на Python.

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test=
 train_test_split(X, y, random_state=0)
```

Загальний набір даних був випадково розділений.

```
X_train
⇐ {{8.9223,-0.639932},{8.73371,2.49162},{9.32298,5.09841},
   {7.99815,4.85251},{11.033,-0.168167},{9.17748,5.09283},
   {11.564,1.33894},{9.15072,5.49832},{8.3481,5.13416},
   {11.9303,4.64866},{8.10623,4.28696},{8.67495,4.47573},
   {9.67285,-0.202832},{9.50169,1.93825},{8.69289,1.54322},
   {9.96347,4.59677},{9.50049,-0.264303},{9.25694,5.13285},
   {8.68937,1.4871}}
⇒ Xtrain=%;
```

```
X_test
⇐ {{11.5416,5.21116},{10.0639,0.990781},{9.49123,4.33225},
   {8.18378,1.29564},{8.30989,4.80624},{10.2403,2.45544},
   {8.34469,1.63824}}
⇒ Xtest=%;
```

```
y_train
⇐ {0,0,1,1,0,1,0,1,1,1,0,1,0,0,0,1,0,1,0}
⇒ ytrain=%;
```

```
y_test
⇐ {1,0,1,0,1,1,0}
⇒ ytest=%;
```

Отже, тепер у нас є 19 елементів у навчальному наборі та 7 елементів у тестовому наборі.

Тепер, використовуючи навчальний набір, ми створюємо класифікатор із трьома сусідами.

```
⇒ trainingData=MapThread[#2 → #1&,{ytrain,Xtrain}];
⇒ ctrained=Classify[trainingData,Method → {"NearestNeighbors",
    "NeighborsNumber" → 3,"NearestMethod" → "Scan"}];
```

Тестування класифікатора на навчальних даних

```
⇒ trainingData=MapThread[#2 → #1&,{ytrain,Xtrain}];
⇒ ctraining=ClassifierMeasurements[ctrained,trainingData]
⇐ ClassifierMeasurementsObject[ Classifier: NearestNeighbors
                                Number of test examples: 19 ]
```

Точність класифікатора на навчальному наборі,

```
⇒ ctraining["Accuracy"]
⇐ 0.947368
```

Матриця неточності показує кількість помилково класифікованих елементів як недіагональних елементів як на Рис. 3.5.

```
⇒ ctraining["ConfusionMatrixPlot"]
```

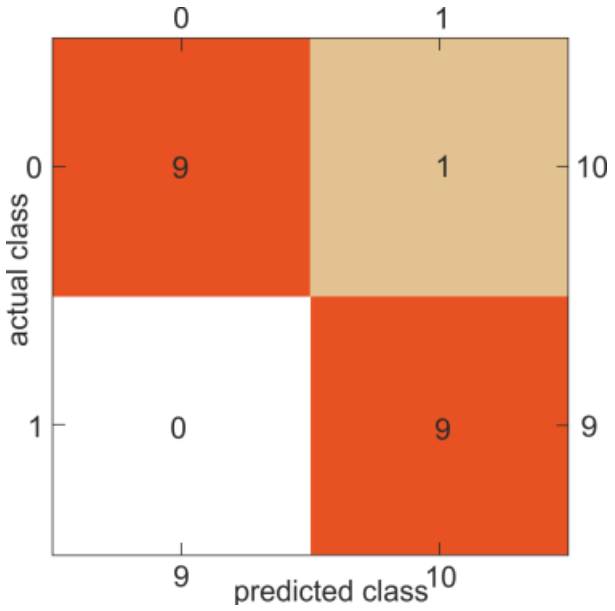


Рис. 3.5. Матриця неточності для навчального набору

Тепер подивимося ту ж статистику для тестового набору, який не брав участі у процесі навчання.

```
⇒ testingData=MapThread[#2 → #1&,{ytest,Xtest}];
⇒ ctesting=ClassifierMeasurements[ctrained,testingData]
⇐ ClassifierMeasurementsObject[ Classifier: NearestNeighbors
                                Number of test examples: 7 ]
```

Точність тестового набору,

```
⇒ ctesting["Accuracy"]
⇒ 0.857143
⇐ ctesting["ConfusionMatrixPlot"]
```

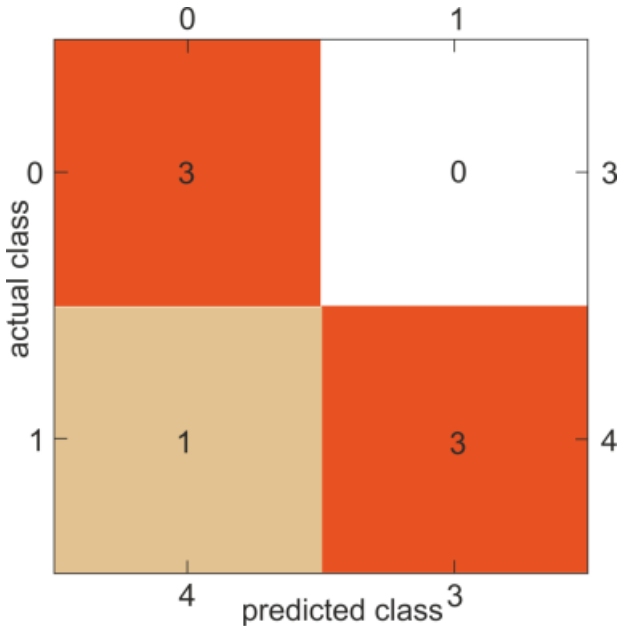


Рис. 3.6. Матриця неточності для тестового набору

3.2. Логістична регресія

Розглянемо елемент, представлений вектором $z_i = \{x_1, x_2, ..., x_m\}$ і помічений як y_i. Логістичну регресію можна розглядати як спеціальну дрібну нейронну мережу, де функція активації є сигмаподібною функцією, графік якої подано на Рис. 3.7.

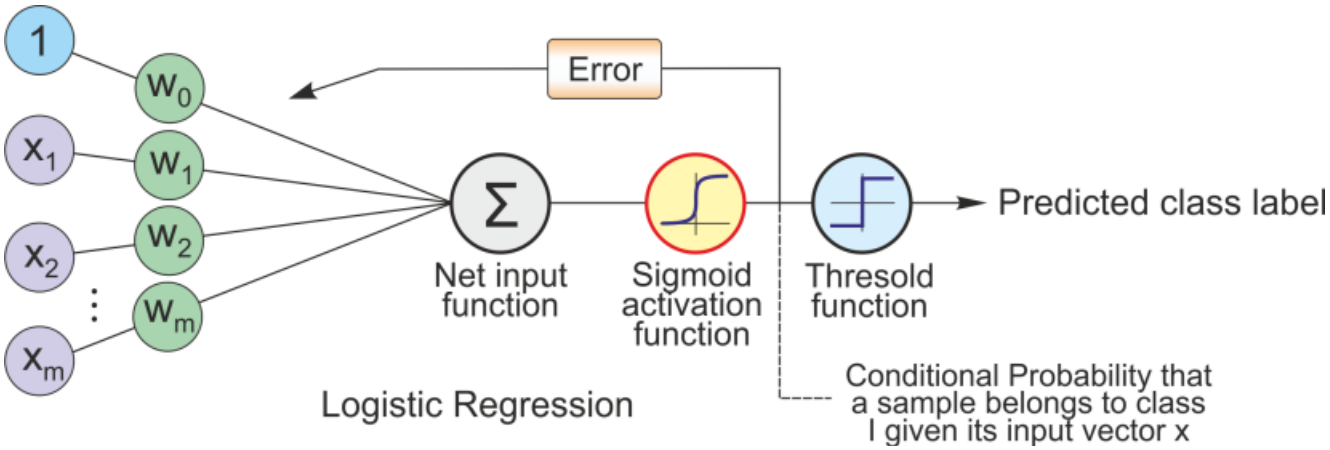


Рис. 3.7. Принцип логістичної регресії (згенеровано у Mathematica)

Вихідні дані сигмовидної функції можна інтерпретувати як ймовірність того, що конкретний зразок належить класу 1,

$$\phi(z_i) = \frac{1}{1+\mathrm{Exp}(-z)} = P\left(y=1 \middle| z:w\right),$$

враховуючи його вектор ознак z параметризований вагами w. Оскільки цей метод за своєю суттю є методом бінарної класифікації, іноді додається порогова функція (одинична або Хевісайда). Функція вартості – це логарифмічна функція правдоподібності, яку необхідно максимізувати для отримання оптимальної ваги,

$$\mathcal{L}(w) = \ln L(w) = \sum_{i=1}^{n}\left[y_i \ln\left(\phi(z_i)\right) + (1-y_i)\ln\left(1-\phi(z_i)\right)\right].$$

У Mathematica та Python цільова функція може бути розширена за допомогою умов регуляризації L1 та L2:

$$\mathcal{L}_{\mathcal{R}}(w) = \ln L(w) + \lambda_1 \sum_{i=1}^{n} |w_i| + \frac{\lambda_2}{2}\sum_{i=1}^{n} w_i^2,$$

де λ_i – параметри регуляризації (Prince, 2012).

Основна ідея хороша лише для бінарної класифікації. Однак існує можливість узагальнити бінарний метод, наприклад, на множину послідовних кроків. Техніка називається "один проти багатьох": припускаючи наявність k класів, ми починаємо з двох класів з мітками 1 та іншого з (2, 3, ... k), потім знову з двох класів з елементами, поміченими 2 та іншими (3,..k).

Переваги цього алгоритму у тому, що він вимагає великих обчислювальних витрат, простий у реалізації і дозволяє легко інтерпретувати отримані результати. Однак він часто має тенденцію мати низьку точність.

Спершу розглянемо підходи до класифікації Iris. Тепер у нас є 150 зразків, дві характеристики: довжина і ширина пелюстки, а також три цільові набори: Iris-setosa (0), Iris-versicolor (1) та Iris-virginica (2). Почнемо з Python.

Давайте зчитаємо елементи, які потрібно класифікувати (X) та їх мітки (y),

```
from sklearn import datasets
iris=datasets.load_iris()
```

```
X=iris.data[:,[2,3]]
y=iris.target
```

Збережемо X у файлі для Mathematica,

```
import numpy as np
```

```
np.savetxt('M:\\dataX.txt',X,fmt='%.2e')
```

Навчальний регресор Python,

```
from sklearn.linear_model  import LogisticRegression
lr=LogisticRegression(C=100.0,random_state=1).fit(X,y)
```

Результат, що дає Python такий:

```
lr.predict(X[:150,:])
```

```
⇐ {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,
   1,1,2,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
   2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,
   2,2,2,2,2,2,2,2,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2}

⇒ yP=%;
```

Оригінальні навчальні мітки,

```
y
```

```
⇐ {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
   1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
   2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,
   2,2,2,2,2,2,2,2,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2}

⇒ yTR=%;
```

Точність на навчальному наборі,

```
print("Training set score: {:.2f}".format(lr.score(X, y)))
```

```
Training set score: 0.96
```

Читання даних у Mathematica

```
Xtrain=Import["M:\\dataX.txt","Table"];
```

Повний навчальний набір

```
dataT=MapThread[Join[#1,{#2}]&,{Xtrain,yTR}];
```

Щоб візуалізувати три набори даних, ми ділимо набір на три класи:

```
data0=Map[{#[[1]],#[[2]]}&,Select[dataT,#[[3]]==0&]];
data1=Map[{#[[1]],#[[2]]}&,Select[dataT,#[[3]]==1&]];
data2=Map[{#[[1]],#[[2]]}&,Select[dataT,#[[3]]==2&]];
```

Далі на Рис. 3.8 показані елементи, що підлягають класифікації:

```
p0=ListPlot[{data0,data1,data2},PlotStyle → {Green,Blue,Red},
  Frame → True,Axes → None,PlotMarkers → {Automatic},
  AspectRatio → 0.9,FrameLabel → {"petallength","petal width"},
  Frame → True]
```

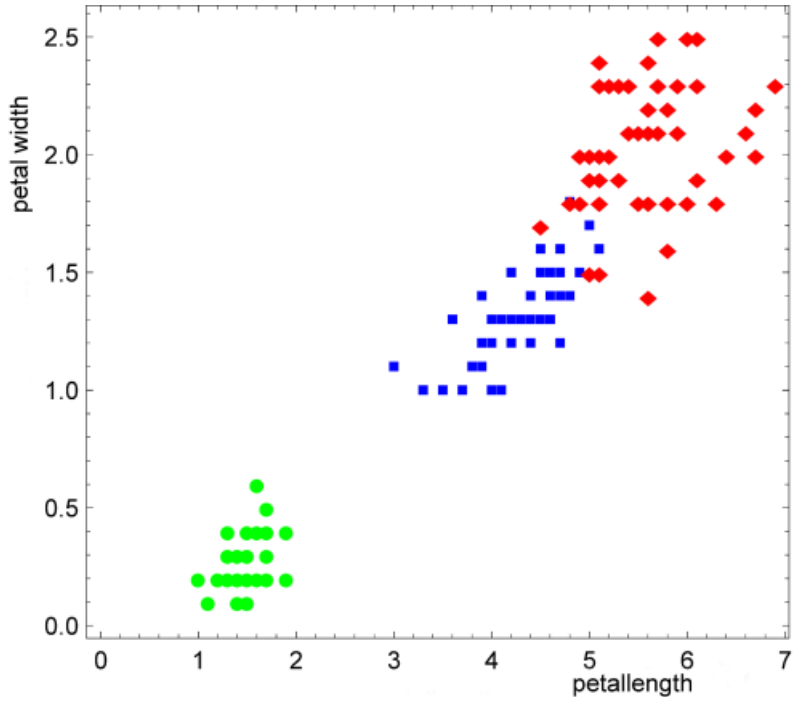


Рис. 3.8. Три класи набору даних Iris

Навчальний набір даних

```
dataTrain=Map[{#[[1]],#[[2]]} → #[[3]]&,dataT];
```

процес навчання буде такий:

```
⇒ c=Classify[dataTrain,Method → "LogisticRegression",
    PerformanceGoal → "Quality"];
```

Потім знаходимо точність класифікатора на навчальному наборі

```
⇐ ClassifierMeasurementsObject[ Classifier: LogisticRegression
                                Number of test examples: 150 ]
```

```
⇒ ctraining["Accuracy"]
⇐ 0.96
```

Рис. 3.9 демонструє матрицю неточності,

```
⇒ ctraining["ConfusionMatrixPlot"]
```

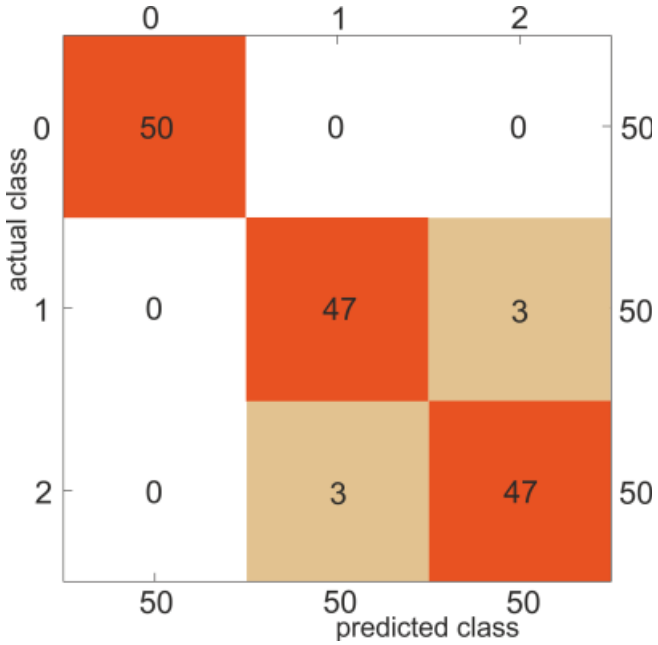


Рис. 3.9. Матриця неточності навчального набору

Давайте візуалізуємо досліджувані три класи,

```
⇒ Show[{DensityPlot[c[{u,v}],{u,0,7},{v,-1,4.5},
    ColorFunction → "CMYKColors",PlotPoints->50],p0},
    AspectRatio->1.2]
```

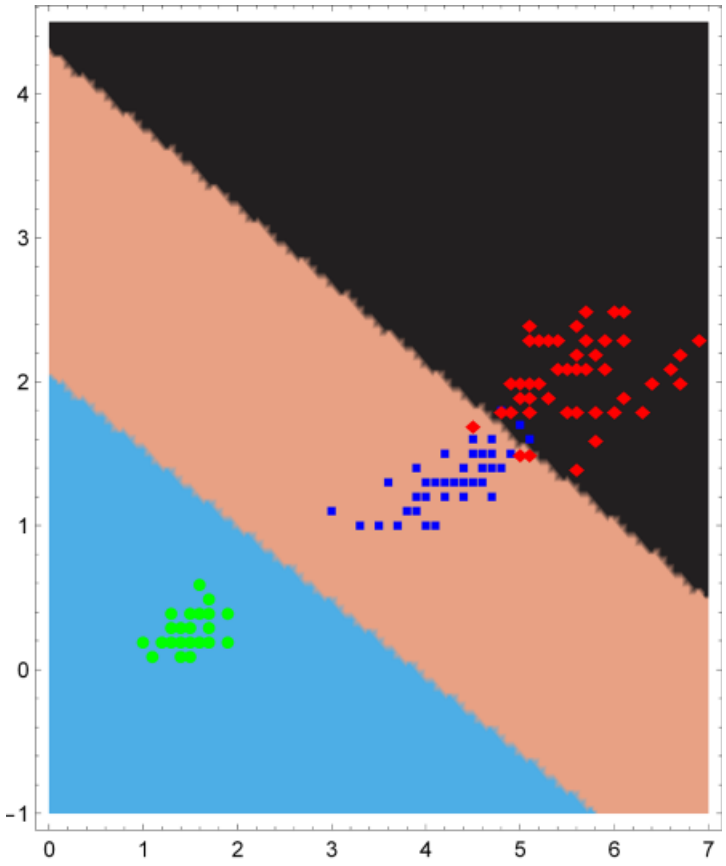


Рис. 3.10. Три класи набору даних Iris

Помилку класифікації можна охарактеризувати нормою неправильно класифікованих елементів:

```
⇒ yP=Map[c[#]&,Join[{data0,data1,data2}]]//Flatten;
⇒ Norm[yTR-yP]
⇐ √6
```

Таким чином обидва підходи приводять нас до однакових результатів із точності та ефективності.

3.3. Розпізнавання цифр

Цифри представлені у вигляді сірих зображень розміром 26×16 зі значеннями пікселів $\in[0,1]$. Наприклад, давайте розглянемо цифру 2,

```
⇒ image= 2 ;
⇒ dataD=ImageData[image];
⇒ Dimensions[dataD]
⇐ {20,16}
```

Візуалізуємо для прикладу цифру 2:

```
⇒ MatrixPlot[dataD]
```

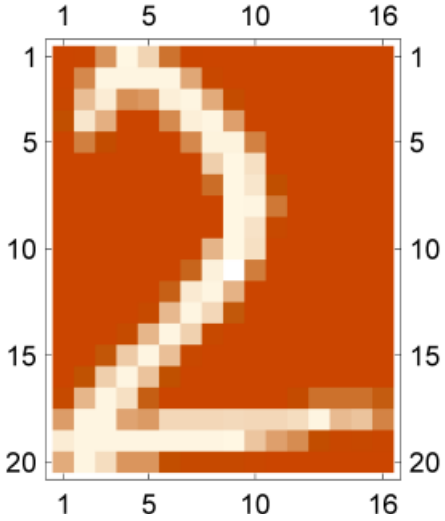
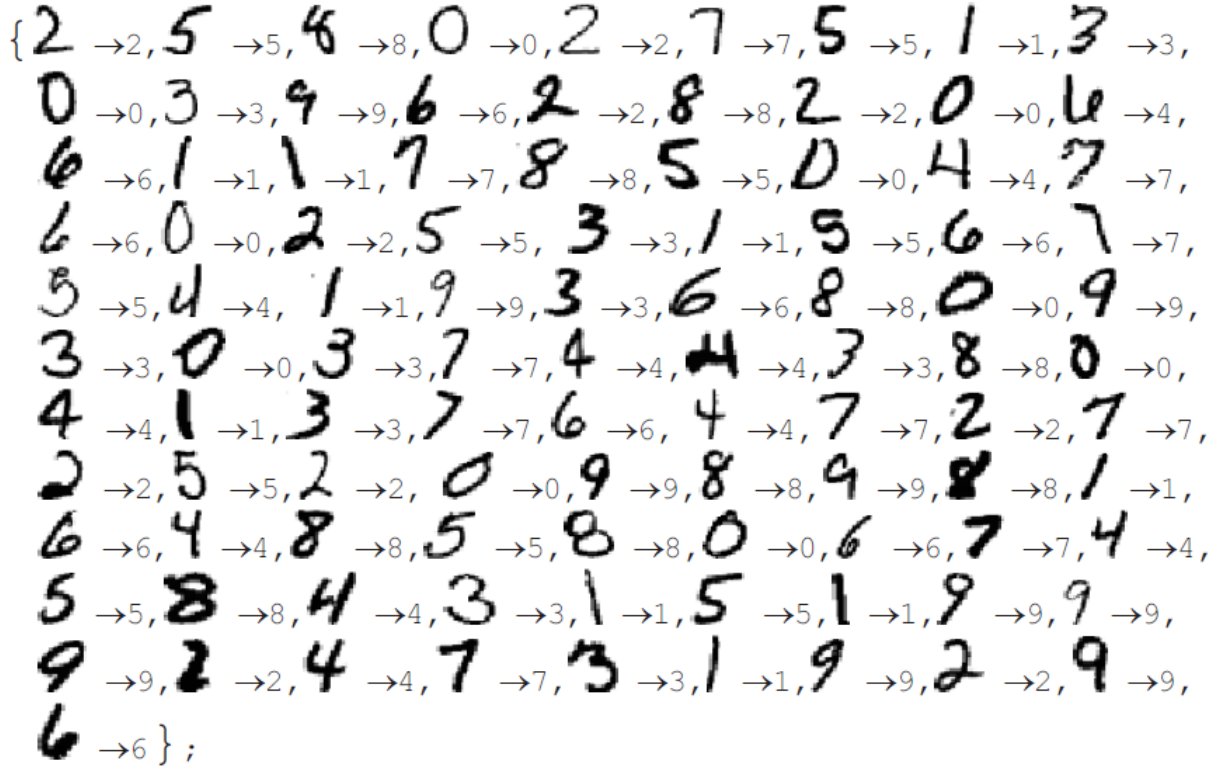


Рис. 3.11. Зображення цифри 2

Наш навчальний набір,

```
⇒ digitset=
```



який містить 100 елементів.

Тепер, щоб покращити здатність до узагальнення, крім тестового набору, ми вводимо набір перевірки. Набір перевірки буде включено до процесу навчання, проте його помилка просто відстежується і не впливає на зміну параметрів класифікатора. Цей метод може перешкоджати перенавчанню, як і регуляризація. Ми випадково вибираємо ці три набори.

```
⇒ RandomSeed[1234];
⇒ trainingset=RandomSample[digitset,70];
⇒ validationset=RandomSample[digitset,15];
⇒ testset=RandomSample[digitset,15];
```

Тепер виконуємо навчання з результатами набору перевірки,

```
⇒ logistic=Classify[trainingset,ValidationSet → validationset,
    Method → "LogisticRegression",PerformanceGoal → "Quality"]
⇐ ClassifierFunction[ Input type: Image
                      Number of classes: 10 ]
```

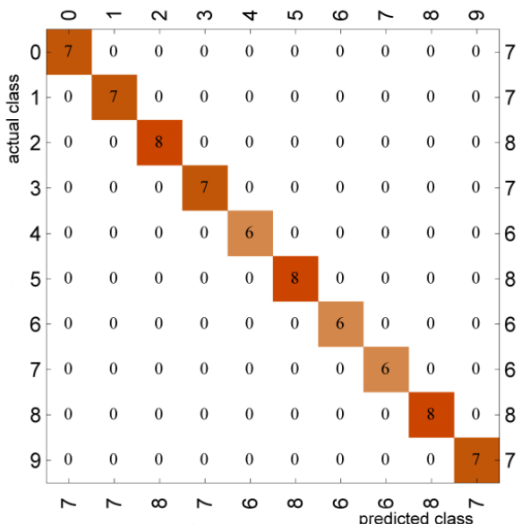
Тестування класифікатора на навчальних даних

```
⇒ ctraining=ClassifierMeasurements[logistic,trainingset]
⇐ ClassifierMeasurementsObject[ Classifier: LogisticRegression
                                Number of test examples 70 ]

⇒ ctraining["Accuracy"]
⇐ 1.
```

Матриця неточності наведена на Рис. 3.12

```
⇒ ctraining["ConfusionMatrixPlot"]
```




```
⇐
```

Рис. 3.12. Матриця неточності на навчальному наборі

Тестування класифікатора на тестових даних

```
⇒ ctesting=ClassifierMeasurements[logistic,testset]
⇐ ClassifierMeasurementsObject[ Classifier: LogisticRegression
                                Number of test examples 15 ]

⇒ ctesting["Accuracy"]
⇐ 0.866667
```

Матриця неточності,

```
⇒ ctesting["ConfusionMatrixPlot"]
```

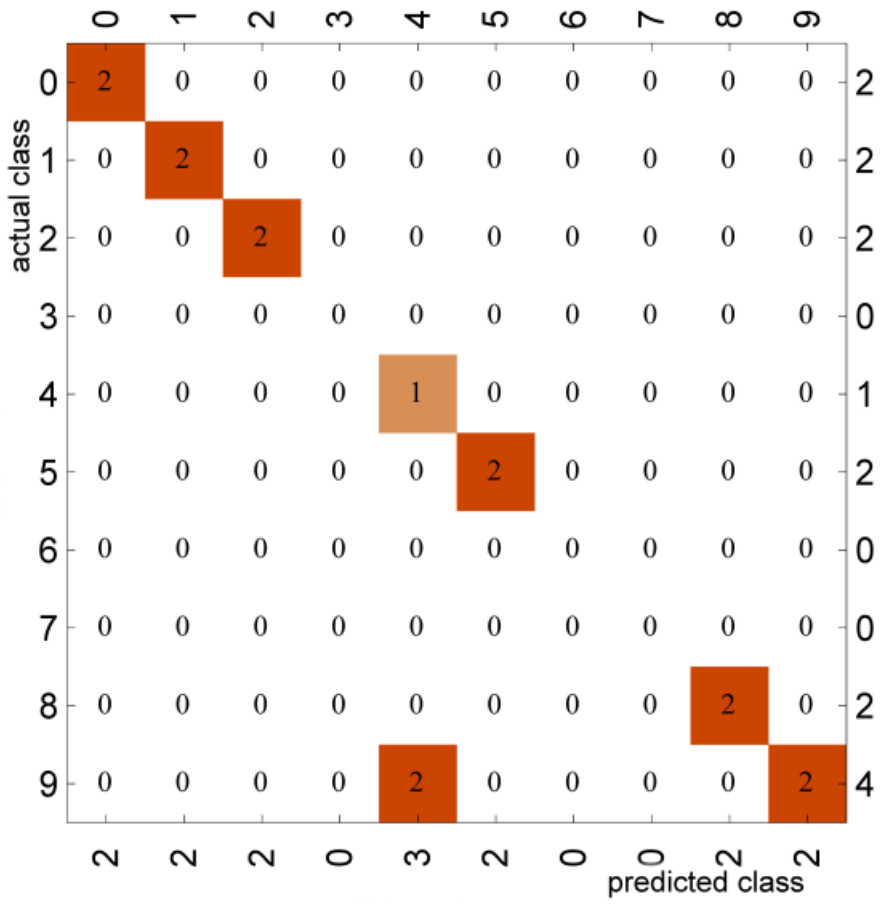


Рис. 3.13 Матриця неточності на тестовому наборі

Тут ми використовували зображення цифр як номінальні вхідні дані, числове кодування не використовувалося.

Скористаємось Python. Зараз ми використовуємо 75 елементів у навчальному наборі та 25 елементів у тестовому наборі. Набір для перевірки відсутній.

```
⇒ Xtrain=RandomSample[digitset,75];
⇒ Xtest=Complement[digitset,Xtrain];
```

Розмір зображень цифр дещо відрізняється, тому слід використовувати процес зміни розміру. Сірі зображення зі зміненим розміром мають розмір 16×20. Тому ми використовуємо вектор ознак розміром 320 кожної цифри. Елементи та мітки навчального набору (числове кодування).

```
⇒ Xtra=
    Map[Flatten[ImageData[ImageResize[#[[1]],{16,20}]],1]&,Xtrain];
⇒ ytra=Map[#[[2]]&,Xtrain];
```

Аналогічно для тестового набору,

```
⇒ Xtes=
    Map[Flatten[ImageData[ImageResize[#[[1]],{16,20}]],1]&,Xtest];
⇒ ytes=Map[#[[2]]&,Xtest];
```

Зберігаємо отримані дані для Python.

```
⇒ Export["Xtest.mtx",Xtes];
⇒ yy={ytes};
⇒ Export["ytest.mtx",yy];
⇒ Export["Xtrain.mtx",Xtra];
⇒ yy={ytra};
⇒ Export["ytrain.mtx",yy];
```

Вектор ознак розміром 320, що представляють цифру у вигляді зображення.

```
⇒ Xtra//Dimensions
⇐ {75,320}
```

Підготовка до читання файлів *.mtx,

```
import numpy as np
from numpy import array, matrix
from scipy.io import mmread, mmwrite
```

Зчитаємо дані, виконавши такий код:

```
X=mmread('Xtrain.mtx')
y=mmread('ytrain.mtx')
yy=y[0]
```

Процес навчання на даних

```
from sklearn.linear_model  import LogisticRegression
lr=LogisticRegression(C=100.,random_state=1).fit(X,yy)
```

прогнозований результат, що дає Python

```
ytr=lr.predict(X)
```

```
ytr
```

```
⇐ {0,9,7,7,7,4,1,2,5,1,1,5,0,0,1,3,7,0,1,1,8,9,8,9,4,
   0,6,5,5,5,7,3,8,6,9,8,7,9,1,9,5,6,8,6,4,1,7,2,6,6,
   0,2,2,0,5,9,1,3,3,7,4,2,4,2,5,3,7,2,8,4,0,3,5,8,2}
```

```
⇒ yP=%;
```

Початкові навчальні значення

```
⇒ yy
⇐ {0,9,7,7,7,4,1,2,5,1,1,5,0,0,1,3,7,0,1,1,8,9,8,9,4,
   0,6,5,5,5,7,3,8,6,9,8,7,9,1,9,5,6,8,6,4,1,7,2,6,6,
   0,2,2,0,5,9,1,3,3,7,4,2,4,2,5,3,7,2,8,4,0,3,5,8,2}
⇒ yT=%;
```

Помилка є такою:

```
⇒ Norm[yP-yT]
⇐ 0
```

Точність на навчальній вибірці

```
Xt=mmread('Xtest.mtx')
yt=mmread('ytest.mtx')
yt
```

```
⇐ {4,1,9,4,9,6,6,0,7,8,3,9,6,3,3,4,6,8,2,2,3,4,8,5,0}
⇒ yT=First[%];
```

```
yte=lr.predict(Xt)
yte
```

```
⇐ {4,1,9,1,9,6,8,0,7,8,3,9,6,3,5,4,1,8,3,2,3,4,8,5,4}
⇒ yP=%;
```

```
print("Training set score: {:.2f}".format(lr.score(X, yy)))
```

```
print("Test set score: {:.2f}".format(lr.score(Xt, yt[0])))
```

```
Training set score: 0.76
```

Отже, Mathematica забезпечує дещо кращий результат завдяки використанню набору перевірок, однак параметр Python C = 100, що вказує на регуляризацію, також дає певне покращення набору тестів.

Дерево рішень - один з методів непараметричної класифікації, що найчастіше використовуються. У разі непараметричної моделі у неї немає параметрів, що підлягають оцінці, і домінує алгоритм, що застосовується. На основі особливостей даних моделі дерева рішень вивчають низку питань, щоб визначити мітки класів зразків. У ході цього процесу дані згодом будуть розділені, щоб зменшити домішку даних, що вимірюється ентропією даних. Чим більше дані змішані, тим вища ентропія. Отримана інформація є очікуване зниження ентропії, викликане поділом прикладів по заданому атрибуту. Ідея полягає в тому, щоб почати зі змішаних класів і продовжувати розбиття доти, доки кожен вузол не досягне свого спостереження найчистішого класу.

Переваги цього алгоритму у тому, що він вимагає великих обчислювальних витрат, простий у реалізації, а уявлення знань легко інтерпретується. Однак вона має тенденцію мати низьку точність. Додаткові особливості алгоритму включають відсутність визначеної форми моделі, відповідність моделі найкращої можливої класифікації на основі даних і надання найкращих результатів, коли більшість змінних мають категоріальний характер, з одного боку, а також коли викиди і відсутні. з іншого боку, цінності враховуються у деревах рішень.

Можна покращити продуктивність класифікатора дерева рішень, використовуючи ваги або ансамбль деревоподібних моделей, відомий як метод випадкового лісу.

Це можна як ансамбль дерев рішень, і алгоритм переважно наступний:

1) Випадково вибрати розмір вибірки *n* з навчального набору із заміною,

2) Виконати обчислення дерева рішень,

3) Повторити кроки 1 і 2 *k* разів,

4) Об'єднати прогноз по кожному дереву, щоб призначити клас, позначений більшістю голосів.

Розглянемо приклад застосування розробленого методу. У наступному прикладі змінна відповіді має лише два класи; грати в теніс чи ні. Чи гратиме Джон у теніс чи ні, якщо погода буде дощова, вологість висока і вітер слабкий? Таблиця 3.1 складена з урахуванням різних умов, зафіксованих у різні дні. Доступна така інформація:

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Week	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Week	Yes
D14	Rain	High	Strong	No

Рис 3.13. Тренувальний набір даних задачі

Характеристики→ Outlook: Values → Sunny, Rain, Overcast,

Humidity: Values → High, Normal, Wind: Values → Weak, Strong,

Класи даних→Play: Values → Yes, No.

У навчальному прикладі вказано 9 Так проти 5 Ні.

Наш навчальний набір складається із 14 елементів. Вхідний елемент характеризується нечисловим вектором ознак із трьома елементами (Прогноз, Вологість та Вітер). Вихідні дані є двійковими (Так, Ні), що говорять про те, чи належить вхід до безлічі Play чи No. Проблема прогнозування полягає в наступному: належить вхідний елемент (Rain, High, Weak) набору Play чи No? Розділимо елементи навчального набору за першою ознакою Outlook. Ми отримуємо три підмножини (значення) Sunny, Humidity та Rain. Набір «Sunny»

можна класифікувати за другою ознакою «Вологість», а набір «Rain» можна класифікувати за третьою ознакою «Wind». Таким чином, ми отримуємо п'ять наборів: перші два чисті за ознакою Вологість, третій за Перспективою та останні два за Вітром такі як на Рис. 3.14.

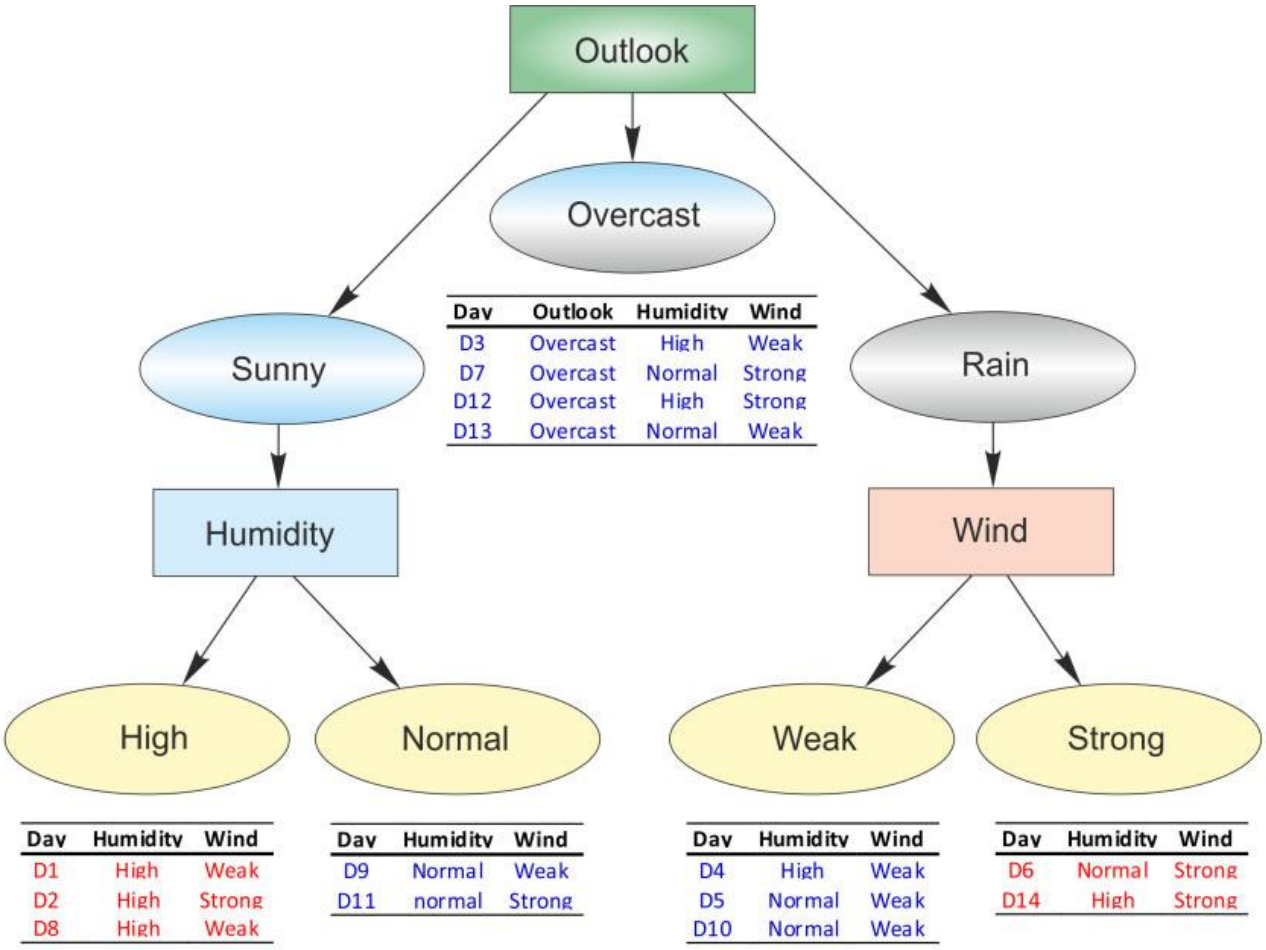


Рис. 3.14. Чисті підмножини дерева рішень.

Оцінку дерева можна побачити на Рис. 3.15, де за результатами прогноз – так.

Давайте вирішимо задачу за допомогою Mathematica. Пари введення-виведення відповідно до Таблиці 3.1.

```
⇒ dataTraining={{"Sunny","High","Weak"} → "No",
    {"Sunny","High","Strong"} → "No",
    {"Overcast","High","Weak"} → "Yes",
    {"Rain","High","Weak"} → "Yes",
    {"Rain","Normal","Weak"} → "Yes",
    {"Rain","Normal","Strong"} → "No",
    {"Overcast","Normal","Strong"} → "Yes",
    {"Sunny","High","Weak"} → "No",
    {"Sunny","Normal","Weak"} → "Yes",
    {"Rain","Normal","Weak"} → "Yes",
    {"Sunny","Normal","Strong"} → "Yes",
    {"Overcast","High","Strong"} → "Yes",
    {"Overcast","Normal","Weak"} → "Yes",
    {"Rain","High","Strong"} → "No"};
⇒ TableForm[dataTraining,TableAlignments → {Right,Center}]
⇐ {Sunny,High,Weak} → No
    {Sunny,High,Strong} → No
    {Overcast,High,Weak} → Yes
    {Rain,High,Weak} → Yes
    {Rain,Normal,Weak} → Yes
    {Rain,Normal,Strong} → No
    {Overcast,Normal,Strong} → Yes
    {Sunny,High,Weak} → No
    {Sunny,Normal,Weak} → Yes
    {Rain,Normal,Weak} → Yes
    {Sunny,Normal,Strong} → Yes
    {Overcast,High,Strong} → Yes
    {Overcast,Normal,Weak} → Yes
    {Rain,High,Strong} → No
```

Давайте створимо функцію класифікації,

```
⇒ c=Classify[dataTraining,Method → "DecisionTree",
    PerformanceGoal → "Quality"]
⇐ ClassifierFunction[ Input type: {Nominal, Nominal, Nominal}
                      Classes: No, Yes ]
```

Надаємо наші дані як вхідні дані для класифікатора,

```
⇒ c[{"Rain","High","Weak"}]
⇐ Yes
```

Для «Так» чи «Ні» ми тепер можемо отримати їх ймовірності:

```
⇒ c[{"Rain","High","Weak"},"Probabilities"]
⇐ <|No → 0.142857,Yes → 0.857143|>
```

Можна отримати додаткову інформацію про процес класифікації, а також класифікатор,

```
⇒ ClassifierInformation[c]
```

⇐

Classifier Information

Input time	{Nominal, Nominal, Nominal}
Casses	No, Yes
Method	decision Tree
Accuracy	45.6% ± 6.6%
Loss	0.556 ± 0.059
Single evaluation time	1.59 ms/example
Batch evaluation speed	149. example/s
Classifier memory	105. kB
Training examples used	14 examples
Training time	1.19 s

Точність процесу класифікації, а також матрицю неточності можна легко обчислити.

```
⇒ cm=ClassifierMeasurements[c,dataTraining]
⇐ ClassifierMeasurementsObject[ Classifier: DecisionTree
                                Number of test examples 14 ]

⇒ cm["Accuracy"]
⇐ 1.
⇒ cm["ConfusionMatrixPlot"]
```

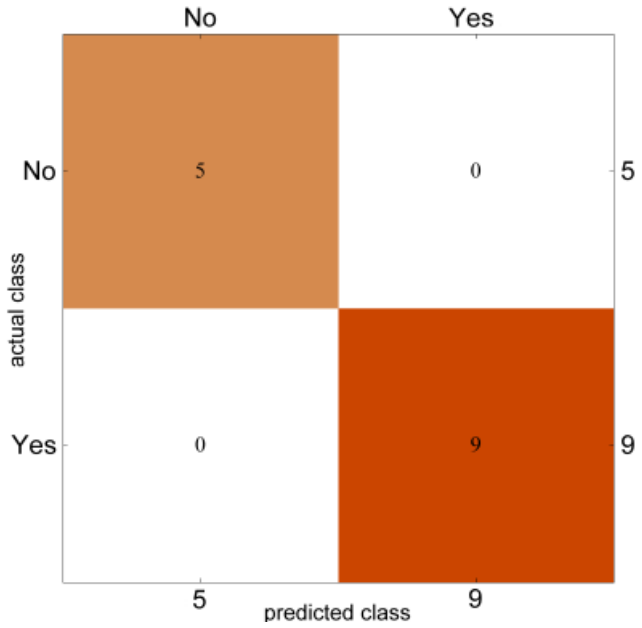


Рис. 3.15. Матриця неточності для системи даних

Приклад. Тепер ми хочемо класифікувати зображення за двома різними категоріями: сніговики та кубики, див. Рис. 3.16 і Рис. 3.17.

Рис. 3.16. Представники категорії сніговики

Рис. 3.17. Представники категорії кубиків

Виділення ознак цих зображень, представлених матрицями зображень розміром 64×64 пікселі, здійснювалося за допомогою вейвлет-перетворення з використанням фільтра Добеші другого порядку, а потім з використанням методу

усереднення для семи смуг спектру, що не перетинаються. Наприклад, давайте розглянемо версію зображення у відтінках сірого.

Рис. 3.18. Зображення для аналізу

Створимо форму списку матриці зображення,

```
⇒ dims = Flatten[ImageData[im]]
```

DFT використовує 6 рівнів уточнення,

```
⇒ wtr=DiscreteWaveletTransform[dims,DaubechiesWavelet[2],6]
⇐ DiscreteWaveletData[ Data dimension: {4096}
                       Refinements: 6 ]
```

Побудова графіка коефіцієнтів вейвлет-перетворення на різних рівнях уточнення,

```
⇒ WaveletListPlot[wtr,Ticks → Full]
```

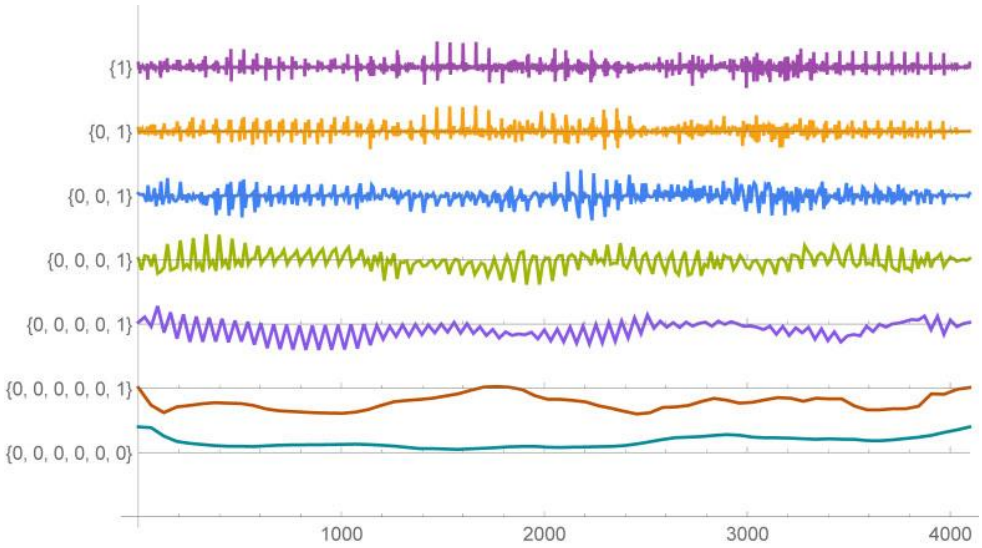


Рис. 3.19. Коефіцієнти DFT на різних рівнях уточнення

Енергетичний зміст коефіцієнтів $W_{i,j}$, $j=1,2...,n_i$ на і-му рівні може бути виражений як:

$$\left|\log\left[\frac{1}{n_i}\sum_{j=1}^{n_i}W_{i,j}^{\ 2}\right]\right|$$

Це може бути і-й компонент вектора ознак зображення. Наприклад, враховуючи, що перший рівень:

```
⇒ Normal[wtr][[2]][[1]]
⇐ {1}
```

а відповідні коефіцієнти:

```
⇒ ListPlot[Normal[wtr][[2]][[2]],Joined → True,
    PlotStyle → Purple,PlotRange → All,AspectRatio → 0.2]
```

Рис. 3.20. Коефіцієнти DFT першому рівні уточнення

Енергетичний зміст цього рівня, першого елемента вектора ознак, можна обчислити як:

```
⇒ Log[Apply[Plus,Normal[wtr][[2]][[2]]^2]/Length[Normal[wtr][[2]][[2]]]]//Abs
⇐ 5.97594
```

Оскільки:

```
⇒ Map[#[[1]]&,Normal[wtr]]
⇐ {{0},{1},{0,0},{0,1},{0,0,0},{0,0,1},{0,0,0,0},{0,0,0,1},
    {0,0,0,0,0},{0,0,0,0,1},{0,0,0,0,0,0},{0,0,0,0,0,1}}
```

Тоді вектор ознак зображення буде такий як на Рис. 3.21,

```
⇒ Map[Abs[Log[ Apply[Plus,Normal[wtr][[2]][[2]]^2] / Length[Normal[wtr][[2]][[2]]] ]]
    &,{2,4,6,8,10,11,12}]
⇐ {5.97594,3.93134,2.74834,1.71002,0.515124,2.58898,0.746729}
```

Отже, розмірність цих векторів ознак дорівнює семи, $n = 7$. Вектори ознак сніговиків:

```
⇒ Snowman=Import["M:\\Snowman.dat"];
⇒ Dimensions[Snowman]
⇐ {10,7}
```

а також для кубиків

```
⇒ DiCes=Import["G:\\Dice.dat"];
```

Вихідне значення для класу сніговика дорівнює 1, а класу кубика — 0.

```
⇒ dataSetSnowman=Map[# → 1&,SnowMan];
⇒ dataSetDice=Map[# → 0&,DiCes];
```

З 10–10 зображень перші 7–7 вважаються елементами навчального набору, а останні 3–3 є тестовим набором.

```
⇒ trainingData=Join[RandomSample[dataSetSnowman,7],
    RandomSample[dataSetDice,7]];
```

Давайте створимо функцію класифікації, використовуючи метод випадкового лісу,

```
⇒ c=Classify[trainingData,Method → "RandomForest",
    PerformanceGoal → "Quality"]
⇐ ClassifierFunction[ Input type: NumericalVector (Length: 7)
                      Classes: 0, 1 ]
```

Точність процесу класифікації, а також матрицю неточності можна легко обчислити на основі навчальних даних, див. Рис. 3.22.

```
⇒ cm=ClassifierMeasurements[c,trainingData]
⇐ ClassifierMeasurementsObjec[ Classifier: RandomForest
                               Number of test examples 14 ]

⇒ cm["Accuracy"]
⇐ 1.

⇒ cm["ConfusionMatrixPlot"]
```

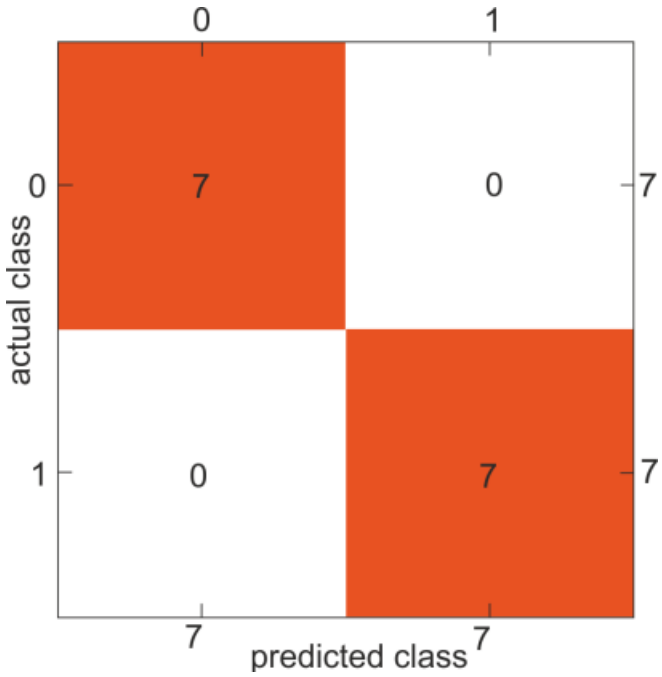


Рис. 3.21. Матриця неточності для навчального набору

3.4. Теоретичні основи кластеризації

Мета навчання без вчителя — виявити приховані закономірності чи структури даних, у яких немає цільової змінної, застосування методів класифікації чи регресії. Методи навчання без вчителя часто є складнішими, оскільки результати суб'єктивні і немає простої мети аналізу, такий як прогнозування класу чи безперервної змінної. Ці методи виконуються як частина розвідувального аналізу даних. Крім того, оскільки не існує загальноприйнятого механізму перевірки результатів, оцінка результатів, отриманих за допомогою методів навчання без вчителя, може бути складною (Han et al. 2012). Тим не менш, в даний час значення методів навчання без вчителя в різних галузях зростає, про що свідчать багато дослідників, які активно працюють над цією темою.

Алгоритм кластеризації K-Means являє собою ітераційний процес переміщення центрів кластерів або центроїдів у середнє положення складових їх точок та ітеративне перепризначення екземплярів найближчим до них кластерам до тих пір, поки не станеться істотна зміна кількості можливих центрів кластерів

або досягнуто кількості ітерацій. Кластеризація на основі прототипів означає, що кожен кластер представлений прототипом, який може бути або центроїдом (середнім) аналогічних точок з безперервними ознаками, (найбільш репрезентативною або точкою, що найбільш часто зустрічається) у разі категоріальних ознак. Хоча K-Means дуже добре ідентифікує кластери сферичної форми, одним із недоліків цього алгоритму кластеризації є те, що необхідно заздалегідь вказати кількість кластерів K. Неправильний вибір K може призвести до поганої продуктивності кластеризації. Пізніше в цьому розділі буде обговорюватися метод силуету для обчислення коефіцієнта силуету як індикатор викидів, які є корисними методами оцінки якості кластеризації та допомагають визначити оптимальну кількість кластерів K.

Проблема з K-середніми полягає в тому, що один або кілька кластерів можуть бути порожніми. Слід звернути увагу, що ця проблема не існує для K-методів або нечітких C-середніх. Цей метод можна використовувати, коли зразків дуже багато, кластерів не надто багато, а геометрія елементів плоска, що, по суті, означає, що кластер можна легко відокремити. При застосуванні K-середніх до реальних даних з використанням метрики евклідової відстані мотивація полягає в тому, щоб переконатися, що об'єкти виміряні в одному масштабі та при необхідності стандартизовані або масштабовані мінімально-максимально (стандартизовано в Python). Коротше кажучи, K-середні можуть бути корисні у разі великої кількості вибірок та невеликої кількості кластерів та плоскої геометрії.

Розглянемо простий приклад з використанням невеликого набору даних з репозиторію Python, так як на Рис. 3.22.

Виконаємо запуск сеансу Python у Mathematica,

```
⇒ session=
    StartExternalSession[<|"System" → "Python",
     "Version" → "3.5.4","Executable" →
      "C:\Users\Ben\AppData\Local\Programs\Python\Python35\
        python.exe"|>]//Quiet

⇐ ExternalSessionObject[
    SummaryPanel[ System: Python   EvaluationCount: None
                  UUID: 7b76966c2-ebb7-4ee8-bb25-02322d1456a8 ]]
```

Читання необхідних процедур,

```
from numpy import array, matrix
from scipy.io import mmread, mmwrite
import numpy as np
```

Імпорт функції кластеризації K-Means.

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

Читання набору даних і кластеризація у випадку припущення трьох кластерів,

```
X, y = make_blobs(random_state=1)
kmeans = KMeans(n_clusters=3).fit(X)
pre= kmeans.predict(X)
pred=np.array([pre])
```

Запишемо дані та отримані мітки у файли для Mathematica.

```
mmwrite('labels.mtx',pred)
mmwrite('dataX',X)
```

Координати центру кластерів такі:

```
cent= kmeans.cluster_centers_
cent
```

```
⇐ {{-6.58197,-8.17239},{-1.47108,4.33722},{-10.0494,-3.85954}}
```

Зчитування результатів Python у Mathematica для візуалізації Рис. 3.22.

```
⇒ centers=%;
⇒ labels=Import["labels.mtx"]//Flatten;
⇒ datap=Import["dataX.mtx"];
⇒ p0=ListPlot[datap,
    Frame → True,PlotMarkers → "\[DifferenceDelta]",Axes → None]
```

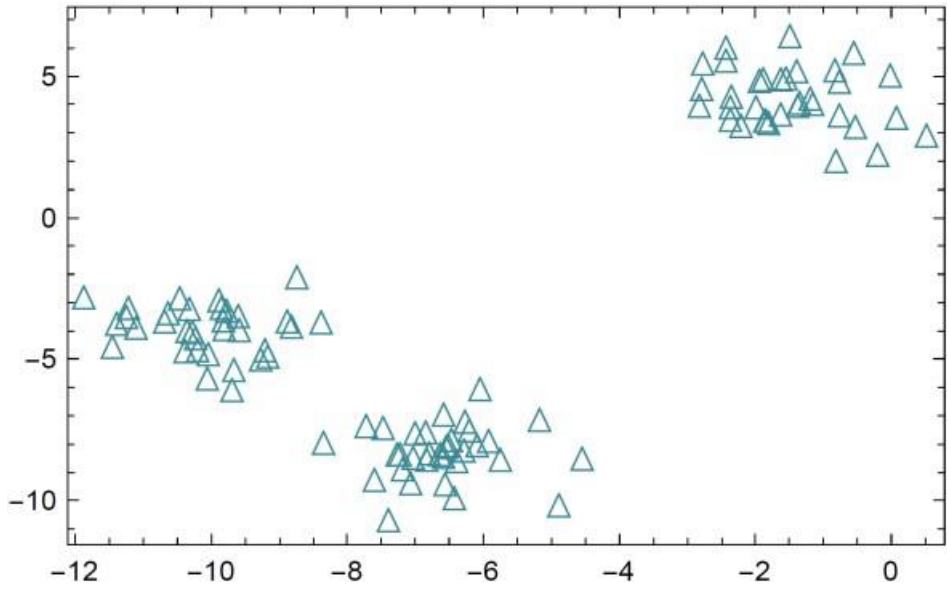


Рис. 3.22. Точки даних для кластеризації

Обчислення елементів, що належать до різних кластерів,

```
⇒ dataC=MapThread[Flatten[{#1,#2}]&,{datap,labels}];
⇒ dataC1=Map[Most,Select[dataC,#[[3]]==0&]];
⇒ dataC2=Map[Most,Select[dataC,#[[3]]==1&]];
⇒ dataC3=Map[Most,Select[dataC,#[[3]]==2&]];
```

Візуалізація різних кластерів з їх центрами виконуємо так як на Рис. 3.23,

```
⇒ Show[{p0,ListPlot[{dataC1,dataC2,dataC3},Frame → True,
    PlotMarkers → Automatic,Axes → None],
   ListPlot[centers,PlotMarkers → "♠",
    PlotStyle → {Black,Large}]},AspectRatio → 0.7]
```

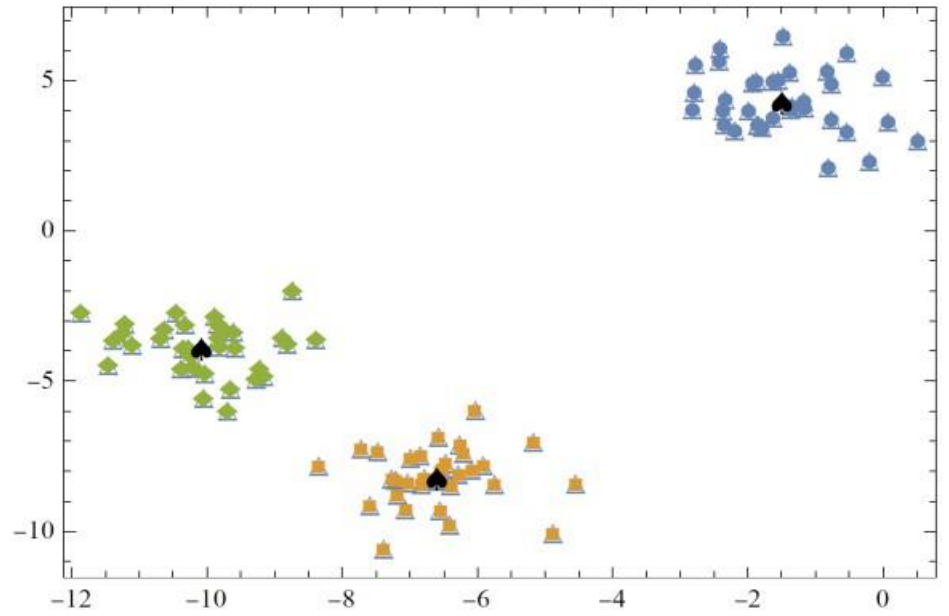


Рис. 3.23. Результати кластеризації з центрами кластерів за допомогою Python

Продуктивність кластеризації можна оцінити шляхом обчислення коефіцієнта силуету:

$$s = (b - a)\max(a, b),$$

де a - середня відстань між зразком і всіма іншими точками в тих же кластерах, а b - середня відстань між зразком та іншими точками найближчого кластера. Python має вбудовану функцію.

```
from sklearn import metrics
s=metrics.silhouette_score(X,prediction,'euclidean')
s
```

```
⇐ 0.769709
```

Тепер ми використаємо Mathematica для обчислення кластерів.

Знову ж таки, припускаючи три кластери,

```
⇒ clusters=FindClusters[datap,3,Method → "KMeans",
    PerformanceGoal → "Quality"];
```

Центрами кластерів є такі точки:

```
⇒ centers=Map[Mean[#]&,clusters]
⇐ {{-1.47108,4.33722},{-10.0494,-3.85954},{-6.58197,-8.17239}}
```

Візуалізуємо кластери та їх центри, результати подано на Рис. 3.24

```
⇒ Show[{p0,ListPlot[clusters,Frame → True,
    PlotMarkers → Automatic,Axes → None],ListPlot[centers,
    PlotMarkers → "♠",PlotStyle → {Black,Large}]},
    AspectRatio → 0.7]
```

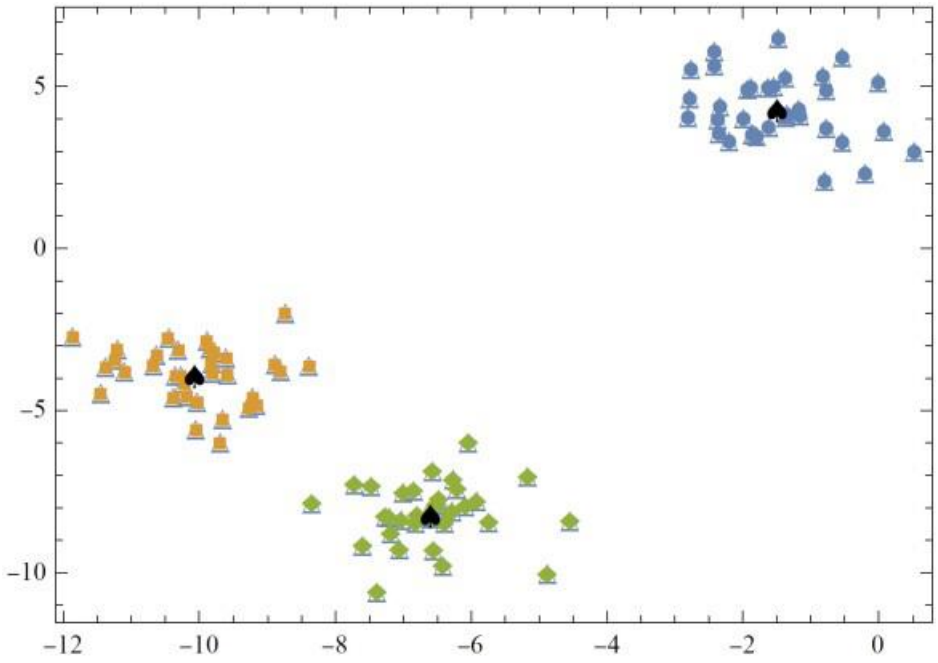


Рис. 3.24. Результати кластеризації з центрами кластерів, отримані за допомогою Mathematica.

Для визначення межі різних кластерів можна використовувати складніший метод. Ми можемо визначити функцію кластеризації, наприклад:

```
⇒ clustersF=ClusterClassify[datap,3,Method → "KMeans",
    PerformanceGoal → "Quality"]
⇐ ClassifierFunction[ Input type: NumericalVector (Length: 2)
                      Classes: 1, 2, 3 ]
```

Давайте перевіримо це у двох точках (-6, -8) та (-5,-1).

```
⇒ clustersF[{-6,-8},"Probabilities"]//Normal
⇐ 1 → 1.,2 → 3.92839 × 10^-36,3 → 4.92367 × 10^-10 [
```

Перша точка з ймовірністю 1 належить першому кластеру,

```
⇒ clustersF[{-6,-8},"Probabilities"]//Normal
⇐ 1 → 1.,2 → 3.92839 × 10^-36,3 → 4.92367 × 10^-10 [
```

а друга точка належить частково першому (0,31), частково другому (0,67) кластеру. Ці ймовірності можна як значення функції приналежності.

```
⇒ clustersF[{-5,-1},"Probabilities"]//Normal
⇐ {1 → 0.305066,2 → 0.665971,3 → 0.0289628}[
```

Візуалізуємо області кластера, результат подано на Рис. 3.25

```
⇒ pD=DensityPlot[clustersF[{x,y}],{x,-12,2},{y,-12,7},
    PlotPoints → 100];
⇒ Show[{pD,p0,ListPlot[clusters,Frame → True,
    PlotMarkers → Automatic,Axes → None],
  ListPlot[centers,PlotMarkers → "♠",PlotStyle → {Black,Large}]},
    AspectRatio → 0.7]
```

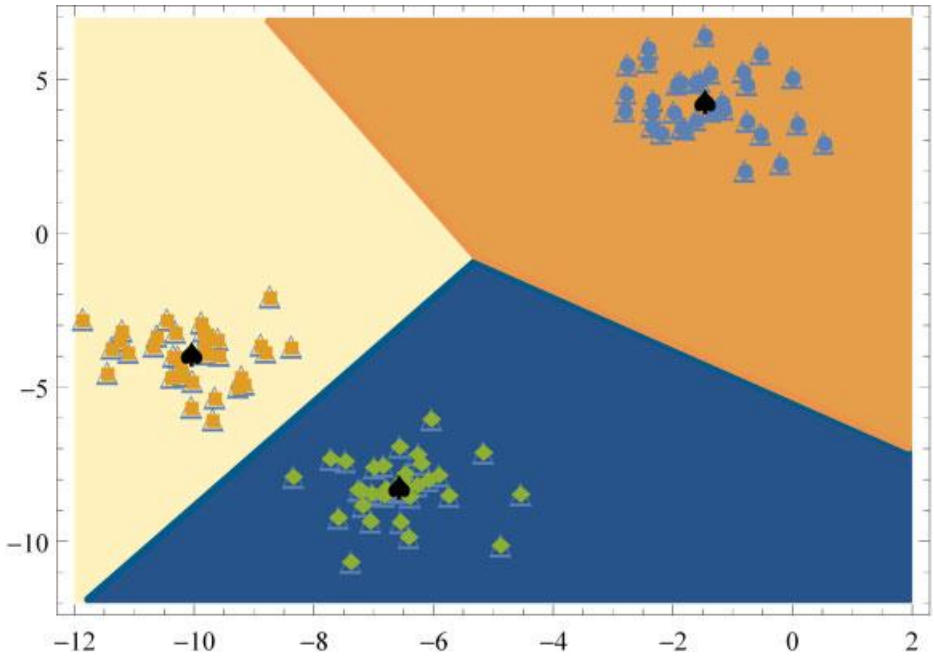


Рис. 3.25. Результати кластеризації з межею кластера.

У загальній вершині трьох регіонів є лише один елемент із приналежністю (1/3,1/3,1/3). Щоб обчислити коефіцієнт силуету для результату Mathematica, ми зберігаємо його мітки у файлі:

```
⇒ index=Map[#-1&,ClusteringComponents[datap,3,1,Method → "KMeans",
    PerformanceGoal → "Quality"]]
⇐ {0,1,1,1,2,2,2,1,0,0,1,1,2,0,2,2,2,0,1,1,2,1,2,0,1,
   2,2,0,0,2,0,0,2,0,1,2,1,1,1,2,2,1,0,1,1,2,0,0,0,0,
   1,2,2,2,0,2,1,1,0,0,1,2,2,1,1,2,0,2,0,1,1,1,2,0,0,
   1,2,2,0,1,0,1,1,2,0,0,0,0,1,0,2,0,0,1,1,2,2,0,2,0}

⇒ Export["pred.mtx",{index}]
⇐ pred.mtx
```

і зчитуємо в Python,

```
〉 index=mmread('pred.mtx')
```

Тоді ми можемо використати вбудовану функцію Python,

```
ind=index[0]
from sklearn import metrics
s=metrics.silhouette_score(X,ind,'euclidean')
s
```

```
⇐ 0.769709
```

Це означає, що Python і Mathematica забезпечили однакову якість для кластеризації у випадку трьох кластерів. Тепер давайте спробуємо здійснити кластеризацію з п'ятьма кластерами.

```
⇒ clusters=FindClusters[datap,5,Method → "KMeans",
    PerformanceGoal → "Quality"];
```

Центри кластерів,

```
⇒ centers=Map[Mean[#]&,clusters]
⇐ {{-1.47108,4.33722},{-9.8649,-4.74746},{-11.0221,-3.37753},
    {-9.3663,-3.32909},{-6.58197,-8.17239}}
```

В результаті центри кластерів такі як показано на Рис. 3.26.

```
⇒ Show[{p0,ListPlot[clusters,Frame → True,PlotMarkers → Automatic,
    Axes → None],
   ListPlot[centers,PlotMarkers → "♠",PlotStyle → {Black,Large}]},
    AspectRatio → 0.7]
```

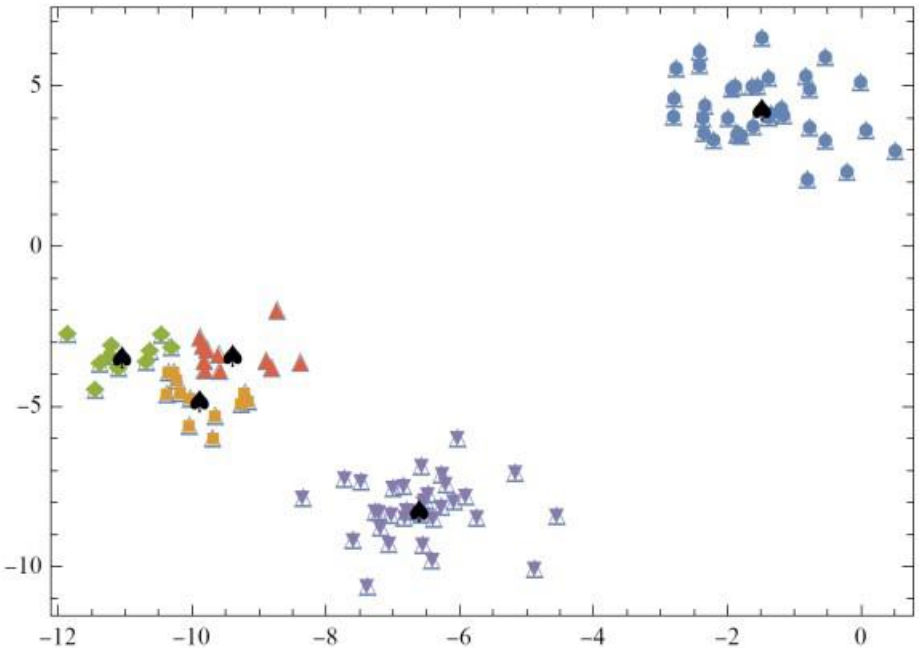


Рис. 3.26. Результати кластеризації у випадку п'яти кластерів

Щоб візуалізувати результати, ми виконуємо ті самі кроки, що й у випадку трьох кластерів,

```
⇒ clustersF=ClusterClassify[datap,5,Method → "KMeans",
    PerformanceGoal → "Quality"]
⇐ ClassifierFunction[ Input type: NumericalVector (Length: 2)
                      Classes: 1, 2, 3, 4, 5 ]

⇒ clustersF[{-6,-8},"Probabilities"]//Normal
⇐ {1 → 1., 2 → 3.34515 × 10^-57, 3 → 9.54873 × 10^-23,
    4 → 2.77988 × 10^-14, 5 → 3.27983 × 10^-13}

⇒ clustersF[{-5,-1},"Probabilities"]//Normal
⇐ {1 → 0.00221919, 2 → 0.00757256,
    3 → 1.97708 × 10^-11, 4 → 0.990203, 5 → 5.05125 × 10^-6 }

⇒ pD=DensityPlot[clustersF[{x,y}],{x,-12,2},{y,-12,7},
    PlotPoints → 100];
```

Отриманий результат демонструє Рис. 3.27.

```
⇒ Show[{pD,p0,ListPlot[clusters,Frame → True,
    PlotMarkers → Automatic,Axes → None],
  ListPlot[centers,PlotMarkers → "♠",PlotStyle → {Black,Large}]},
    AspectRatio → 0.7]
```

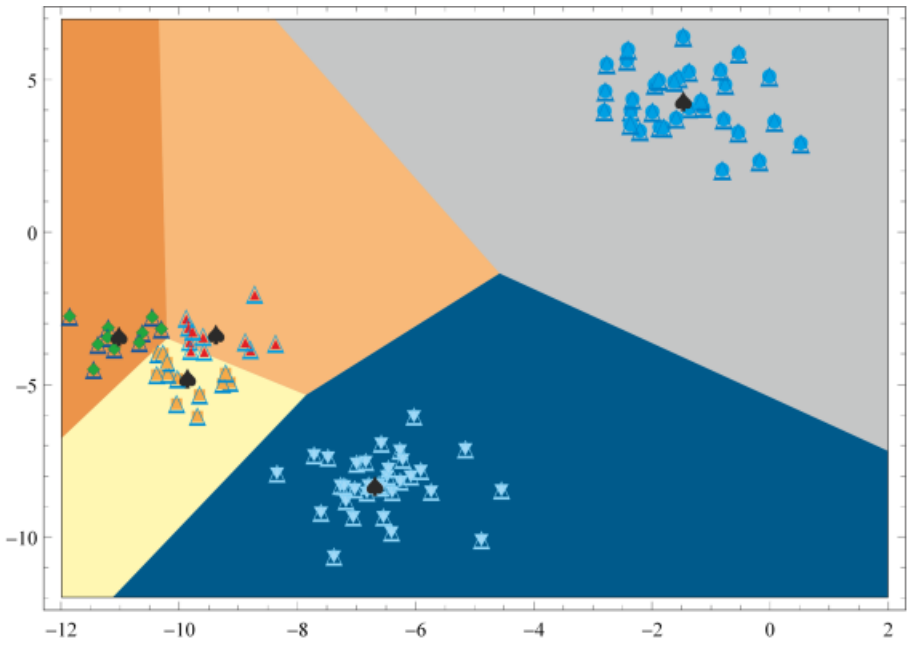


Рис. 3.27. Результати кластеризації у випадку п'яти кластерів з межами

Тепер значення коефіцієнта силуету менше, ніж у разі трьох кластерів, що свідчить про гіршу якість кластеризації, менш розділені кластери.

```
⇒ index=Map[#-1&,ClusteringComponents[datap,5,1,
    Method → "KMeans",PerformanceGoal → "Quality"]];
⇒ Export["pred.mtx",{index}]
⇐ pred.mtx
```

```
index=mmread('pred.mtx')
```

```
ind=index[0]
s=metrics.silhouette_score(X,ind,'euclidean')
s
```

```
⇐ 0.638162
```

Як бачимо, якість кластеризації настільки хороше в порівнянні з випадком трьох кластерів. Отже, коефіцієнт силуету є індикатором знаходження потрібної кількості кластерів.

3.5. Кластеризація зображень.

Як приклад зображення розглянемо зображення трьох видів: зебр, горил і пінгвінів, такі як на Рис. 3.28.

Рис. 3.28. Зображення трьох видів тварин, які потрібно згрупувати

Щоб використовувати числові дані, ми застосовуємо вилучення ознак зменшення розмірностей за допомогою t-SNE (t розподіленого стохастичного вбудовування сусідів). Існує клас алгоритмів візуалізації, званий алгоритмами навчання різноманіттю, які дозволяють створювати набагато складніші відображення та часто забезпечують кращу візуалізацію. Особливо корисним є алгоритм t-SNE. У цьому обчислює нове уявлення навчальних даних, але дозволяє перетворення нових даних. Це означає, що цей алгоритм не можна застосувати до тестового набору, він може лише перетворювати дані, для яких він навчений. Різноманітне навчання може бути корисним для дослідницького аналізу даних, але використовується рідко, якщо кінцевою метою є навчання з учителем.

Ідея t-SNE полягає в тому, щоб знайти двовимірне подання даних, яке якнайкраще зберігає відстані між точками.

```
⇒ animals=Join[zebra,gorilla,penquin];
```

Оскільки зображення мають різні розміри, ми змінюємо їх розмір до 396×512,

```
⇒ animalsReduced=Map[ImageResize[#,{392,512}]&,animals];
```

Представимо їх двовимірними векторами, як на Рис 3.29. Оскільки алгоритм зменшення розмірності вимагає багато часу замість процесора використовується графічний процесор.

```
⇒ reduced=
    DimensionReduce[animals,2,Method → "TSNE",TargetDevice → "GPU"];
```

Візуалізація результатів дає:

```
⇒ pani=ListPlot[MapThread[Labeled[#1,#2]&,{reduced,
    animalsReduced}],Frame → True,PlotRange → All]
```

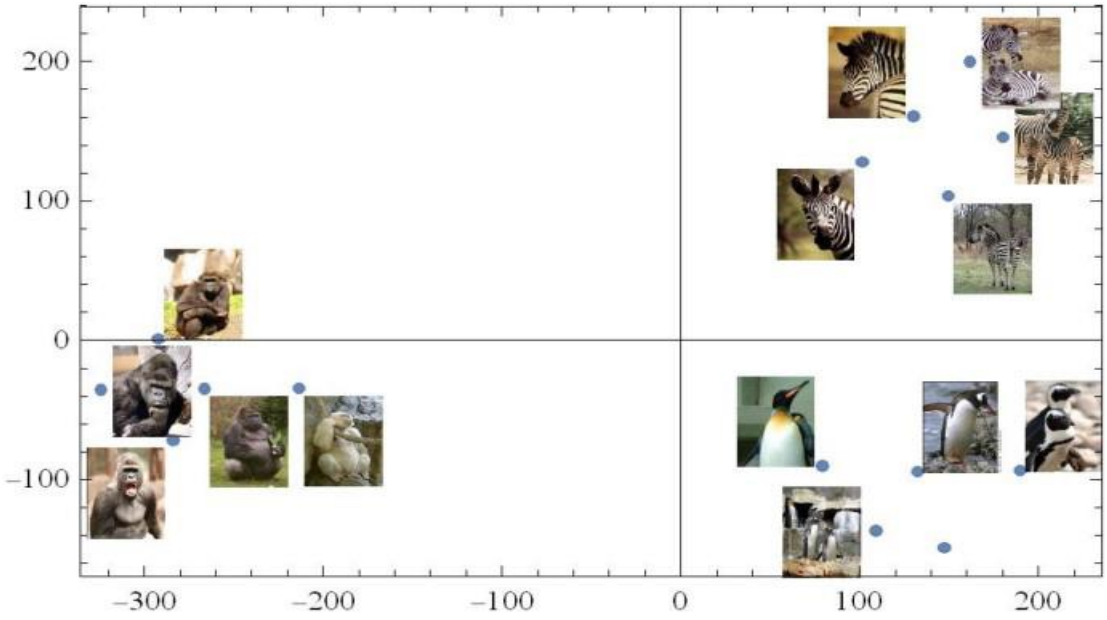


Рис. 3.29. Мапа зображень після застосування 2D-зменшення розмірів.

Як видно, різні види прозоро групуються в різні групи. Центри груп легко обчислити:

```
⇒ centersM=Map[Mean[#]&,Partition[reduced,5]]
⇐ {{144.597,147.594},{-276.146,-35.0111},{131.549,-112.583}}
```

Отже, тут ми не використовували будь-якої кластеризації, оскільки кластери цілком прозорі. Збережемо наведені координати зображень для Python.

```
⇒ X=reduced
⇐ {{180.106,145.609},{130.228,160.8},{149.597,103.534},
    {101.513,128.057},{161.541,199.97},{-324.505,-35.4624},
    {-266.282,-34.5745},{-292.198,0.989372},{-284.038,-71.5398},
    {-213.707,-34.4679},{79.1827,-89.9688},{189.736,-93.374},
    {132.477,-94.0187},{109.039,-136.691},{147.311,-148.862}}

⇒ Export["dataX.mtx",X]
⇐ dataX.mtx
```

Отримано той самий результат, що й у Mathematica,

```
kmeans = KMeans(n_clusters=3).fit(X)
prediction= kmeans.predict(X)
prediction
```

```
⇐ {0,0,0,0,0,1,1,1,1,1,2,2,2,2,2}
```

```
cent= kmeans.cluster_centers_
cent
```

```
⇐ {{144.597,147.594},{-276.146,-35.0111},{131.549,-112.583}}
⇒ centersP=%;
```

Якість кластеризації можна ще раз оцінити за коефіцієнтом силуету:

```
from sklearn import metrics
s=metrics.silhouette_score(X,prediction,'euclidean')
s
```

```
⇐ 0.779621
```

Кінцевий результат подано на Рис. 3.30:

```
⇒ Show[{pani,ListPlot[centersM,PlotMarkers → "♠",
    PlotStyle → {Blue,Large}],ListPlot[centersP,
    PlotMarkers → "▼",PlotStyle → {Red,Large}]},AspectRatio → 0.7]
```

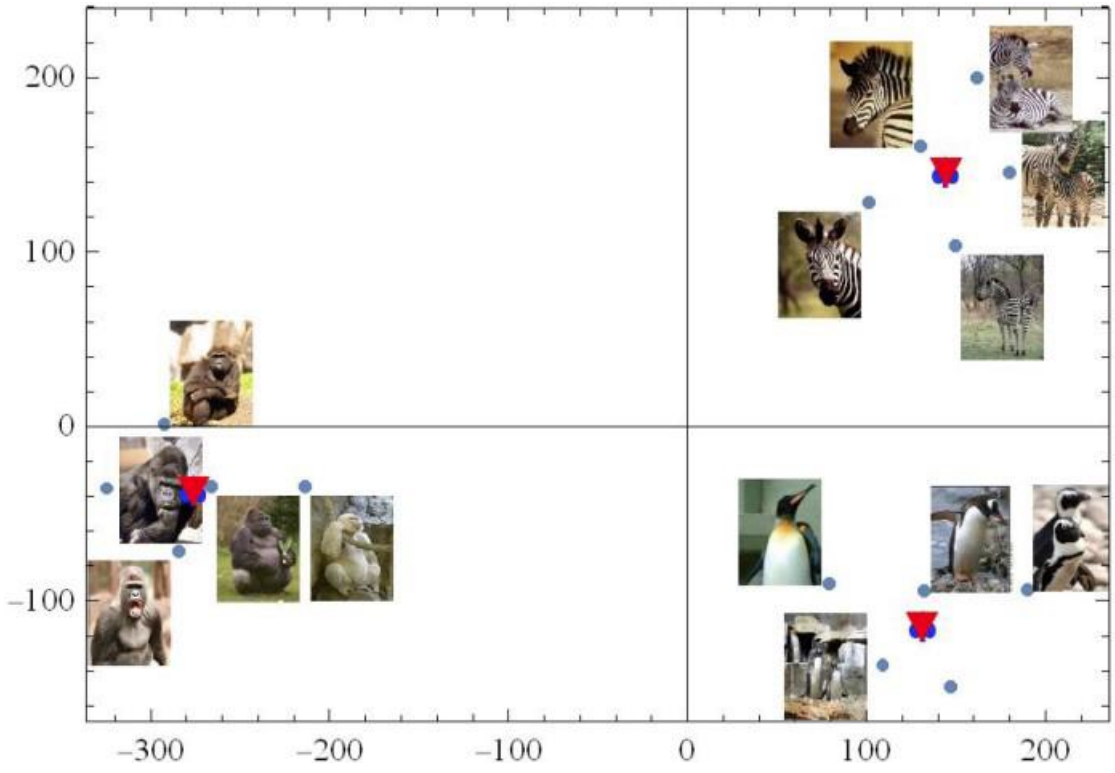


Рис. 3.30. Результати Python використвуючи KMeans Clustering у випадку K = 3

3.6. Просторова кластеризація додатків із шумом на основі щільності

Просторова кластеризація додатків із шумом на основі щільності (DBSCAN) — це алгоритм кластеризації, який групує разом точки (див. Рис. 3.31), які щільно упаковані (основні точки, червоний A) або досяжні з основних точок (жовті точки

B). разом відзначають точки викидів (синій N), які розташовані окремо у регіонах із низькою щільністю населення. Це може бути корисно у випадку невеликої кількості кластерів, не плоскої геометрії, а кластерів довільної форми і у разі викидів. Є два параметри: радіус червоних кіл та мінімальна кількість червоних (A) точок. Ці параметри є слабким місцем методу, оскільки результат дуже чутливий до цих значень параметрів.

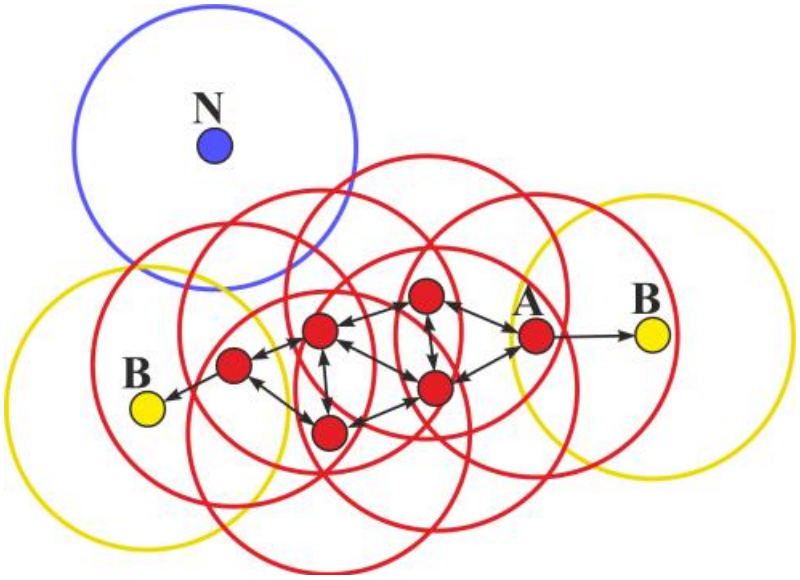


Рис. 3.31. Топологія методу DBSCAN

Проведемо ілюстрацію роботи методу на прикладі набору даних – Місяці.

Простий вступний приклад демонструє позитивні особливості методу. Давайте розглянемо набір даних про Місяць із репозиторію Python:

```
from sklearn.datasets import make_moons
X, y =make_moons(n_samples=200,noise=0.05,random_state=0)
```

Точки 2D даних є такі:

```
In[] := X
Out= {{0.816805,0.521645},{1.6186,-0.379829},{-0.0212695,0.273728},
      {-1.02181,-0.07544},{1.76655,-0.170699},{1.88203,-0.0423845},
      {0.974816,0.209994},{0.887988,-0.489367},{0.898652,0.366378},
      {1.11639,-0.534604},{-0.3638,0.827902},{0.247024,-0.238567},
      ....
      {0.424479,0.932688},{0.808614,0.535999},{0.940009,0.271114},
      {-0.0160918,0.373696},{-0.536334,0.860268},{1.88282,0.244356},
      {0.175752,-0.007231},{0.124236,1.0079},{1.62153,-0.223285}}

In[] := trainX=%;
```

Щоб візуалізувати їх, ми використовуємо їх теги,

```
In[] := y
Out= {0,1,1,0,1,1,0,1,0,1,0,1,1,1,0,0,0,1,0,0,1,1,0,1,0,1,1,1,1,
      0,0,0,1,1,0,1,1,0,0,1,1,0,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,1,
      0,0,1,0,1,0,1,0,0,1,0,0,1,0,1,1,1,0,1,0,0,1,1,0,1,1,1,0,0,
      0,1,1,0,0,1,0,1,1,1,1,0,1,1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,1,
      0,1,1,0,0,0,1,0,1,0,0,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,0,0,0,
      0,1,1,0,1,1,1,0,0,1,0,1,1,0,0,1,1,0,1,1,1,0,1,1,1,0,0,0,0,
      1,1,1,0,0,0,1,0,1,1,1,0,0,1,0,0,0,0,0,0,1,0,1,1,0,1}

In[] := clusters=%;
In[] := total=MapThread[{#1,#2}&,{trainX,clusters}];
In[] := clust1=Select[total,#[[2]]==0&];
In[] := clust2=Select[total,#[[2]]==1&];
In[] := pclust1=Map[#[[1]]&,clust1];
In[] := pclust2=Map[#[[1]]&,clust2];
```

Тоді на Рис. 3.32 показано два набори даних,

```
In[] := p0=ListPlot[{pclust1,pclust2},PlotStyle → {Pink,Green}]
```

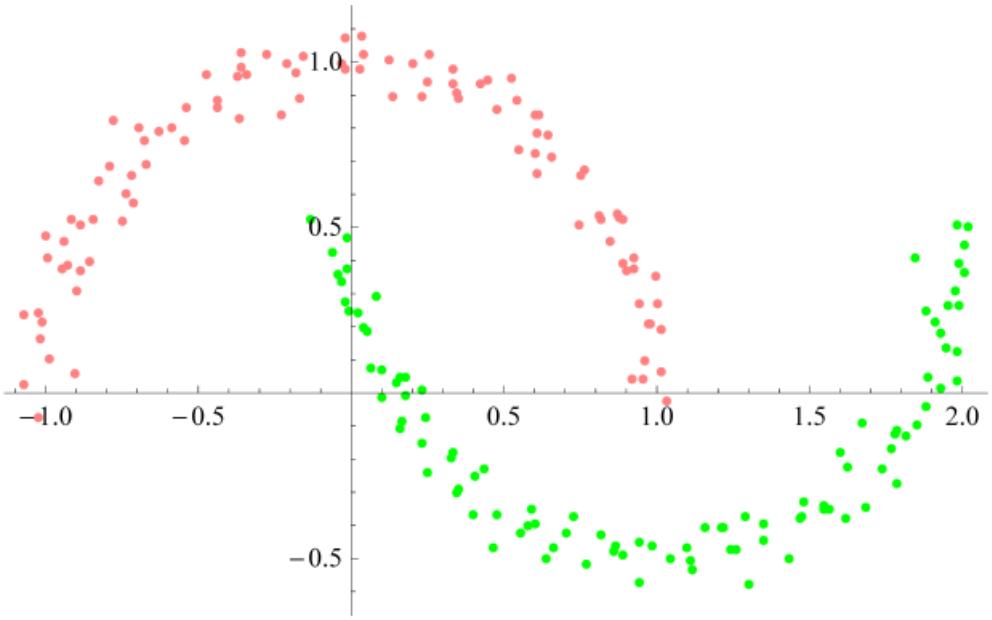


Рис 3.32. Набори даних про місяць.

Спочатку давайте спробуємо розділити два набори за допомогою кластеризації K-середніх. І тут необхідно вказати кількість кластерів.

Використовуючи Mathematica, ми отримуємо,

```
⇒ c=FindClusters[trainX,2,Method → "KMeans"];
```

З Рис. 3.32 видно, що цей метод забезпечує лише лінійне розділення,

```
⇒ ListPlot[{c[[1]],c[[2]]}]
```

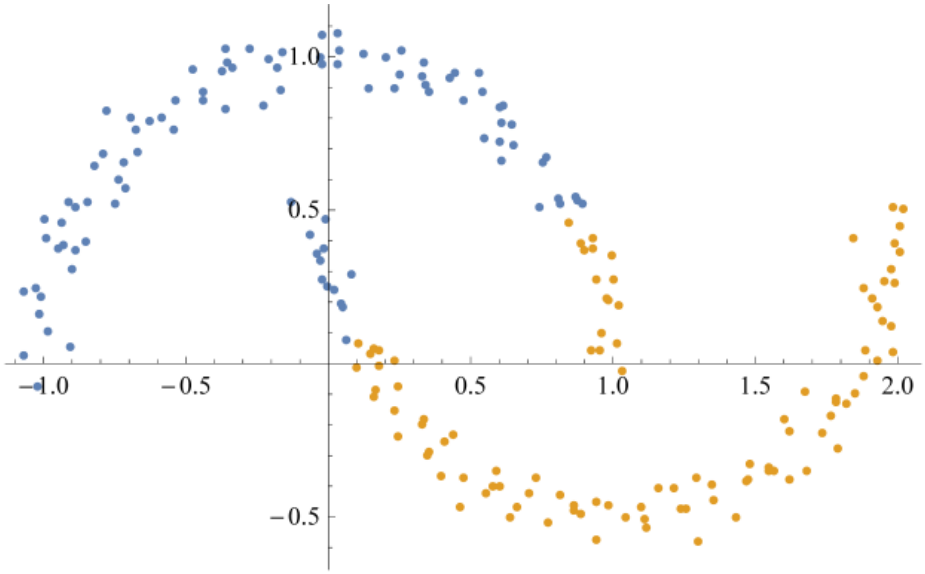


Рис. 3.33. Набори даних про Місяць кластеризовані за допомогою K-Mean Clustering.

У разі ієрархічної кластеризації нам не потрібно попередньо вказувати кількість кластерів.

```
⇒ Needs["HierarchicalClustering`"]
⇒ DendrogramPlot[trainX,TruncateDendrogram → 10,HighlightLevel → 2]
```

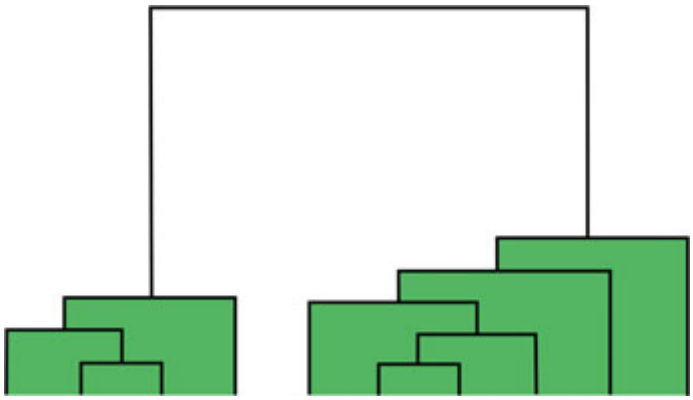


Рис. 3.34. Дендрограма кластерного набору даних про Місяць.

```
⇒ XN=MapThread[(#1 → ToString[#2])&,{trainX,Range[Length[trainX]]}];
```

Використовуючи два кластери, ми знову отримуємо в основному лінійне розділення, результат подано на Рис. 3.34.

```
⇒ c=FindClusters[XN,2]
⇐ {{1,3,4,11,15,17,19,20,23,25,30,31,32,34,35,39,40,42,43,46,48,51,
    54,56,57,59,60,62,64,66,67,69,70,72,74,76,77,79,82,85,86,88,91,
    94,99,100,103,105,106,107,108,110,111,112,113,114,117,120,121,
    122,123,124,125,126,132,133,138,142,143,144,145,149,150,154,156,
    159,160,163,167,171,172,173,174,178,179,180,182,186,187,189,190,
    191,192,193,195,196,199},
    {2,5,6,7,8,9,10,12,13,14,16,18,21,22,24,26,27,28,29,33,36,37,38,
    41,44,45,47,49,50,52,53,55,58,61,63,65,68,71,73,75,78,80,81,83,
    84,87,89,90,92,93,95,96,97,98,101,102,104,109,115,116,118,119,
    127,128,129,130,131,134,135,136,137,139,140,141,146,147,148,151,
    152,153,155,157,158,161,162,164,165,166,168,169,170,175,176,177,
    181,183,184,185,188,194,197,198,200}}
⇒ c1=Map[ToExpression[#]&,c[[1]]]; c2=Map[ToExpression[#]&,c[[2]]];
⇒ C1=Map[trainX[[#]]&,c1];
⇒ C2=Map[trainX[[#]]&,c2];
⇒ ListPlot[{C1,C2}]
```

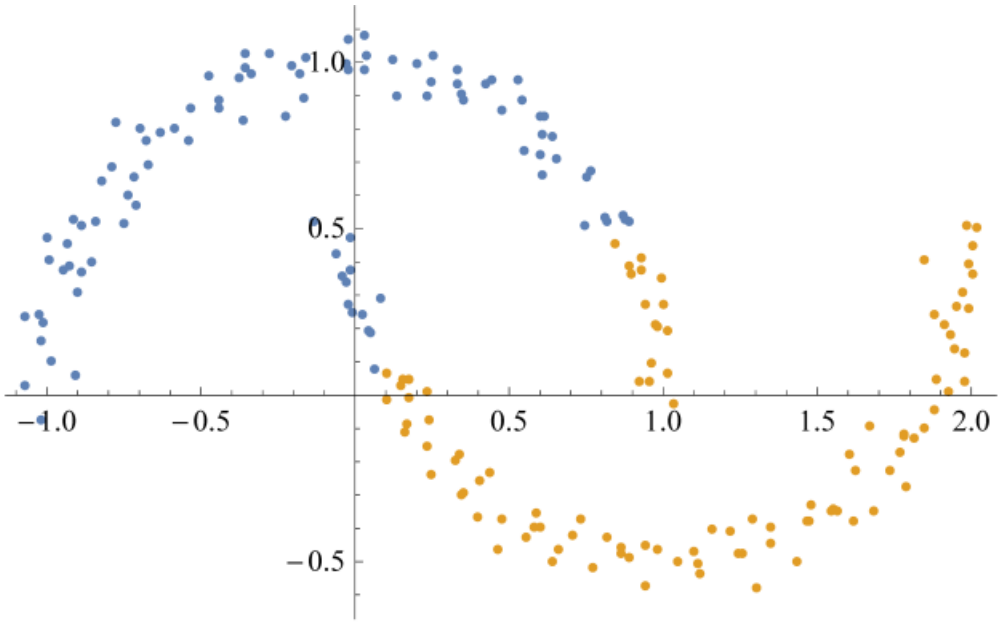


Рис. 3.35. Результати ієрархічної кластеризації

Для DBSCAN< знову не потрібно знати кількість кластерів,

```
⇒ c=FindClusters[trainX,Method → "DBSCAN"];
⇒ Length[c]
⇐ 2
```

Результат майже ідеальний, його побудова подана на Рис. 3.36.

```
⇒ ListPlot[{c[[1]],c[[2]]}]
```

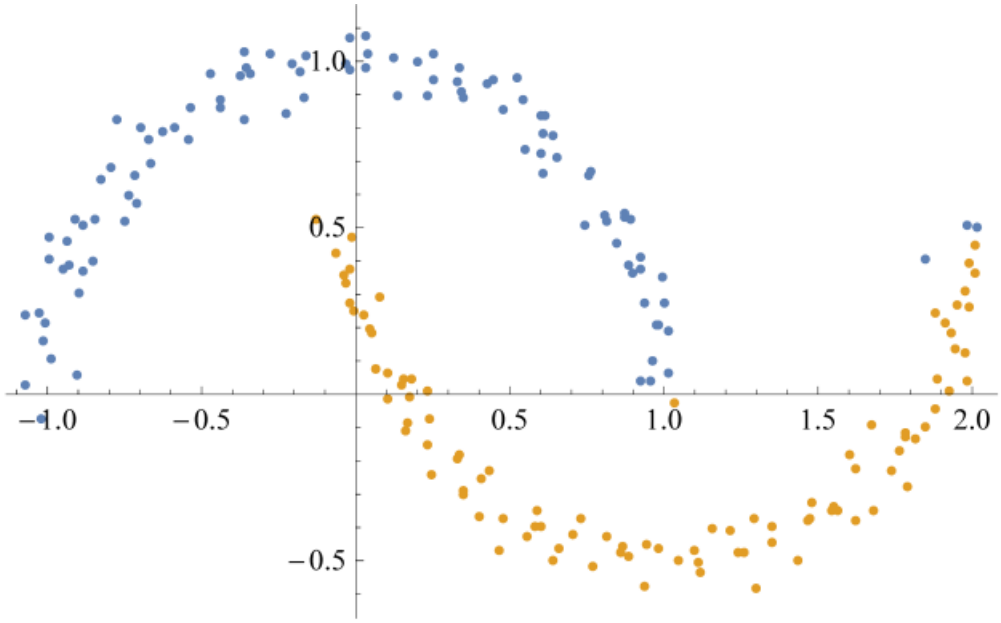


Рис. 3.36. Результати кластеризації DBSCAN

Деякі «неправильно згруповані» точки можна як шуми, саме викиди. Візуалізуємо регіони та межі кластерів. Мітки елементів даних:

```
⇒ index=Table[0,{i,1,200}];
⇒ Do[If[MemberQ[c[[2]],trainX[[i]]],index[[i]]=1],{i,1,200}];
⇒ index
⇐ {0,1,1,0,1,1,0,1,0,1,0,1,1,1,0,0,0,1,0,0,1,1,0,1,0,1,1,1,1,
    0,0,0,1,1,0,1,1,0,0,1,1,0,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,1,
    0,0,1,0,1,0,1,0,0,1,0,0,1,0,1,1,1,0,1,0,0,1,1,0,1,1,1,0,0,
    0,1,1,0,0,1,0,1,1,1,1,0,1,1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,1,
    0,1,1,0,0,0,1,0,1,0,0,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,0,0,0,
    0,1,1,0,1,1,1,0,0,1,0,1,1,0,0,1,1,0,1,1,1,0,1,1,1,0,0,0,0,
    1,1,1,0,0,0,1,0,1,1,1,0,0,1,0,0,0,0,0,0,1,0,1,1,0,1}
```

Тепер ми визначимо функцію кластеризації,

```
⇒ f=ClusterClassify[trainX,Method → "DBSCAN"]
⇐ ClassifierFunction[ Input type: NumericalVector (Length: 2)
                      Classes: 1, 2 ]
```

Отже, ми отримуємо:

```
⇒ Show[{DensityPlot[f[{u,v}],{u,-1.1,2.1},{v,-0.7,1.2},
    PlotPoints → 50],p0}]
```

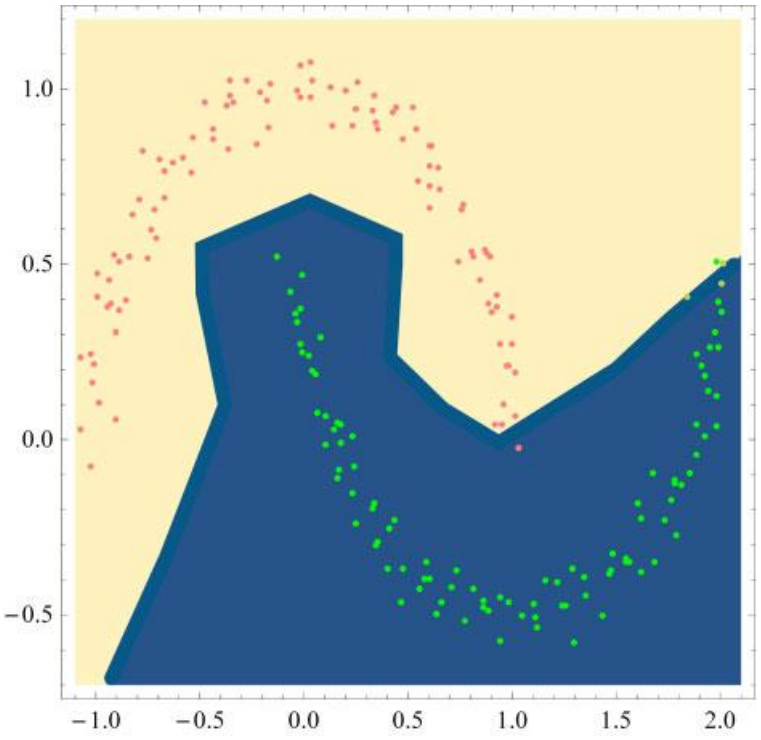


Рис. 3.37. Область кластерів у випадку кластеризації DBSCAN

На жаль, різниця між двома кластерами для перших координат $x \geq 1$ практично дорівнює нулю, що робить метод дуже чутливим до шумів або помилок вимірювань. У наступному розділі побачимо, що спектральна кластеризація може максимізувати цей запас, що призводить до надійної кластеризації.

3.7. Сегментація МРТ головного мозку

Обробка медичних зображень в даний час є найбільш складною областю, що розвивається. Магнітно-резонансні зображення (МРТ) є джерелом для розробки системи класифікації. Вилучення, ідентифікація та сегментація інфікованої області на магнітно-резонансному (МРТ) зображенні мозку є серйозною проблемою, але це стомлююче і трудомістке завдання, що виконується рентгенологами або клінічними експертами, а остаточна точність класифікації залежить тільки від їх досвіду. Щоб подолати ці обмеження необхідно використовувати комп'ютерні методи. Підвищення ефективності точності класифікації та зниження складності розпізнавання відіграють важливу роль у медичній візуалізації.

Розглянемо МРТ мозку, що на подано на Рис. 3.38.

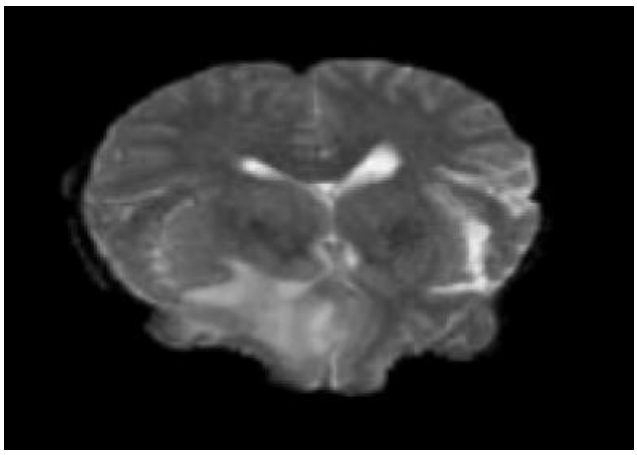


Рис. 3.38. МРТ головного мозку

Давайте скористаємось різними значеннями параметрів методу DBSCAN, щоб вивчити їх вплив на кластеризацію. Занадто малий радіус околу (ε) і мала сусідів призводить до утворення занадто великої кількості кластерів, Рис. 3.39.

```
⇒ ClusteringComponents[img01,Method → {"DBSCAN",
    "NeighborsNumber" → 3,"NeighborhoodRadius" → 0.01}]//Colorize
```

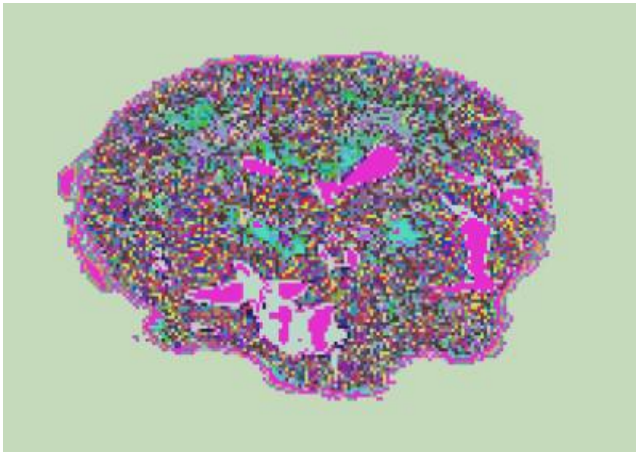


Рис. 3.39. Результат методу DSBCAN з ε = 0,01 і NeighboursNumber = 3

Збільшення мінімальної кількості сусідів призводить до появи навіть до більш фрагментованих кластерів. Оптимальне значення для мінімальних сусідів дорівнює розмірності даних плюс один або вище. У нашому випадку нехай це буде 2+2 = 4 до ε = 0,1, див. Рис. 3.40.

```
⇒ ClusteringComponents[img01,Method → {"DBSCAN",
    "NeighborsNumber" → 4,"NeighborhoodRadius" → 0.1}]//Colorize
```

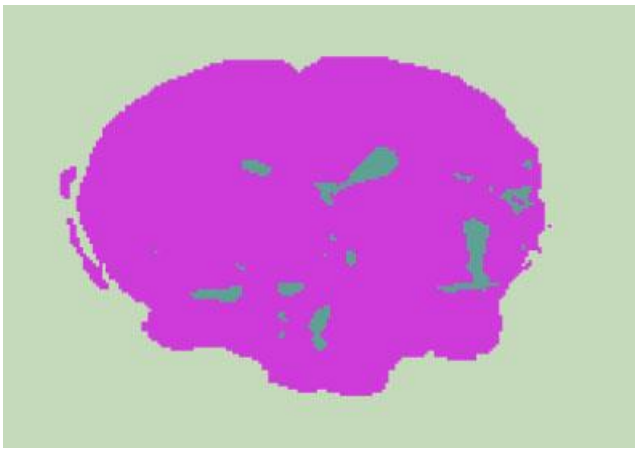


Рис. 3.40. Результат методу DSBCAN з ε = 0.1 and NeighborsNumber = 4

Далі в нас залишилося лише кілька кластерів. Однак, зменшивши ε = 0,053 ми можемо отримати реалістичну сегментацію, результат такий як на Рис. 3.41.

```
⇒ ClusteringComponents[img01,Method → {"DBSCAN",
    "NeighborsNumber" → 4,"NeighborhoodRadius" → 0.053}]//Colorize
```

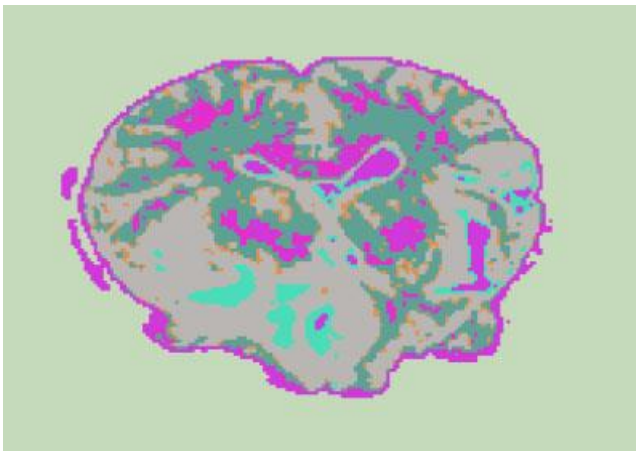


Рис. 3.41. Результат методу DSBCAN з ε = 0.053 and NeighborsNumber = 4

Таким чином, правильне налаштування параметрів методу DBSCAN є його найскладнішим місцем, проте дозволяє коректним вибором параметрів методу отримати актуальний результат.

3.8. Сегментація зображення папуги

Розглянемо наступне зображення папуги,

Рис. 3.42. Зображення папуги, яке потрібно сегментувати

Сегментація здійснюється за допомогою кластеризації пікселів із використанням різних методів. Розмір даних зображення:

```
⇒ datap=ImageData[img];
⇒ Dimensions[datap]
⇐ {526,800,3}
```

Кластеризація піксельних векторів RGB

```
⇒ X = Flatten[datap, 1];Dimensions[X]
⇐ {420800,3}
```

Спочатку ми використовуємо метод KMeans. Розглянемо 3 кластери:

```
⇒ XC=FindClusters[X,3,Method → "KMeans"];
```

Візуалізація кластеризованих векторів пікселів RGB у просторі RGB

```
⇒ ListPointPlot3D[XC, PlotStyle → PointSize[0.001],
    AxesLabel → {Red, Green, Blue}, BoxRatios → {1, 1, 1}]
```

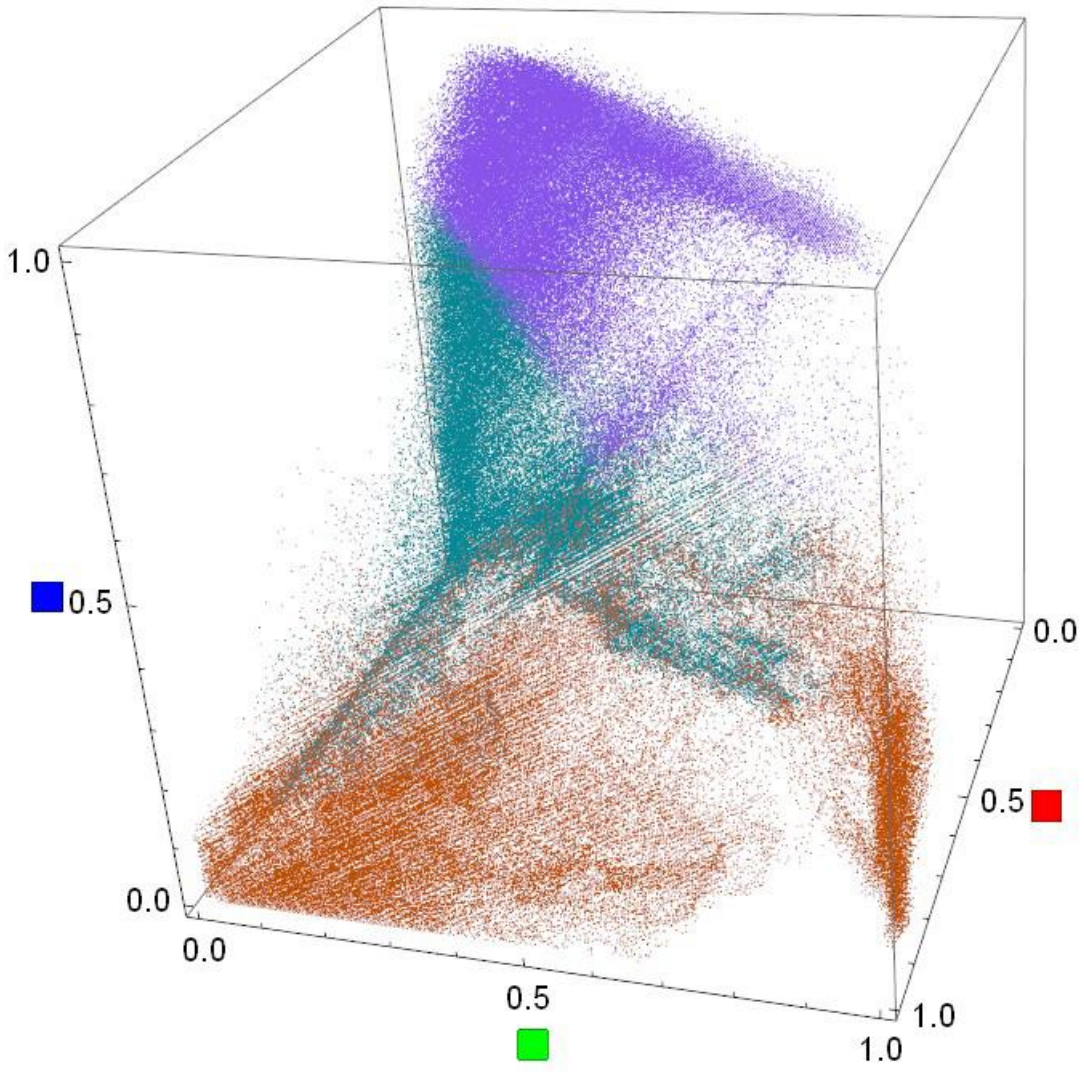


Рис. 3.43. Подання трьох кластерів у колірному просторі RGB

Використання кластеризації для сегментації

```
⇒ AbsoluteTiming[c=ClusteringComponents[
    img,3,Method → "KMeans",PerformanceGoal → "Quality"];]
⇐ {0.835659,Null}
```

Розфарбуємо сегментоване зображення

```
⇒ Colorize[c, ColorFunction → "RoseColors",
    ColorRules → {0 → Black}]
```

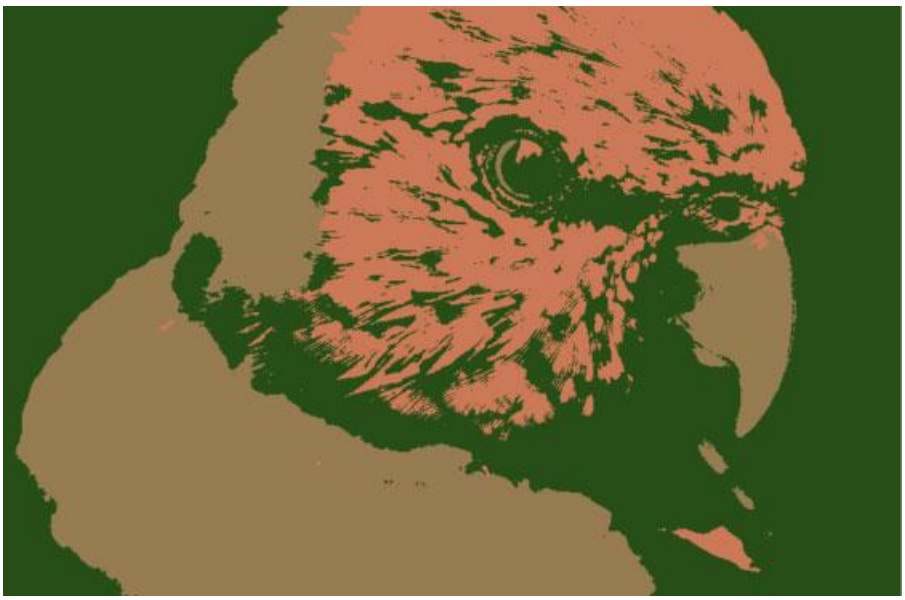


Рис. 3.44. Сегментоване зображення з використанням 3 кластерів

Якість кластеризації можна визначити шляхом обчислення оцінки силуету кластеризації. Тут ми використовуємо вбудовану функцію Python

Мітки кластерів:

```
⇒ index=Map[#-1&,c]//Flatten;
⇒ index//Dimensions
⇐ {420800}
```

Кількість кластерів:

```
⇒ nc=Max[index]+1
⇐ 3
```

Збереження даних для Python:

```
⇒ Export["predi.mtx",{index}]
⇐ predi.mtx

⇒ Export["predX.mtx",X]
⇐ predX.mtx
```

Зчитування даних в Python:

```
from numpy import array, matrix
from scipy.io import mmread, mmwrite
import numpy as np
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples,
    silhouette_score
```

```
index=mmread('predi.mtx')
```

```
X=mmread('predX.mtx')
```

Застосовуючи вбудовану функцію Python будемо мати:

```
ind=index[0]
from sklearn import metrics
```

```
s=silhouette_score(X,ind,'euclidean',sample_size=20000)
s
```

```
⇐ 0.457202
```

Далі була проведена кластеризація KMeans з різною фіксованою кількістю кластерів. Крім того, були застосовані різні методи кластеризації, такі як DBSCAN та спектральна кластеризація. У разі нових методів кількість кластерів визначається автоматично самим способом. Результати підсумовані у таблицях 3.1 та 3.2.

Метод	Час виконання	Оцінка силуету	Кількість кластерів
KMeans	0,77	0,43	2
KMeans	0,84	0,43	3
KMeans	1,25	0,44	4
KMeans	1,55	0,46	5

DBSCAN	1,63	0,14	2 (за замовчуванням)
Spectral	2,83	0,35	2 (за замовчуванням)

Табл.3.1. Порівняння методів класифікації

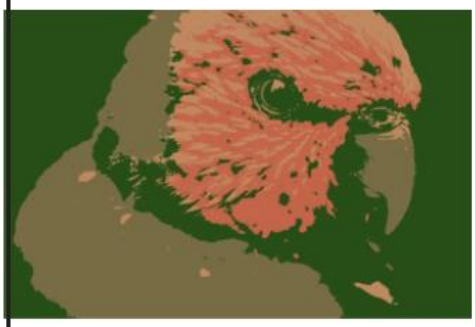
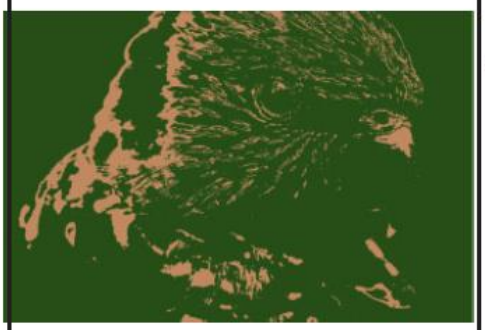
Method	Segmented image	Method	Segmented image
KMeans cluster=2		KMeans cluster=3	
KMeans cluster=4		KMeans cluster=5	
DBSCAN cluster=4		Spectral cluster=5	

Табл. 3.2. Сегментовані зображення

Враховуючи якість зовнішнього вигляду, час виконання процесу кластеризації та оцінку силуету, метод KMeans з двома кластерами виявляється найкращим вибором.

3.9. Регресія методом пошуку найближчих сусідів

Як було встановлено вище, KNearest Neighbours можна використовувати як кластеризації, так регресії. По суті, алгоритм складається з наступних кроків, які треба виконати:

1. Розгляньте фактичну точку даних у просторі ознак;

2. Обчисліть її відстані від кожної точки даних;

3. Виберіть середнє значення KNearest сусідів, яке формує наступне прогнозоване значення. На останньому етапі можна використовувати середньозважене значення, де ваги є зворотними відстанями. Це означає, що точки, що знаходяться ближче, мають більший вплив на результат прогнозування.

Запускаємо тепер сеанс Python у Mathematica,

```
⇒ session=
    StartExternalSession[<|"System" → "Python",
     "Version" → "3.5.4","Executable" →
      "C:\Users\Ben\AppData\Local\Programs\Python\Python35\
        python.exe"|>]//Quiet

⇐ ExternalSessionObject[
    SummaryPanel[ System: Python   EvaluationCount: None
                  UUID: 7b76966c2-ebb7-4ee8-bb25-02322d1456a8 ]]
```

Аналіз регресора KNeighbours

Ми проілюструємо вплив кількості сусідів на прогноз. По-перше, ми покажемо, що цей метод дає негладкий предиктор. Дані навчання та тестування для прогнозування функції $y = f(x)$.

Далі завантажимо невеликий набір даних з репозиторію Python і розподілимо його на навчальний та тестовий набори:

```
import mglearn
X, y = mglearn.datasets.make_wave(n_samples=40)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
 y_test= train_test_split(X, y,random_state=0)
```

Ми присвоюємо ці значення змінним Mathematica, забезпечуючи їх подальше використання у Mathematica.

```
X_train
⇐ {{0.085407},{1.85038},{-2.41397},{1.39196},{-0.35908},{0.645269},
   {2.79379},{-1.17455},{-1.1723},{0.60669},{-1.72597},{-2.06403},
   {0.14854},{-1.90905},{2.69331},{2.19706},{-2.60969},{2.70429},
   {-1.80196},{1.99466},{-1.9769},{-0.26358},{-2.6515},{-0.801829},
   {-2.16304},{-1.25263},{1.24844},{1.1054},{0.591951},{-0.752759}}
⇒ Xtrain=%;
```

```
X_test
⇐ {{-1.24713},{0.671117},{1.71106},{-2.06389},{-2.87649},
   {-1.89957},{0.554487},{2.81946},{-0.40833},{-2.7213}}
⇒ Xtest=%;
```

```
y_train
⇐ {0.697986,1.87665,-1.41502,0.779321,0.0939886,0.0352788,
   0.868933,0.0844854,0.0945257,1.00032,-1.5137,-2.47196,
   -0.527347,-1.67303,1.53708,1.49417,-0.47411,0.331226,
   -1.13455,0.754188,-2.08582,-0.986181,-1.52731,0.0975635,
   -1.12469,-0.340907,0.229562,0.254389,0.0349788,-0.448221}
⇒ ytrain=%;
```

```
y_test
⇐ {0.372991,0.217782,0.966954,-1.38774,-1.0598,
   -0.90497,0.436558,0.778964,-0.541146,-0.956521}
⇒ ytest=%;
```

Давайте візуалізуємо ці два набори. Тренувальний набір:

```
⇒ training=Map[Flatten[#,1]&,Transpose[Join[{Xtrain,ytrain}]]];
```

Тестовий набір:

```
⇒ test=Map[Flatten[#,1]&,Transpose[Join[{Xtest,ytest}]]];
```

На Рис. 3.45 показані набори даних для навчання (зелений диск) та для тестування (червоні квадрати),

```
⇒ p0=ListPlot[{training,test},PlotStyle → {Green,Red},Frame → True,
    Axes → None,PlotMarkers → {Automatic,Large},AspectRatio → 1]
```

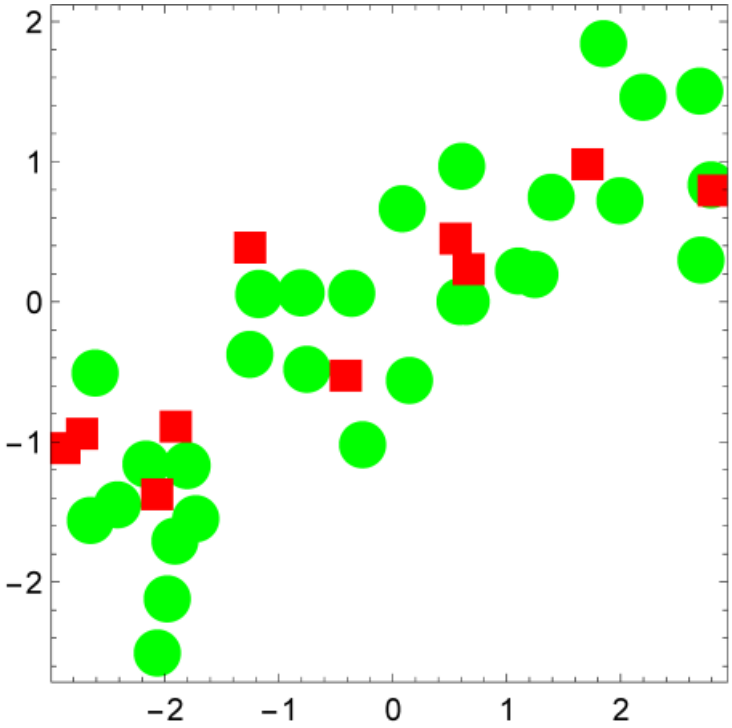


Рис. 3.45. Точки даних навчального (зеленого) та тестового (червоного) наборів у випадку регресії KNearest Neighbours

Використовуючи $k = 3$ сусідів, функція прогнозування f дорівнює:

```
⇒ datatraining=Flatten[Xtrain] → Flatten[ytrain];
⇒ f=Predict[datatraining,Method → {"NearestNeighbors",
    "NeighborsNumber" → 3},PerformanceGoal → "Quality"]
⇐ PredictorFunction[ Input type: Numerical
                     Method: NearestNeighbors ]
```

Передикторну функцію, яку ми отримали подано на Рис. 3.46 де ясно показано, що предикторна функція забезпечує локальну покрокову апроксимацію.

```
⇒ Show[{p0,Plot[f[x],{x,-3,3},PlotStyle → {Thin,Blue}]}]
```

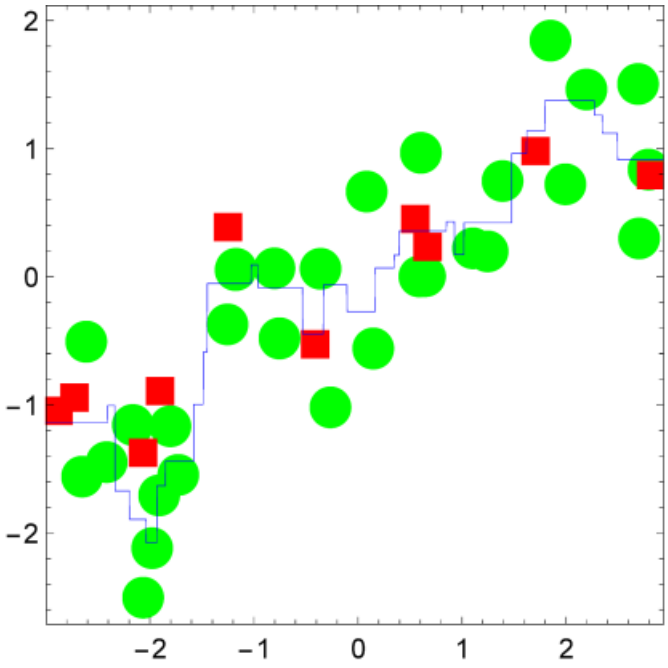


Рис. 3.46. Функція прогнозування з регресією K найближчих сусідів із використанням $k = 3$ сусідів

Помилку прогнозування на тестовому наборі можна візуалізувати, див. Рис. 3.47.

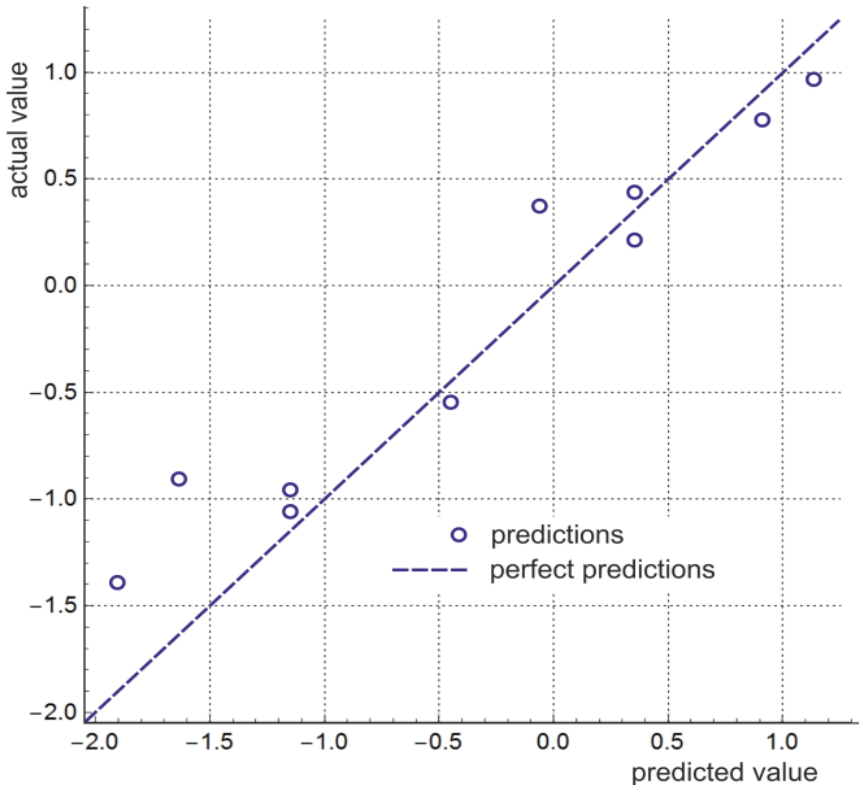


Рис. 3.47. Помилка передбачення на тестовому наборі

Зазвичай вимірюють ефективність передикторної функції *f*(X): обчислення оцінки предиктора,

$$\mathrm{Score}(f) = 1 - \frac{\sum_{i=1}^{n}\left(y_i - f(X_i)\right)^2}{\sum_{i=1}^{n}\left(y_i - \mathrm{Mean}(y_i)\right)^2}.$$

Чим ближче оцінка до 1,00, тим кращий прогноз. Це можна визначити як для навчального, так і для тестового набору. У нашому випадку для тестового набору:

```
⇒ u=Total[Map[#^2&,ytest-Map[f[#]&,Flatten[Xtest]]]]
⇐ 1.08677
```

і

```
⇒ v=Total[Map[#^2&,ytest-Mean[ytest]]]
⇐ 6.56328
```

Таким чином, оцінка тестового набору:

```
⇒ 1-u/v
⇐ 0.834417
```

аналогічно для навчального набору,

```
⇒ u=Total[Map[#^2&,ytrain-Map[f[#]&,Flatten[Xtrain]]]]
⇐ 6.32668
```

і

```
⇒ v=Total[Map[#^2&,ytrain-Mean[ytrain]]]
⇐ 35.0381
```

Далі отримуємо оцінку навчального набору даних:

```
⇒ 1-u/v
⇐ 0.819434
```

Ці дві оцінки близькі одна до одної, тому прогноз вірний.

3.10. Реконструкція поверхні

Розглянемо задачу із реконструкції поверхні. Розглянемо загальну поверхню другого порядку, див. Рис. 3.48.

```
⇒ t[x_,y_]:=ax+by+cx^2+dxy+ey^2
```

Використовуючи певні значення параметрів, ми створюємо синтетичний набір даних,

```
⇒ p1=Plot3D[t[x,y]/.{a → 1,b->11,c → 3,d → -4,
    e → -1},{x,-10,10},{y,-10,10},Mesh → 8,ColorFunction → Hue,
    MeshShading → {{Yellow,Orange},{Pink,Red}},BoxRatios → {1,1,1}]
```

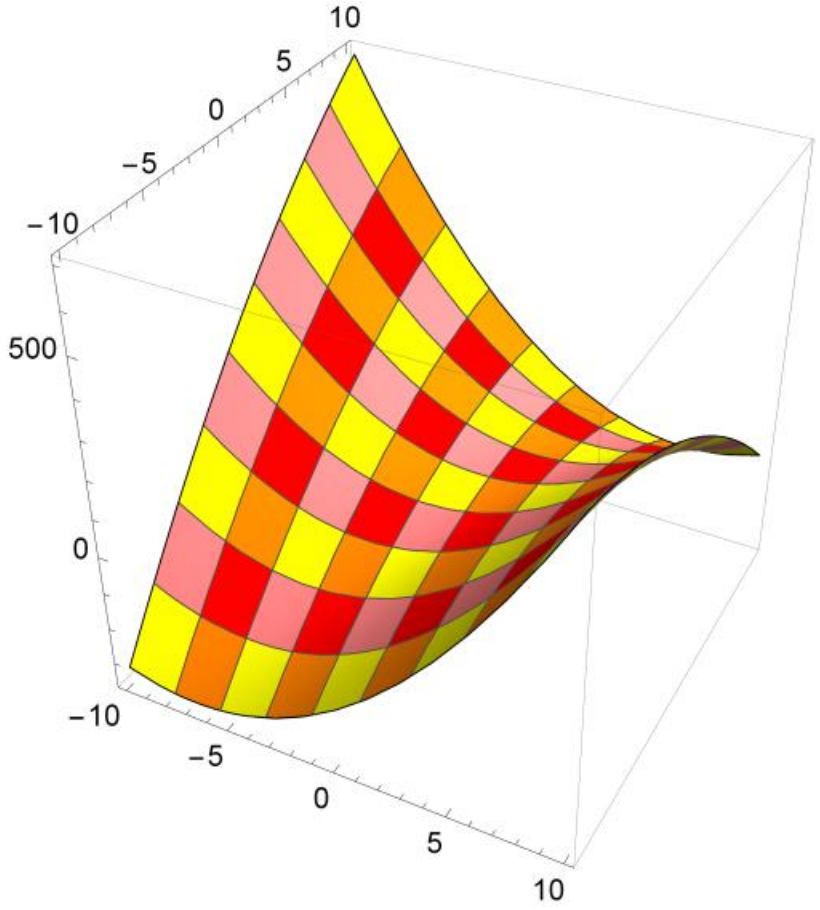


Рис. 3.49. Початкова поверхня

Давайте скористаємося значеннями даних виміряної функції області [-10,10]×[-10, 10] растру [x, y] з нормальною випадковою помилкою $N(0,30)$. Гістограму помилок подамо далі на Рис. 3.50.

```
⇒ Lerror=RandomVariate[NormalDistribution[0,30],{21,21}];
⇒ Histogram[Flatten[Lerror,1]]
```

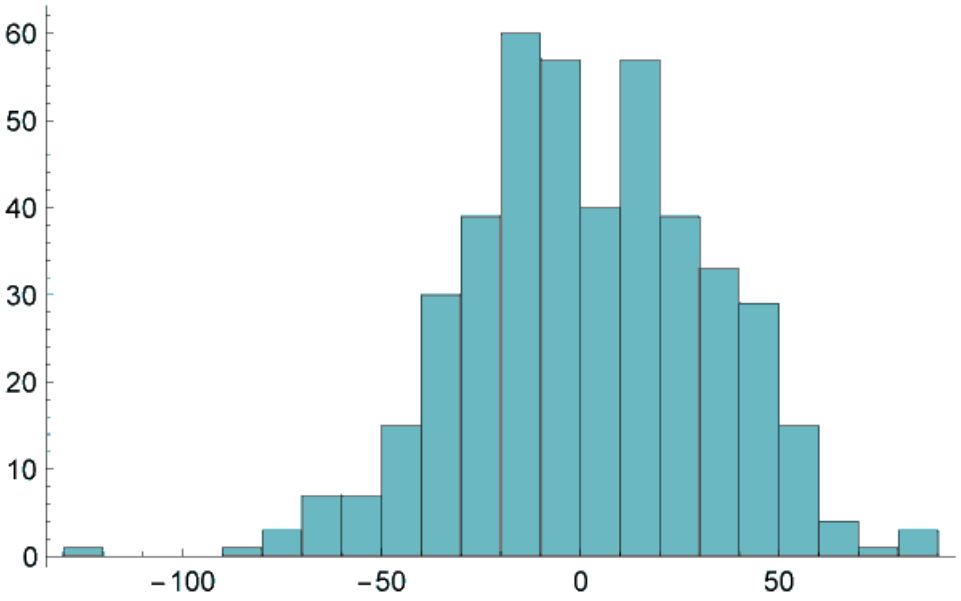


Рис. 3.50. Гістограма випадкової похибки

Тоді «виміряні» дані:

```
⇒ dataPoints=Flatten[Table[{i,j,Lerror[[i+11,j+11]]+t[i ,j]/.
    { → 1,b → 11,c → 3,d → -4,e → -1}},{i,-10,10},{j,-10,10}],1];
```

На Рис. 3.51 дані згенеровані зашумлені точки даних з вихідною поверхнею.

```
⇒ p2=ListPointPlot3D[dataPoints,BoxRatios → {1, 1, 1},
    PlotStyle → Blue];
⇒ Show[{p1,p2}]
```

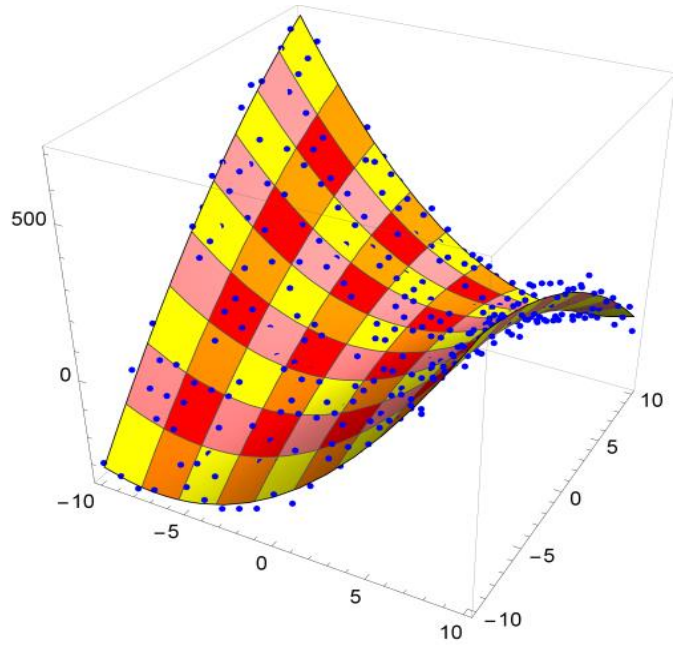


Рис. 3.51. Числові дані на поверхні

Тепер відновимо вихідну поверхню на основі цих зашумлених точок даних. Щоб використати лінійну модель, ми перетворюємо змінні:

```
⇒ transform[r_,s_]={r,s,r s,r^2,s^2}
⇐ {r,s,rs,r^2,s^2}
```

Таким чином, замість використання нелінійної моделі із двома змінними ми тепер маємо лінійну модель із п'ятьма змінними. Використовуючи Mathematica, набір навчальних даних:

```
⇒ ySet=Table[transform[i,j],{i,-10,10},{j,-10,10}];
⇐ trainingdata=
    MapThread[#1 → #2[[3]]&,{Flatten[ySet,1],dataPoints}];

⇒ trainingdata[[1]]
⇐ {-10,-10,100,100,100} → -303.836
```

Давайте тепер скористаємося K-регресією найближчих сусідів з k= 3,

```
⇒ c=Predict[trainingdata,
    Method → {"NearestNeighbors","NeighborsNumber" → 3}]
⇐ PredictorFunction[ Input type: Mixed (Number:5)
                     Method: NearestNeighbors ]
```

Цю функцію можна застосувати до вихідного набору даних, наприклад:

```
⇒ c[transform[1,1]]
⇐ 20.1957
```

На Рис. 3.52 подана апроксимована поверхня:

```
⇒ p3=Plot3D[c[transform[x,y]],{x,-10,10},{y,-10,10},
    BoxRatios → {1, 1, 1}]
```

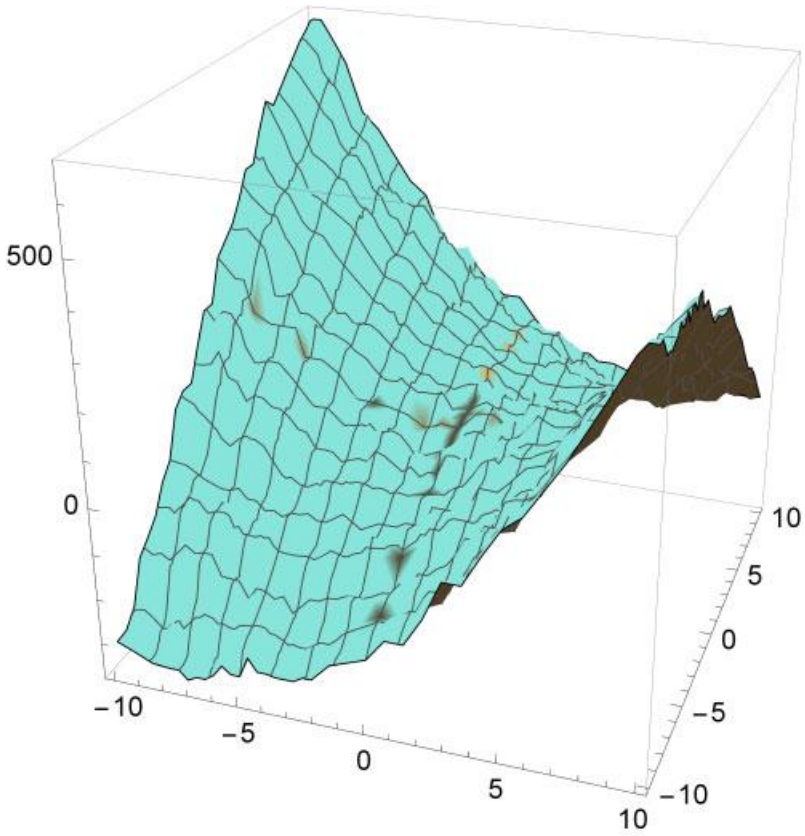


Рис. 3.52. Наближення найближчих сусідів

Ми можемо порівняти апроксимацію та вихідну поверхню,

```
⇒ Show[{p1,p3}]
```

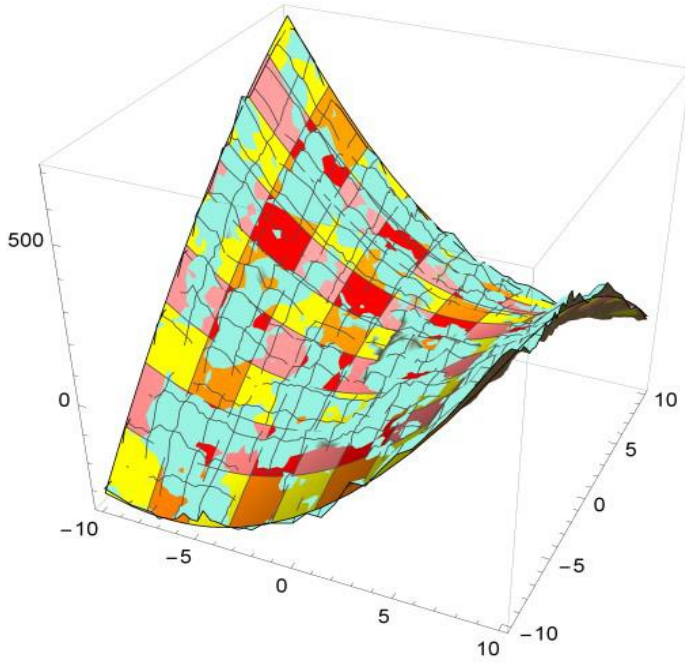


Рис. 3.53. Апроксимація та вихідне зображення поверхні

Гістограму розподілу похибок апроксимації є такою як на Рис. 3.54.

```
⇒ error=Table[(t[x,y]/.{a → 1,b → 11,c → 3,d → -4,e → -1})-
    c[transform[x,y]],{x,-10,10},{y,-10,10}];
⇒ Histogram[Flatten[Error,1]]
```

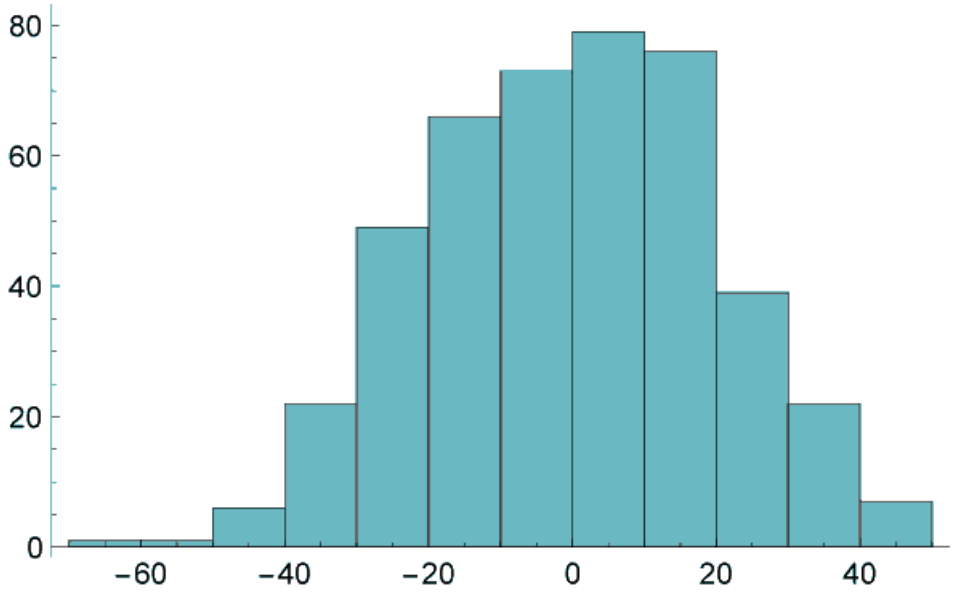


Рис. 3.54. Гістограма помилок апроксимації

3.11. Границі кільця Сатурна як задача нелінійної регресії

Цей приклад ілюструє, що іноді можна уникнути регресії і ми можемо використовувати інтерполяцію як окремий випадок регресії. Це можна зробити, виключивши "неправильні" дані, так звані викиди. Для цього ми розглянемо знімок космічного корабля Кассіні в 2016 році, що показує північну півкулю Сатурна. Зображення надано НАСА/Лабораторією реактивного руху. Каліфорнійський технологічний інститут/Інститут космічних наук.

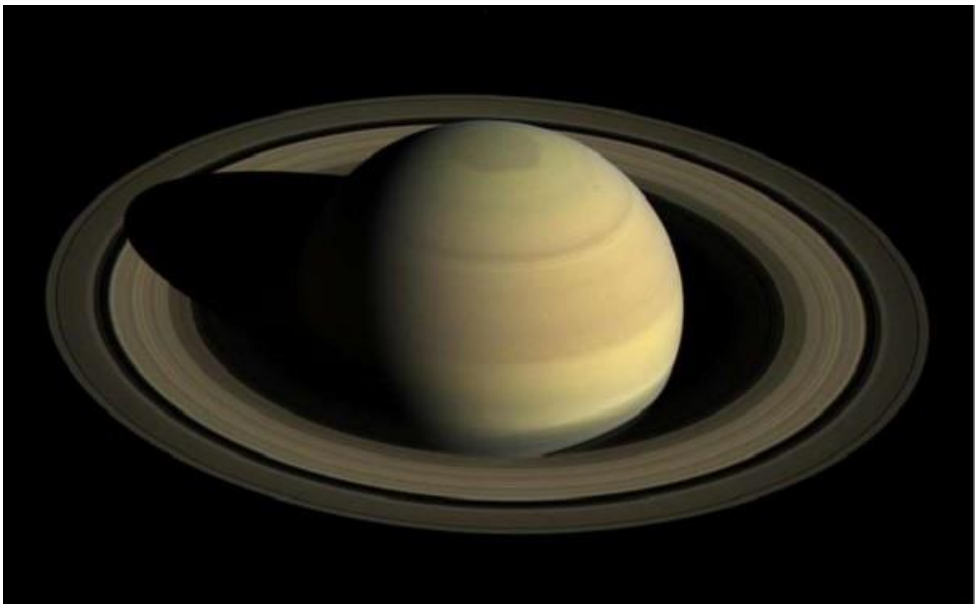


Рис. 3.55. Північна півкуля Сатурна. Зображення надано НАСА/Лабораторією реактивного руху. Каліфорнійський технологічний інститут/Інститут космічних наук

Використовуючи це зображення, ми хотіли б вгадати зовнішню межу кілець, наклавши на неї криву. Щоб отримати деякі точки кордону, спочатку видалімо фон зображення, див. Рис. 3.56.

```
⇒ img1=RemoveBackground[img]
```

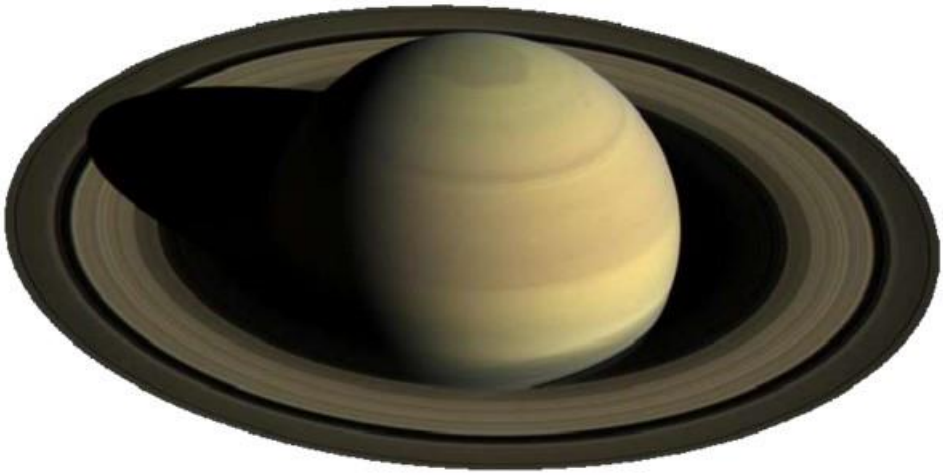


Рис. 3.56. Зображення без фону

Потім ми шукаємо ключові точки цього зображення.

```
⇒ corners=ImageCorners[img1,MaxFeatures → 40];
⇒ hu=HighlightImage[img1,corners]
```

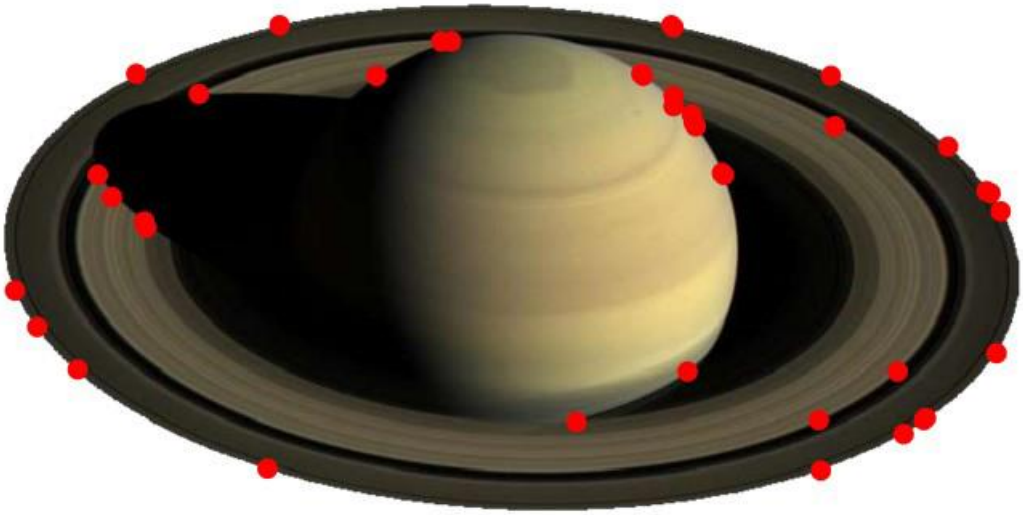


Рис. 3.57. Зображення з ключовими точками

Щоб вибрати точки на периметрі, ми обчислюємо опуклу оболонку точок:

```
⇒ Needs["ComputationalGeometry`"]
```

Вбудована функція забезпечить нумерацію кутових точок.

```
⇒ convexhull=ConvexHull[corners]
⇐ {34,27,7,20,19,11,15,32,40,37,9,13,23,18,35}
```

Координати цих точок:

```
⇒ pp=Map[corners[[#]]&,convexhull]
⇐ {{746.5,271.5},{738.5,284.5},{708.5,317.5},{625.5,367.5},
    {511.5,403.5},{232.5,403.5},{130.5,368.5},{44.5,214.5},
    {60.5,188.5},{88.5,158.5},{224.5,88.5},{618.5,87.5},
    {676.5,113.5},{691.5,122.5},{742.5,170.5}}
```

Отже, ми вибрали периметричні точки

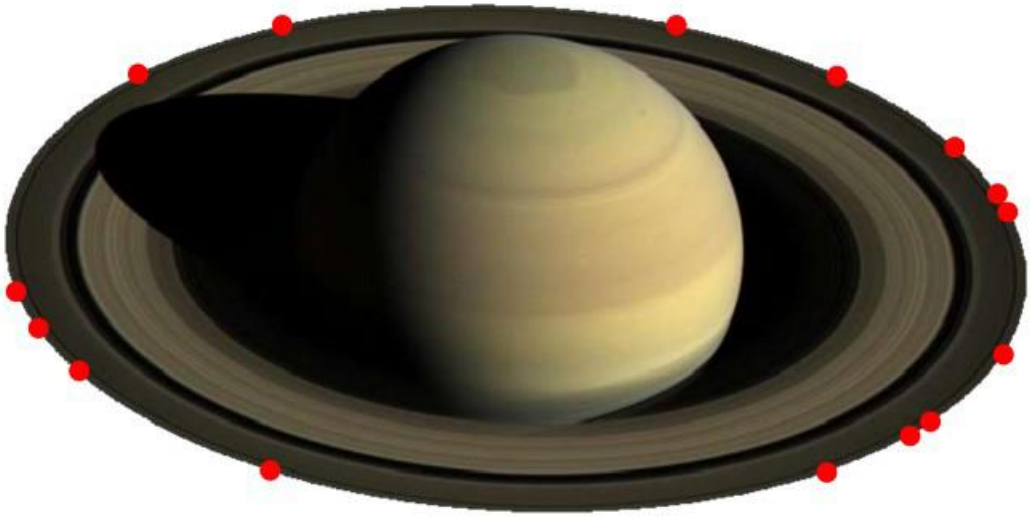


Рис. 3.58. Периметричні точки.

Підженемо еліпс до периметричних точок. Загальна форма еліпса,

$$Ax^2 + B\,x\,y + Cy^2 + D\,x + E\,y + F = 0$$

Використовуючи Mathematica будемо мати для параметрів еліпса:

```
⇒ lin=#1^2,#1,#2,2#1#2,#2^2&@@@pp;
⇒ lm=LinearModelFit[lin,{1,a,b,c,d},{a,b,c,d}]
⇐ FittedModel[-71352.4-0.246181a+214.937b+507.896c-0.0396957d]
⇒ lm["ParameterTable"]
```

```
⇒ lin=#1^2,#1,#2,2#1#2,#2^2&@@@pp;
⇒ lm=LinearModelFit[lin,{1,a,b,c,d},{a,b,c,d}]
⇐ FittedModel[-71352.4-0.246181a+214.937b+507.896c-0.0396957d]
⇒ lm["ParameterTable"]
⇐
```

	Estimate	*Standard Error*	*t-Statistic*	*P-Value*
1	-71352.4	99.228	-719.076	6.65759×10^{-25}
a	-0.246181	0.000555238	-443.379	8.38011×10^{-23}
b	214.937	0.460803	466.439	5.04728×10^{-23}
c	507.896	0.39964	1270.88	2.2389×10^{-27}
d	-0.0396957	0.000394002	-100.75	2.27349×10^{-16}

```
⇒ pa=lm["BestFitParameters"];
⇒ w[x_,y_]:=pa.{1,x^2,x,y,2xy}-y^2;
⇒ cc=ContourPlot[w[x,y]==0,{x,0,800},{y,0,500},
    ContourStyle → {Thick,Red},Epilog → Point[points]];
```

Отримаємо рівняння нашої моделі:

```
⇒ TraditionalForm[-w[x,y]==0]
⇒ 0.246181x^2+0.0793913xy-214.937x+y^2-507.896y+71352.4==0
⇒ Show[{img1,cc}]
```

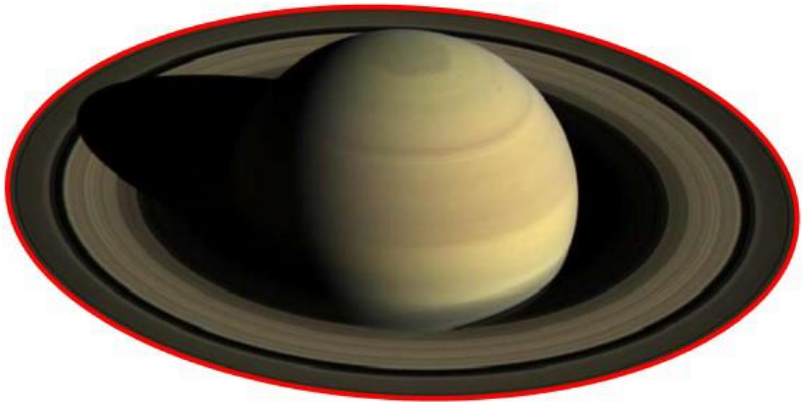


Рис. 3.59. Передбачувана лінія меж кілець Сатурна

Як альтернатива замість припасування еліпса до цих точок ми демонструємо більш загальний метод оцінки інших точок периметра, оскільки альтернативним рішенням може бути параметрична регресія або, у нашому випадку, інтерполяція. Ми шукаємо функції:

$$x = x(t)$$

$$y = y(t)$$

де $(x(t), y(t))$ координати точки периметра. Використовуватимемо довжину лінійних ліній, що з'єднують сусідні точки опуклої оболонки:

```
⇒ HighlightImage[img1,PlanarGraphPlot[corners,convexhull]]
```

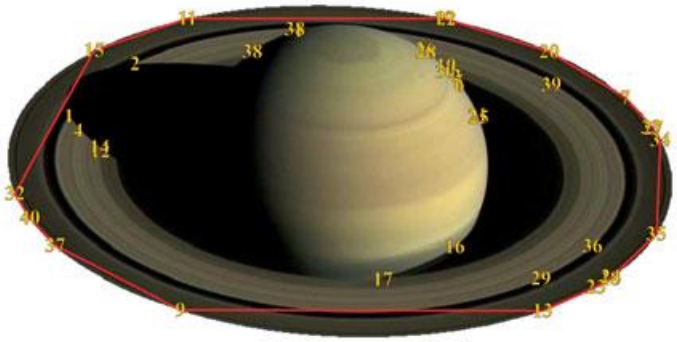


Рис. 3.60 Периметричні точки

Обчислимо значення цього параметра, що належать різним точкам. Закриваючи контур маємо:

```
⇒ data=Join[pp,{First[pp]}];
```

ці параметри такі:

```
⇒ n=Length[data];
⇒ t={0};
    Do[t=AppendTo[t,Last[t]+Norm[data[[i+1]]-data[[i]]]],{i,1,n-1}]
```

Передавши ці параметри координатам x_i та y_i, ми отримуємо списки (x_i, t_i) та (y_i, t_i),

```
⇒ dataxt=MapThread[{#1,#2[[1]]}&,{t,data}];
⇒ datayt=MapThread[{#1,#2[[2]]}&,{t,data}];
```

Тепер ми використовуємо інтерполяцію замість регресії для $x(t)$ та $y(t)$:

```
⇒ xatInt=Interpolation[dataxt,InterpolationOrder → 3];
⇒ yatInt=Interpolation[datayt,InterpolationOrder → 2];
```

Потім ми можемо використовувати параметричний графік візуалізації зовнішньої границі.

```
⇒ par=ParametricPlot[{xatInt[u],yatInt[u]},{u,0,Max[t]},
    PlotStyle → {Thick,Red}];
⇒ Show[{hu,par}]
```

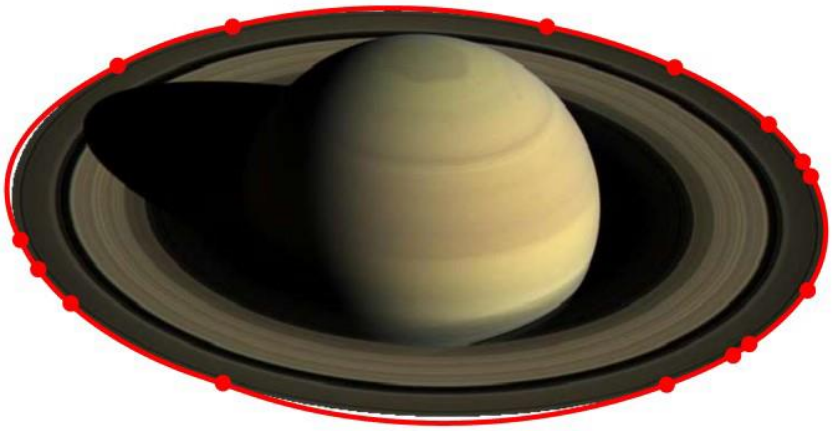


Рис. 3.61. Передбачувана лінія меж кілець Сатурна

Інтерполяція не вимагає припущення про багатоточковість вхідного масиву даних.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Зараз у нашій країні проводиться розробка національних нормативних документів, спрямованих на охорону праці користувачів ПК. Найбільш повним нормативним документом щодо забезпечення охорони праці користувачів ПК є «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин» (ДСПіН 3.3.2.007–98.) [5].

Особлива увага звертається на те, що дотримання вимог, викладених в Правилах, значно знизить наслідки несприятливої дії на працівників шкідливих та небезпечних факторів, які супроводжують роботу з відеодисплейними терміналами. У Правилах викладені гігієнічні й ергономічні вимоги до організації робочих приміщень та робочих місць, параметрів робочого середовища, дотримання яких дає змогу запобігти порушенням стану здоров’я користувачів ПК.

Користувачі ПЕОМ постійно працюють з обладнанням, яке підключено до електропостачання. При організації робочого місця потрібно ретельно перевіряти безпечність підключення до електромереж.

На користувачів під час роботи з комп'ютерною технікою можуть діяти такі види небезпек:

- ураження електричним струмом;
- енергетична небезпека (виникає через коротке замикання: опіки, електрична дуга, викид розплавленого металу);
- небезпека займання;
- термонебезпека (дія високих температур через нагрівання конструктивних елементів);
- механічна небезпека (травми через падіння, дії рухомих частин, поріз про гострі частини конструктивних елементів);

- небезпека випромінювання (дія звукового (акустичного), високочастотного, інфрачервоного, ультрафіолетового та іонізуючого випромінювання, а також видимого світла високої інтенсивності когерентної (лазерного випромінювання);

- хімічна небезпека (контакт із деякими хімікатами, які використовують для того, щоб обслуговувати обладнання, або від вдихання їхньої пари).

У процесі експлуатації електрообладнання може статися порушення цілісності ізоляції проводів, кабелів, обмоток машин та інших, які перебувають під напругою струмоведучих частин тобто, відбувається замикання струмовідних частин на землю. Це неминуче тягне у себе поява напруги на нетоковедущих частинах устаткування й у результаті працівник виявляється під впливом електричного струму, що може призвести до нещасного випадку.

Електробезпека обслуговуючого персоналу забезпечується системою організаційних та технічних заходів та засобів, що забезпечують захист людей від шкідливого та небезпечного впливу електричного струму, електричної дуги, електромагнітного поля та статичної електрики. Одним із засобів захисту персоналу при спробі ізоляції є застосування захисного заземлення.

Замиканням на землю називається випадкове електричне з'єднання з землею, що знаходяться під напругою електроустановок. Замикання на землю може статися внаслідок появи контакту між струмовідними частинами та заземленим корпусом або конструктивними частинами обладнання при падінні на землю обірваного дроту, при порушенні ізоляції обладнання і т.д. У всіх цих випадках струм від частин, що знаходяться під напругою, проходить у землю через електроди, які здійснюють контакт із ґрунтом. Спеціальні металеві електроди прийнято називати заземлювачами.

Захисне заземлення – це навмисне з'єднання із землею за допомогою заземлювального пристрою металевих невідповідних частин обладнання, які можуть опинитися під напругою внаслідок порушення ізоляції електроустановки.

Відповідно до СанПіН 2.2.2/2.4.1340-03 «Гігієнічні вимоги до персональних електронно-обчислювальних машин та організації роботи» приміщення, де

розміщуються робочі місця з ПЕОМ, повинні бути обладнані захисним заземленням (зануленням) відповідно до технічних вимог щодо експлуатації електроустановок та обчислювальної техніки.

Відповідно до цього нормативного документа заземлювальні пристрої повинні бути обрані та змонтовані таким чином, щоб:

– значення опору розтіканню заземлювального пристрою відповідало вимогам захисту та роботи установки протягом періоду експлуатації;

– протікання струму замикання на землю і струмів витоку не створювало небезпеки, зокрема щодо нагрівання, термічної та динамічної стійкості;

– було забезпечено необхідну міцність або додатковий захист залежно від заданих зовнішніх факторів.

Робочі місця з ПЕОМ не слід розміщувати поблизу силових кабелів та вводів, високовольтних трансформаторів, технологічного обладнання, що створює перешкоди у роботі ПЕОМ.

Оскільки безпосередньо на ПЕОМ має подаватися стабілізоване електроживлення (з відхиленням від 220 В не більше -10% +15%), подачу електроенергії до комп'ютерних приміщень слід здійснювати від окремого незалежного джерела живлення.

Зазвичай підключення ПЕОМ здійснюється через блок живлення або пристрій живлення, що мають мережевий фільтр, конденсатори якого призначені для шунтування через провід занулення, і відповідні трисмугові вилку і розетку високочастотних перешкод мережі живлення на землю.

У цьому випадку до розетки повинні бути підключені три дроти: один фазний, другий нульовий робочий провідник і третій нульовий захисний провідник (НЗП). Нульовий захисний провідник необхідно з'єднувати з нульовим дротом мережі.

В іншому випадку (якщо НЗП нікуди не підключати), на корпусі системного блоку може з'явитися напруга близько 110 змінного струму. Це станеться тому, що конденсатори фільтра працюють як ємнісний дільник напруги.

Оскільки ємності конденсаторів мають однакові значення, напруга мережі 220 В розділиться навпіл і, якщо врахувати, що середній опір тіла людини-1000 Ом, а опір статі (дерев'яного) та взуття близько-330 Ом, то сила струму, що проходить через тіло людини, становитиме 83 мА. При цьому можливе настання паралічу дихання [6].

Надалі при експлуатації ПЕОМ необхідно дотримуватись наступних рекомендацій:

- постійно контролювати надійність з'єднання контактів трипровідних розеток;

- додатково підключити системний блок до НЗП, наприклад закріпити провідник під гвинт кріплення джерела живлення;

- підключати дисплей (за наявності лише двопровідної однофазної мережі) рекомендується через узгоджувальний пристрій. При цьому мережні фільтри та всі кабелі живлення повинні знаходитися якнайдалі від оператора в компактному положенні з тильного боку робочого місця;

- не підключайте корпус комп'ютера до парового або водяного опалення. Якщо джерело живлення комп'ютера несправне, батареї можуть опинитися під напругою;

- не ставити системний блок у зоні підвищеної вологості та підвищеного вмісту пилу, на підлогу, біля ніг оператора;

- не можна торкатися одночасно екрана монітора та клавіатури (можливий підвищений електростатичний потенціал);

- щоб уникнути ураження електричним струмом, забороняється торкатися задній панелі системного блоку і перемикати роз'єми периферійних пристроїв працюючого комп'ютера;

- необхідно встановлювати ПЕОМ (ПК) тільки на жорстко закріпленій підставці, яка виключає навіть випадковий струс системного блоку;

- не рекомендується встановлення ПЕОМ та його клавіатури на поверхні, що накопичують статичну електрику (органічне скло та поліровані лакові поверхні);

- температура повітря в приміщенні допускається в межах 20-25 ° С при відносній вологості до 75%; різкі перепади температури не допускаються;

- не допускається зайва запиленість повітря в приміщенні (не більше 1 мг/м3 при максимальному розмірі частинок 3 мкм); обов'язкове вологе щоденне прибирання приміщення;

- необхідно щодня протирати вологою серветкою екран, приекранний фільтр, клавіатуру та інші частини ПЕОМ.

4.2. Безпека в надзвичайних ситуаціях

Приміщення із робочими місцями користувачів комп’ютерів для забезпечення електробезпеки обладнання, а також для захисту від ураження електричним струмом самих користувачів ПК повинні мати достатні технічні засоби захисту відповідно до ГОСТ 12.1.009-76, НПАОП 40.1-1.07-01 “Правила експлуатації електрозахисних засобів”, НПАОП 40.1-1.21-98 “Правила безпечної експлуатації електроустановок споживачів”, НПАОП 40.1-1.32-01 “Правила будови електроустановок. Електрообладнання спеціальних установок”.

З метою запобігання ушкодженням, що можуть статися через ураження електричним струмом, загоряння, коротке замикання тощо, розроблено загальний стандарт безпеки IEC 950. Загальним стандартом електробезпечності для країн Європейської співдружності є Cemark.

Під час проектування систем електропостачання, монтажу силового електрообладнання та електричного освітлення будівель та приміщень для ПЕОМ необхідно дотримуватись вимог вищеназваних нормативно-правових актів, а також Правил пожежної безпеки в Україні, ДСанПіН 3.3.2.007-98, розділів ДБН, що стосуються штучного освітлення і електротехнічних пристроїв, та вимог нормативно-технічної і експлуатаційної документації заводу-виробника ПЕОМ.

ЕОМ, периферійні пристрої ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ, інше устаткування (апарати управління, контрольно-вимірювальні прилади, світильники тощо), електропроводи та кабелі за виконанням та ступенем захисту мають відповідати класу зони за ПУЕ, мати апаратуру захисту від струму короткого замикання та інших аварійних режимів.

Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, перейти на негорючу ізоляцію.

Лінія електромережі для живлення ЕОМ, периферійних пристроїв ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ виконується як окрема групова трипровідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів.

Використання нульового робочого провідника як нульового захисного провідника забороняється. Нульовий захисний провід прокладається від стійки групового розподільчого щита, розподільчого пункту до розеток живлення. Не допускається підключення на щиті до одного контактного затискача нульового робочого та нульового захисного провідників. Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі повинна бути не менше площі перерізу фазового провідника.

Усі провідники повинні відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам ПУЕ.

У приміщенні, де одночасно експлуатується або обслуговується більше п'яти персональних ЕОМ, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

ПЕОМ, периферійні пристрої ПЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ повинні підключатися до електромережі тільки з допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників повинні мати спеціальні контакти для підключення нульового захисного провідника. Конструкція їх має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше ніж

приєднання фазового та нульового робочого провідників. Порядок роз'єднання при відключенні має бути зворотним. Необхідно унеможливити з'єднання контактів фазових провідників з контактами нульового захисного провідника.

Неприпустимим є підключення ПЕОМ та периферійних пристроїв ПЕОМ до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв.

Електромережі штепсельних з'єднань та електророзеток для живлення ПЕОМ, периферійних пристроїв слід виконувати за магістральною схемою, по 3…6 з'єднань або електророзеток в одному колі. Штепсельні з'єднання та електророзетки для напруги 12 В та 36 В за своєю конструкцією повинні відрізнятися від штепсельних з'єднань для напруги 127 В та 220 В і мають бути пофарбовані в колір, який візуально значно відрізняється від кольору штепсельних з'єднань, розрахованих на напругу 127 В та 220 В.

Індивідуальні та групові штепсельні з'єднання та електророзетки необхідно монтувати на негорючих або важкогорючих пластинах з урахуванням вимог ПУЕ та Правил пожежної безпеки в Україні.

Електромережу штепсельних розеток для живлення ПЕОМ, периферійних пристроїв ПЕОМ при розташуванні їх уздовж стін приміщення прокладають по підлозі поряд зі стінами приміщення, як правило, в металевих трубах і гнучких металевих рукавах з відводами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання.

При розташуванні в приміщенні за його периметром до 5 ПЕОМ, використанні трипровідникового захищеного проводу або кабелю в оболонці з негорючого або важкогорючого матеріалу дозволяється прокладання їх без металевих труб та гнучких металевих рукавів.

Електромережу штепсельних розеток для живлення ПЕОМ при розташуванні їх у центрі приміщення, прокладають у каналах або під знімною підлогою в металевих трубах або гнучких металевих рукавах. При цьому не дозволяється застосовувати провід і кабель в ізоляції з вулканізованої гуми та інші матеріали, що містять сірку. Відкрита прокладка кабелів під підлогою

забороняється. Металеві труби та гнучкі металеві рукави повинні бути заземлені. Заземлення повинно відповідати вимогам НПАОП 40.1-1.21-98 [7].

Для підключення переносної електроапаратури застосовують гнучкі проводи в надійній ізоляції.

Тимчасова електропроводка від переносних приладів до джерел живлення виконується найкоротшим шляхом без заплутування проводів у конструкціях машин, приладів та меблях. Доточувати проводи можна тільки шляхом паяння з наступним старанним ізолюванням місць з'єднання.

Є неприпустимими:

– експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості за час експлуатації, ізоляцією; залишення під напругою кабелів та проводів з неізольованими провідниками;

– застосування саморобних подовжувачів, які не відповідають вимогам ПВЕ до переносних електропроводок;

– застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання;

– користування пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має сліди затемнення або випинання;

– підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною та іншими горючими матеріалами, експлуатація їх зі знятими ковпаками (розсіювачами);

– використання електроапаратури та приладів в умовах, що не відповідають вказівкам (рекомендаціям) підприємств-виготовлювачів.

ВИСНОВКИ

Основним результатом, отриманим в магістерській роботі є розроблена загальна методологія створення складних програмних систем на базі синтезу методів близькоспоріднених мов програмування Python та Wolfram Mathematica й застосування отриманих результатів в різноманітних напрямках науки про дані та інженерії даних.

Безпосередньо в результаті виконання завдань зумовлених метою магістерської роботи було розроблено групу методів, що забезпечують ефективну реалізацію гібридної візуалізації, кластеризації, інтерполяції та регресії різнорозмірних масивів даних. Встановлено, що застосування вбудованих функцій Python та Wolfram Mathematica дозволяє отримати переваги й оптимізувати роботу з даними ніж при самостійному застосуванні цих мов та середовищ програмування.

Для імплементації розроблених методів використовувались мови програмування Python та Wolfram Mathematica та середовища Spider й Wolfram Cloud. Для тестування розроблених системи використовувались засоби WolframUnit.

Отримані результати будуть корисними як для програмістів працюючих в галузі науки про дані так науковців сфера роботи яких пов’язана з математичним моделюванням складних систем, ідентифікацією параметрів компонент та процесів, суміжних галузях технологій, пов’язаних з візуальною інтерпретацією та аналізом отримуваних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Banerjee T (2018): Deep Learning and Computer Vision: Converting Models for the Wolfram Neural Net Repository, Wolfram Blog, https://blog.wolfram.com/2018/12/06/deep-learning-andcomputer- vision-converting-models-for-the-wolfram-neural-net-repository/
2. Brownlee J. (2016): Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras, https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neuralnetworks- python-keras/
3. Chollet F (2017): Keras: The Python Deep Learning library, Keras Documentation, https://keras.io/
4. Chollet F (2018): Deep Learning with Python, Manning Publications Co., Shelter Island, NY Deshpande M (2017): Perceptrons: The First Neural Networks, https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/
5. Freeman J A (1994): Simulating Neural Networks with Mathematica, Addison-Wesley, Massachusetts
6. Haykin S. (2009): Neural Networks and Learning Machines,Third Edition,Prentice Hall,New York, London, Sydney,
7. Hu X, Yuan Y (2016): Deep-Learning-Based Classification for DTM Extraction from ALS Point Cloud, Remote Sens. 2016, 8, p.730
8. Melnikov O (2017): Trouble fitting simple data with MLPRegressor, stackoverflow https://stackoverflow.com/questions/41069905/trouble-fitting-simple-data-with-mlpregressor
9. Raschka S (2018): Neural Network - Multilayer Perceptron, http://rasbt.github.io/mlxtend/user_guide/classifier/MultiLayerPerceptron/
10. Redmon J (2017) Mathematica implementation of YOLO, a computer vision object detection mode Object detection and localization using neuralnetwork,

Mathematica Stack Exchange Network https://mathematica.stackexchange.com/questions/141598/object-detection-and-localizationusing- neural-network

11. Rohrer B (2017): Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM), https://www.youtube.com/watch?v=WCUNPb-5EYI
12. Sharma A (2017): Convolutional Neural Networks in Python with Keras,DataCamp https://www.google.hu/search?q=convolution+network+in+Python&oq=convolution+network+ in+Python&aqs=chrome..69i57j0l3.14671j0j7&sourceid=chrome&ie=UTF-8
13. Shevchuk Y (2015): Discrete Hopfield Network - NeuPy http://neupy.com/2015/09/20/discrete_hopfield_network.html
14. Shevchuk Y (2017): Self-Organizing Map and Applications, NeuPy, Neural Networks in Python, http://neupy.com/2017/12/09/sofm_applications.html
15. Srinivasan J, Han YK and Ong SH (1993): Image reconstruction by a Hopfield neural network, Image and Vision Computing 11(5), pp. 279-282 https://www.sciencedirect.com/science/article/pii/0262885693900052
16. Sullivan J (2017): Neural Network from Scratch: Perceptron Linear Classifier, https://jtsulliv.github.io/perceptron/
17. Usama M et al (2017): Unsupervised Machine Learning for Networking:Techniques, Applications and Research Challenges. https://arxiv.org/pdf/1709.06599.pdf
18. Saputro DRS and Widyaningsih P (2017): Limited Memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) Method for The Parameter Estimation on Geographically Weighted Ordinal Logistic Regression Model (GWOLR), AIP Conference Proceedings 1868, 040009 (2017); https://doi.org/10.1063/1.4995124 Published Online: 04 August 2017 https://aip.scitation.org/doi/pdf/10.1063/1.4995124?class=pdf

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку кваліфікаційної роботи

«Розробка засобів гібридної візуалізації, класифікації, кластеризації та регресії в середовищі Wolfram Mathematica з використанням мови програмування Python»

Розробники:

виконавець гр. СПм-61

Войтович Ростислав Вікторович

_____

(підпис)

керівник кваліфікаційної роботи

Бойко Ігор Володимирович

_____

(підпис)

Тернопіль 2023

ЗМІСТ

1. ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю 121 «Інженерія програмного забезпечення».

Тема проекту: «на тему: «Розробка засобів гібридної візуалізації, класифікації, кластеризації та регресії в середовищі Wolfram Mathematica з використанням мови програмування Python».

Термін виконання: до «25» січня 2023р.

2. ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Інформаційна система призначена для фахівців, що працюють з даними: інженерів даних та програмістів, що працюють в сфері науки про дані.

Інформаційна система буде корисною в сферах статистичної обробки інформації, машинного навчання, нейромаркетингу.

Інформаційна система створена для покращення ефективності роботи в процесі обробки масивів даних, відшукання кореляції в них та статистичного прогнозування роботи систем.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система повинна передбачати ролі таких користувачів:

- користувач

Роль користувача визначається за результатами проходження процедури аутентифікації в системі Wolfram Mathematica і Wolfram Cloud.

Для авторизованих користувачів системи повинна надавати доступ до наступного переліку функцій:

- створення, зміна;
- перегляд списку розроблених моделей;
- редагування персональної інформації;
- додавання параметрів моделей;
- видалення параметрів моделей;
- створення зв’язків між методами і моделями;
- робота із скомпільованими програмними системами;

- внесення змін у параметри готових моделей;
- можливість редагування коду програмних систем;

Для неавторизованих користувачів система надає доступ до функцій:

- робота із скомпільованими програмними системами;
- внесення змін у параметри готових моделей;

3.2 Технічні вимоги

Вимоги до серверної частини: ОС Windows, ОС Linux, не менше ніж 8Гб ОЗП;

Вимоги до клієнтської частини: веб – браузер, система Wolfram Mathematica, систем Spider;

Додаткові вимоги: наявне підключення до мережі Інтернет, автоматичне резервування, синхронізація із хмарними ресурсами.

3.3 Програмні вимоги

Використання СУБД: MySQL

Розробка структурної частини: блок схеми Wolfram Mathematica

Розробка клієнтської частини: Python, Wolfram Mathematica, C#

Додаткові вимоги: можливість інтегрування системи у Wolfram Cloud.

4 ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться в наступному порядку:

- аналіз предметної області, виявлення акторів та варіантів використання системи;
- вибір засобів розробки та архітектури системи;
- проектування бази даних системи;
- розробка програмного забезпечення системи;
- тестування інформаційної системи на реальних даних;
- оформлення супровідної документації;
- здача проекту.

Результати виконання кожного етапу проекту погоджуються з керівником проекту.

5 СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- пояснювальна записка до проекту;
- презентація проекту;
- рецензія на проект;
- диск з проектом.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6 ПОРЯДОК ЗДАЧІ ПРОЕКТУ

Розроблена інформаційна системи повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	Виконано
Архітектура системи	Виконано
Проектування база даних	Виконано
Використання системи	Виконано
Супровідна документація	Виконано

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

Публікація в науковому виданні

ДОДАТОК В

Лістинг коду

```
from __future__ import print_function

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics
 import silhouette_samples,silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np


# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and
  3 clusters placed close
# together.
X, y = make_blobs(n_samples=500,
                  n_features=2,
                  centers=4,
                  cluster_std=1,
                  center_box=(-10.0, 10.0),
                  shuffle=True,
                  random_state=1)  # For reproducibility

range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)


  # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range
      from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space
      between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

  # Initialize the clusterer with n_clusters value and
    a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for
      all the samples.
    # This gives a perspective into the density and
      separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)
```

```
    # Compute the silhouette scores for each sample
    sample_silhouette_values =
      silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples
          belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i=ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                 0, ith_cluster_silhouette_values,
                 facecolor=color, edgecolor=color, alpha=0.7)


        # Label the silhouette plots with their cluster
          numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i,str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title(
        "The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of
      all the values
    ax1.axvline(x=silhouette_avg,color="red",linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype
             (float) / n_clusters)
    ax2.scatter(X[:,0],X[:,1],marker='.',s=30,lw=0,alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')


    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')
```

УДК 004.8
Р. Войтович, М. Петрик, докт. фіз.-мат. наук, проф .
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ ВИРІШЕННЯ ЗАДАЧ КЛАСИФІКАЦІЇ БІООБ'ЄКТІВ НА ЗОБРАЖЕННЯХ

UDC 004.8
R. Voytovych, M. Petryk, Dr. Prof

APPLICATION OF NEURAL NETWORKS TO SOLVE THE PROBLEMS OF CLASSIFICATION OF BIOOBJECTS IN IMAGES

Нейронні мережі – серія алгоритмів, які намагаються розпізнати основні взаємозв'язки у наборі даних за допомогою процесу, що імітує роботу людського мозку. У цьому сенсі нейронні мережі належать до систем нейронів органічного чи штучного походження.

Нейронні мережі досягли неймовірних висот у широкому спектрі задач, наприклад, порівняння та розпізнавання ідентичних даних, що використовується в системах безпеки інфраструктурних об'єктів. Виконується це шляхом фіксації обличчя людей, та порівняння їх із базою аналогів. Ще один приклад – функція Google по пошуку подібного зображення. Достатньо завантажити фото і система знайде усі схожі зображення.

Серед глибоких нейронних мереж (DNN) згорткова нейронна мережа (CNN) продемонструвала відмінні результати у завданнях комп'ютерного зору, особливо у класифікації зображень. Згорткова нейронна мережа (CNN, або ConvNet) – це особливий тип багатошарової нейронної мережі, натхненний механізмом оптичних та нейронних систем людини[1].

CNN – це структура, розроблена з використанням концепцій машинного навчання. CNN можуть самостійно навчатися та тренуватися на основі даних без втручання людини.

Нами розв'язане завдання класифікації біооб'єктів (різних ракурсів обличчя, тканин людського організму, пальців руки тощо) на зображеннях. Для програмної реалізації обрано мову Python через наявність спеціалізованих бібліотек опрацювання зображень Tensorflow, Keras та OpenCV.

У 2020 році велика глибока згорткова нейронна мережа під назвою AlexNet показала відмінну продуктивність на конкурсі ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2], що започаткувало широке використання та розвиток моделей згорткових нейронних мереж (CNN), таких як VGGNet, Gool і багато інших.

Для виконання завдань класифікації, я скористаюсь моделлю для класифікації зображень VGGNet[3, 4]. В результаті для заданого зображення наша модель виокремлює знайдені біооб'кти.

Перелік використаних джерел

1. A Complete Guide to Image Classification in 2021. URL: https://viso.ai/computer-vision/image-classification/
2. ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). URL: https://image-net.org/challenges/LSVRC/2012/index.php
3. VGGNet-16 Architecture: https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide
4. *Petryk M., Gancarczyk T., Khimich O*. Methods of Mathematical Modeling and Identification of Complex Processes and Systems on the basis of High-performance Calculations (neuro- and nanoporous feedback cyber systems, models with sparse structure data, parallel computations). Scientific Publishing University of Bielsko-Biala. Bielsko-Biala, Poland, 2021, 194 p.