С. Д. ШТОВБА
О. М. КОЗАЧКО

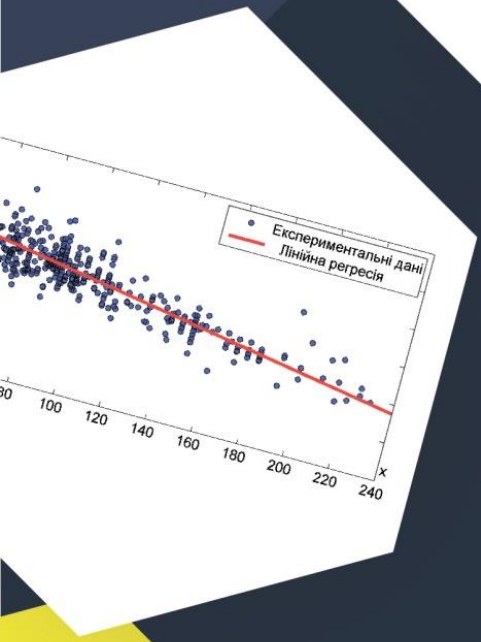
MACHINE LEARNING

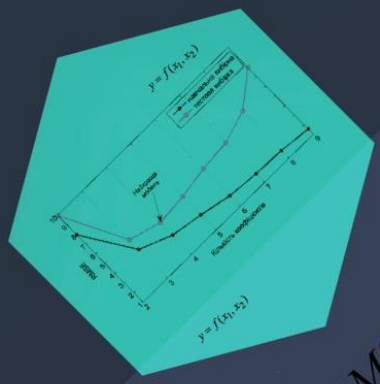
$$y = f(x_1, x_2)$$

$$w_j \in [0, 1]$$

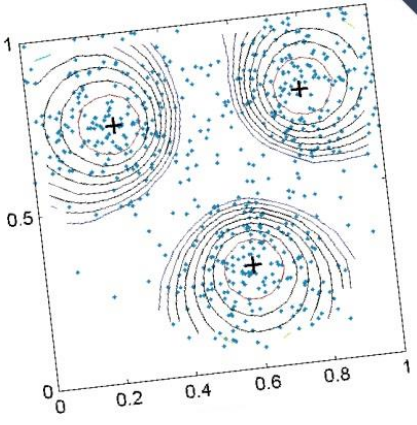
$$SWSE = \sum_{j=1,M} w_j \cdot (y_j - f(X_j))^2$$

$$w_j \in [0, 1]$$





$$RMSE_F = \sqrt{\frac{1}{M-N} \sum_{j=1,M} (y_j - f(X_j, P))^2}$$



СТАРТОВИЙ КУРС

Міністерство освіти і науки України
Вінницький національний технічний університет

С. Д. Штовба
О. М. Козачко

Machine Learning: стартовий курс

електронний навчальний посібник

Вінниця
ВНТУ
2020

УДК 681.3.06(075)
ББК 22.18я73
Ш92

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол №6 від 30.01.2020 р.).

Рецензенти:
О. В. Бісікало, доктор технічних наук, професор;
П. І. Кулаков, доктор технічних наук, професор;
А. В. Баєв, кандидат фізико-математичних наук.

Штовба С.Д.
Ш92 Machine learning: стартовий курс : електронний навчальний посібник / Штовба С.Д., Козачко О.М. – Вінниця : ВНТУ, 2020. – 81 с.

У навчальному посібнику наведено базовий теоретичний матеріал та практичні приклади з регресійного аналізу, класифікації, категоризації та кластеризації. Навчальний посібник призначено для студентів спеціальностей 151 – “Автоматизація та комп'ютерно-інтегровані технології”, 124 – “Системний аналіз”, 125 – “Кібербезпека” та 126 – “Інформаційні технології”, що вивчають дисципліни з аналізу даних та машинного навчання.

УДК [659.1+339.138] 681.3.06 (075)

ББК 22.18

© С. Д. Штовба, 2020

© О.М. Козачко, 2020

ЗМІСТ

Передмова

Machine Learning або машинне навчання – це сукупність методів та технологій обробки експериментальних даних з метою отримання з них різноманітних математичних моделей. Ключовою особливістю Machine Learning є те, що модель навчається на даних, так само, як дитина навчається на прикладах. Термін Machine Learning увів Артур Самуель (Arthur Samuel) 60 років тому. У наукових колах термін набув популярності на початку XXI сторіччя, а в останню п'ятирічку широко вживається в IT-галузі та за її межами. Сплеск інтересу до машинного навчання обумовлено тим, що в техніці, економіці, медицині, біології, політиці, спорті та в інших областях накопичені величезні масиви спостережень за різноманітними багатофакторними залежностями. І з цими даними потрібно щось робити – обробити їх у деякий спосіб, щоб отримати коректні моделі досліджуваних залежностей з метою пояснення процесів, управління процесами, прогнозування, прийняття рішень тощо.

Метою посібника є викладення теоретичних основ машинного навчання, зокрема регресійного аналізу, класифікації, категоризації та кластеризації. Ми не фокусуємося на строгості математичних викладок, натомість наводимо багато прикладів, значна частка яких пов'язана з реальними задачами. Така насиченість прикладами обумовлена тим, що багато студентів IT-галузі переважно навчається лише на прикладах.

Книга займає проміжну позицію між виданнями з аналізу даних в стилі *"математики для математиків про математику"* та інструктивним виданнями із застосуванням програмних середовищ в стилі *"Machine Learning без сліз з пакетом XXX"*. В пропонованій книзі дозовано викладається теоретичний матеріал, а також показується як вирішувати деякі практичні завдання машинного навчання за допомогою сучасного інструментарію – програмного середовища MATLAB. Дані з прикладів доступні для завантаження з репозиторію з машинного навчання Каліфорнійського університету в Ірвіні та kaggle-конкурсів. Це дозволить

студентам не лише відтворити приклади з посібника, але провести і власні обчислювальні експерименти та порівняти отримані моделі з результатами інших дослідників.

Передбачається, що студенти мають елементарні знання з теорії ймовірностей, матричних обчислень та методів оптимізації, а також володіють ключовими навиками роботи в програмному середовищі MATLAB. Для новачків рекомендуємо компактний посібник [6]. Викладення окремих методів передбачає, що студенти володіють базовими знаннями з теорії нечітких множин, які можна почерпнути, наприклад, з [10]. Кожен розділ посібника закінчує дюжина контрольних питань. Частина питань в посібнику не висвітлена – їх мета спрямувати студентів на зростання, на підвищення кваліфікації. При підготовці посібника використано таку літературу: для першого розділу – [1, 5, 7–9]; для другого розділу – [1–4, 7–9, 12–16]; для третього розділу – [4, 7, 10]. Також використовувалися матеріали з документації по системі MATLAB, зокрема з Statistics and Machine Learning Toolbox та Fuzzy Logic Toolbox. Зазначену літературу рекомендуємо для поглибленого вивчення матеріалу. Уся література доступна в інтернеті.

1. РЕГРЕСІЯ

1.1. Ази

Розглядається залежність з n входами $X=(x_1, x_2, ..., x_n)$ та одним виходом y. Вважається відомою вибірка експериментальних даних про цю залежність:

$$(X_j, y_j), \quad j=\overline{1, M}, \tag{1}$$

де M – довжина вибірки;

$X_j=(x_{j1}, x_{j2}, ..., x_{jn})$.

У вибірці (1) вхідні та вихідна змінні приймають числові значення. Фрагмент такої вибірки для задачі прогнозування добової кількості коментарів повідомлення в блозі (y) наведено в табл. 1. Використовуються такі позначення вхідних змінних:

x_1 – кількість отриманих коментарів на поточний момент;

x_2 – кількість коментарів за останні 24 години;

x_3 – вік публікації;

x_4 – довжина тексту публікації.

Коли накопичується достатньо багато експериментальних даних, виникає бажання їх узагальнити. Наприклад, віднайти теоретичну залежність $y=f(X)$, яка найкращим чином їх опише. Існують 2 підходи до узагальнення даних: інтерполяція та апроксимація. Інтерполяція – це знаходження моделі, яка точно «проходить» по експериментальним даним. Інтерполяцію застосовують, коли експериментальні дані достовірні, без шумів. Для багатьох практичних задач експериментальні дані є неточними, зашумленими, тому підібрати модель яка один в один відповідатиме даним неможливо (і непотрібно). Апроксимація – це знаходження моделі, яка згладжує експериментальні дані, здійснює фільтрацію шумів, виділяючи тренд. Саме про апроксимацією будемо вести мову далі.

Таблиця 1 – Фрагмент вибірки даних для задачі Blog Feedback з репозиторію [23]

x_1	x_2	x_3	x_4	y
65	0	65	3568	0
88	2	42	6682	2
92	92	17	3133	2
90	2	65	6682	0
94	2	40	3133	0
94	0	65	3133	0
120	120	23	763	5
4	4	0	1568	18
125	5	47	763	1
22	18	24	1568	1
458	458	23	7089	6
126	1	71	763	0
23	1	48	1568	0
464	6	47	7089	17
199	199	15	2155	25

Наскільки добре модель відповідає даним, наскільки модель точна, оцінюють за допомогою деякого критерію. Чим менше значення критерію, тим ближче результати моделювання до експериментальних даних. Спосіб розрахунку цього критерію називають метрикою або функцією помилки. Останнім часом метрикою називають і сам критерій, тобто критерій і метрика вважаються синонімами. Зазвичай критерієм точності призначають суму квадратів відхилень теорії від експерименту – *SSE* (Sum of Squared Error). Таку метрику вперше застосував Карл Гаус в 1801 р. для розрахунку орбіти Церери – найменшої карликової планети (рис. 1). Цереру відкрив Джузепе Піаці 1 січня 1801 року і впродовж 42 днів детектував її 24 рази. Після 11 лютого усі спроби виявити планету були марними. Карл Гаус апроксимував зашумлені результати спостережень і розрахував місце появи планети. Цереру відразу виявили в вказаному місці 31 грудня 1801 р. Двадцятичотирьохрічний Карл Гаус не опублікував свій метод розрахунку, вважаючи його очевидним. Незалежно від нього аналогічний метод розробив Андрієн-Марі Лежандр, і опублікував його в 1805 р. під назвою «Méthode des moindres quarrés» (метод найменших квадратів). Карл Гаус опублікував свої результати пізніше – в 1809 р., що спричинило найбільшу в історії статистики суперечку за пріоритет

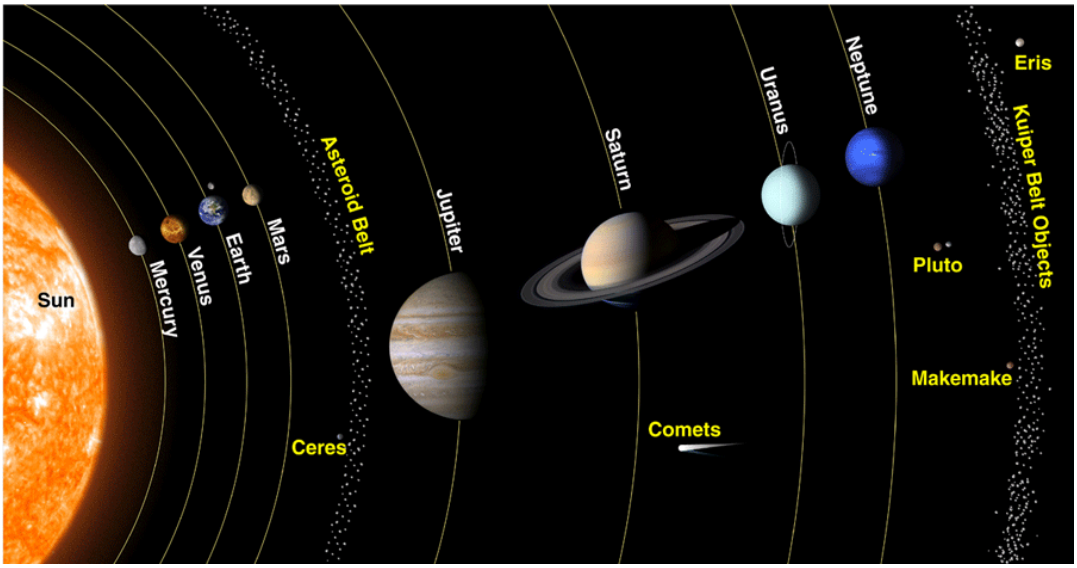


Рисунок 1 – Планета Церера (Ceres) – перший об'єкт регресійного аналізу (рисунок NASA)

Суму квадратів відхилень для регресійної моделі $f(X)$ на вибірці (1) розраховують таким чином:

$$SSE = \sum_{j=1,\,M} \left(y_j - f(X_j)\right)^2 . \tag{2}$$

Для зручності критерій *SSE* нормують – усереднюють на довжину вибірки та приводять до розмірності вихідної змінної *y*. Таку модифікацію критерію *SSE* називають середньою квадратичною нев'язкою – *RMSE* (Root Mean Squared Error). Вона розраховується таким чином:

$$RMSE = \sqrt{\frac{1}{M} \sum_{j=1,\,M} \left(y_j - f(X_j)\right)^2} . \tag{3}$$

В модель $y = f(X)$ входять параметри (*P*), значення яких підбирають таким чином, щоб мінімізувати нев'язку (3): $RMSE(P) \to \min$. Процедура знаходження параметрів моделі називається регресійним аналізом. Відповідно, залежність $y = f(X, P)$ називається регресією.

Регресійний аналіз називається лінійним, якщо залежність $y = f(X, P)$ є лінійною:

$$y = a_0 + \sum_{i=1,\,n} a_i x_i \,, \tag{4}$$

де $a_0, a_1, ..., a_n$ – регресійні коефіцієнти, відповідно, $P = (a_0, a_1, ..., a_n)$.

Приклад лінійного регресійного аналізу залежності тривалості розгону автомобіля до швидкості 60 миль за год. (y) від потужності двигуна (x) наведено на рис. 2. Регресійний аналіз здійснено за даними задачі Auto MPG з репозиторію [23]. Ця задача містить дані за 392 різними автомобілями. Отримана регресійна модель $y = 20.7019 - 0.0494x_1$ апроксимує експериментальні дані з нев'язкою *RMSE*=1.9965.

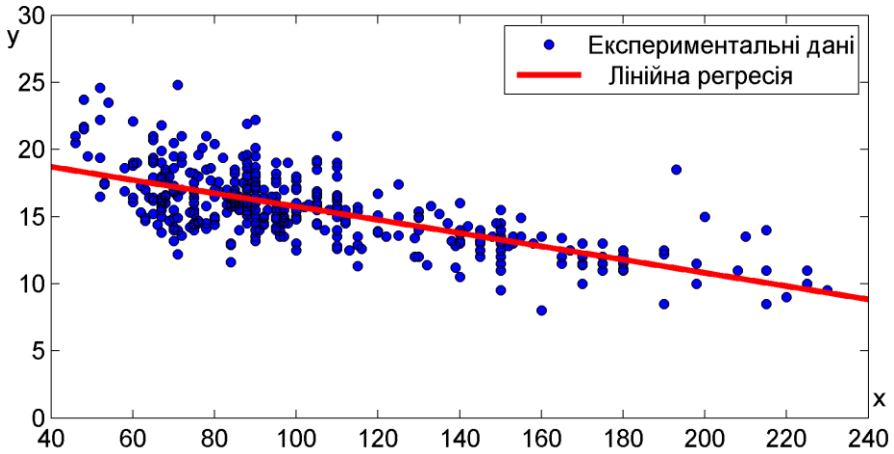


Рисунок 2 – Приклад лінійного регресійного аналізу

Підбір параметрів регресійної моделі зводиться до вирішення задачі оптимізації цільової функції (2) відносно вектора керованих змінних *P*. Цільова функція є диференційованою, тому необхідною умовою мінімуму є нульові частини похідні. Відповідно, для аналітичного знаходження оптимуму необхідно розв'язати таку систему рівнянь:

$$\begin{cases} \dfrac{\partial RMSE}{\partial a_0} = 0 \\ \dfrac{\partial RMSE}{\partial a_1} = 0 \\ \vdots \\ \dfrac{\partial RMSE}{\partial a_n} = 0 \end{cases}.$$

Для найпростішого випадку – для одновимірного лінійного регресійного аналізу коефіцієнти a_0 та a_1 моделі $y = a_0 + a_1 x$ розраховують так:

$$a_1 = \frac{\overline{xy} - \bar{x} \cdot \bar{y}}{\overline{x^2} - \bar{x} \cdot \bar{x}},$$

$$a_0 = \bar{y} - a_1 \bar{x},$$

де $\bar{x} = \frac{1}{M} \sum_{j=1,\,M} x_j$;

$\bar{y} = \frac{1}{M} \sum_{j=1,\,M} y_j$;

$\overline{x^2} = \frac{1}{M} \sum_{j=1,\,M} x_j^2$;

$\overline{xy} = \frac{1}{M} \sum_{j=1,\,M} x_j \cdot y_j$.

Аналітично можна знайти коефіцієнти деяких інших регресійних залежностей, але відповідні формули будуть громіздкими. Компактний запис можна отримати за матричного представлення регресійного рівняння. Розглянемо регресійне рівняння $y = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$. Нехай поточні значення вхідних змінних є такими: $\left(x_1^*, x_2^*, \ldots, x_n^*\right)$. Тоді розрахунок вихідного значення y^* можна реалізувати у такій матричній формі:

$$y^* = \left(1, x_1^*, x_2^*, \ldots, x_n^*\right) \times \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}. \qquad (5)$$

Підставляючи в (5) навчальну вибірку (1), отримуємо:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & & & & \\ 1 & x_{M1} & x_{M2} & \dots & x_{Mn} \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}. \qquad (6)$$

Розв'язуючи матричне рівняння (6), знаходимо невідомі регресійні коефіцієнти:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & & & & \\ 1 & x_{M1} & x_{M2} & \dots & x_{M2} \end{pmatrix} \backslash \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix}, \qquad (7)$$

де символ \ позначає ліве ділення.

Формулу (7) в матричній формі можна записати так:

$$\mathbf{A} = \mathbf{X} \backslash \mathbf{Y}, \qquad (8)$$

де **A** – вектор-стовпчик коефіцієнтів регресійної моделі;

X – матриця вхідних даних спостережень з навчальної вибірки (1) з додатковим стовпчиком одиниць;

Y – вектор-стовпчик вихідної змінної з навчальної вибірки (1).

По аналогії з (7) можна в матричній формі описати процедуру знаходження коефіцієнтів для багатьох популярних нелінійних регресійних моделей. Наприклад, для моделі $y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1^2 + a_4 x_1 x_2$, коефіцієнти знаходяться таким чином:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{11}^2 & x_{11}x_{12} \\ 1 & x_{21} & x_{22} & x_{21}^2 & x_{21}x_{22} \\ \vdots & & & & \\ 1 & x_{M1} & x_{M2} & x_{M1}^2 & x_{M1}x_{M2} \end{pmatrix} \backslash \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix}.$$

Можна вважати, що 2 останні стовпчики в цій формулі описують 2 фіктивні змінні $x_3 = x_1^2$ та $x_4 = x_1 x_2$. Нелінійна регресійна модель $y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1^2 + a_4 x_1 x_2$ відносно змінних x_1 та x_2, стає лінійною відносно змінних x_1, x_2, x_3, x_4: $y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4$. Тому, і коефіцієнти моделі $a_0, a_1, ..., a_4$ розраховуються як для лінійного випадку.

Проведемо за формулою (7) лінійний регресійний аналіз за даними з рис. 2 про залежність тривалості розгону автомобіля (y) від потужності двигуна (x). Для цього в середовищі MATLAB напишемо таку програму:

```
%-------------------------------------------------------------
%Однофакторний лінійний регресійний аналіз для задачі
%Auto MPG.
%-------------------------------------------------------------
%Завантаження набору даних з демо-прикладу Fuzzy Logic
Toolbox:
[data, input_name] = loadgas;
%Примітка: дані з файлу можна завантажити командами load,
%importdata або через кнопку Import Data в меню Home.
input_name
%Відбір вхідної та вихідної змінних:
x = data(:, 3);
y = data(:, 5);
%Довжина навчальної вибірки:
M=length(x);
disp('Лінійний регресійний аналіз:')
X=[ones(M,1) x];
A=X\y
%Розрахунок нев'язки:
prediction_y=X*A;
RMSE=norm(y- prediction_y)/sqrt(M)
```

В результаті отримаємо:

```
input_name =
  7×8 char array

    'Cylinder'
    'Disp    '
    'Power   '
    'Weight  '
```

```
    'Acceler '
    'Year    '
    'MPG     '

Лінійний регресійний аналіз:

A =
   20.7019
   -0.0494

RMSE =
    1.9965
```

Вищенаведені результати відповідають регресійній моделі $y = 20.7019 - 0.0494x_1$ з нев'язкою $RMSE$=1.9965.

В середовищі MATLAB однофакторний регресійний аналіз зручно робити в графічному вікні plot. Для цього візуалізуємо експериментальні дані такими командами:

```
plot(x, y, 'ro')
xlabel('Потужність');
ylabel('Тривалість розгону');
```

Тепер в графічному вікні, що з'явилося, оберемо в меню Tools пункт Basic Fitting. Це відкриє нове графічне вікно для апроксимації даних. Оберемо лінійну залежність (linear) з виведенням на екран формули (Show equation) та нев'язки (Show norm of residuals). В результаті отримуємо коефіцієнти лінійної регресії та значення нев'язки (рис. 3). Графік регресії наведено на рис. 4. Коефіцієнти тотожні тим, що розраховані раніше. А от нев'язки різні: 1.9965 та 39.528. Це і не дивно, тому що в першому випадку середня квадартична нев'язка, а в другому - сумарна квадартична нев'язка. Поділивши на $\sqrt{M}$ отримаємо тотожні значення: $\frac{39.528}{\sqrt{392}} = 1.9965.$

Для підбору коефіцієнтів квадратичної регресії необхідно у графічному вікні з рис. 3 в полі Check to display fits on figure обрати пункт quadratic. В результаті отримаємо квадратичну регресійну модель, яка зображена на рис. 5.

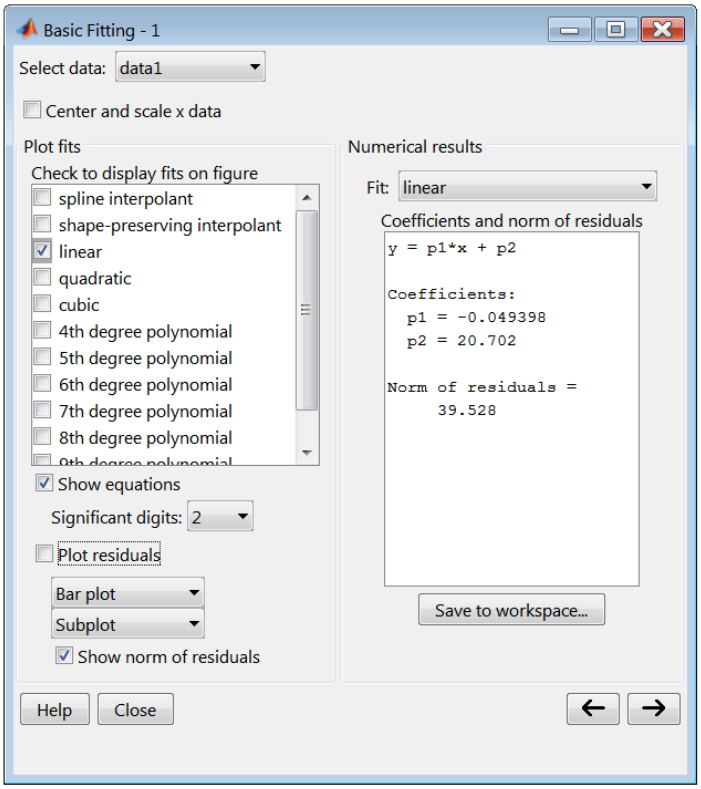


Рисунок 3 – Приклад лінійного регресійного аналізу

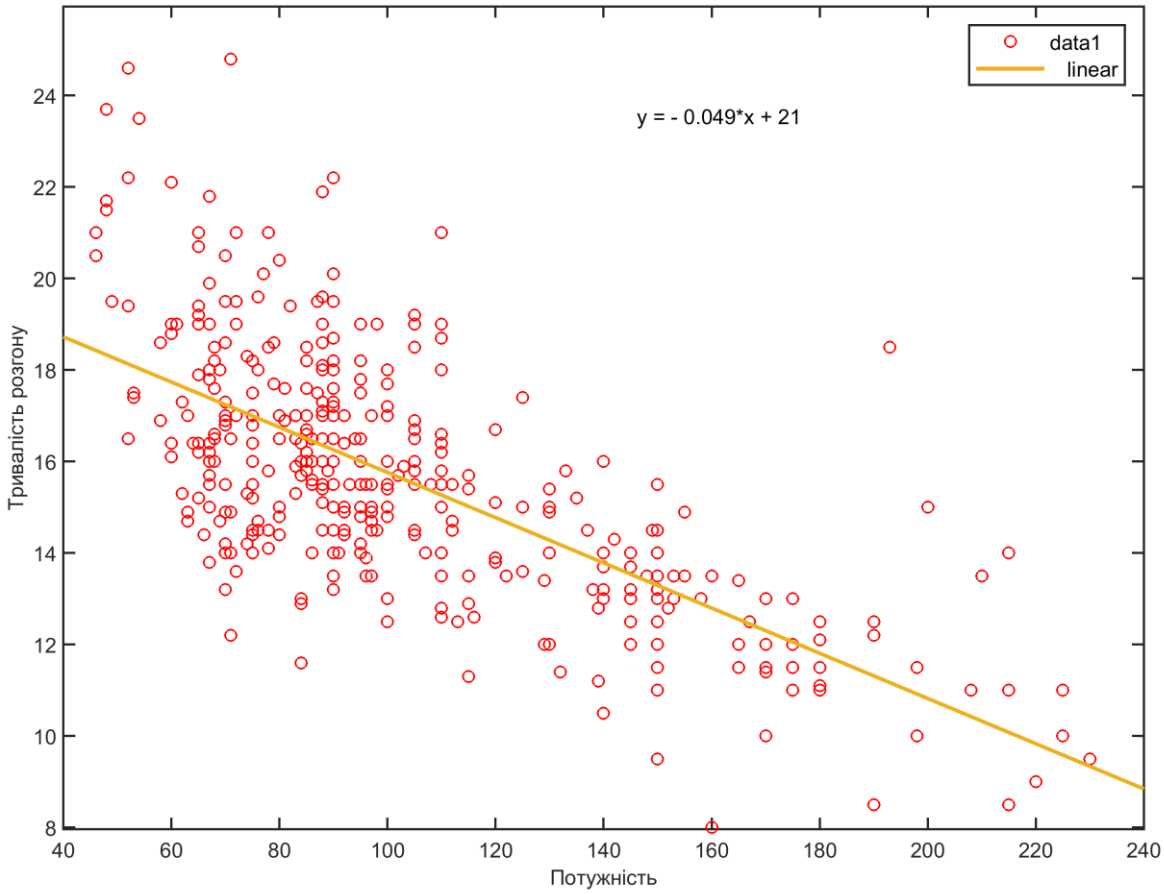


Рисунок 4 – Графік лінійної регресії у вікні plot

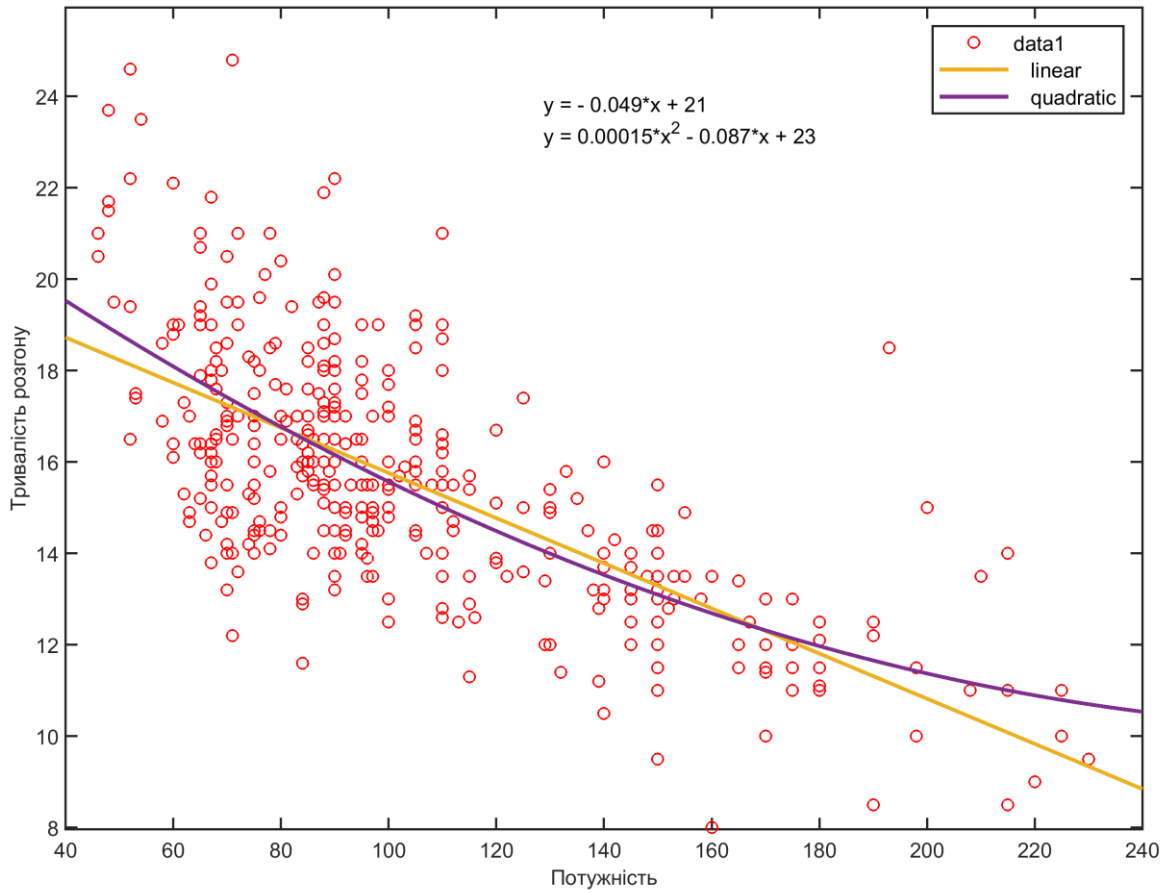


Рисунок 5 – Графіки лінійної та квадратичної регресій у вікні plot

Нев’язка квадратичної моделі трохи менша ніж у лінійної: 39.159 проти 39.528. Логічно спробувати для зменшення нев’язки збільшити порядок регресійної моделі. Результат регресійного аналізу для полінома дев’ятого степеню зображено на рис. 6. Нев’язка цієї моделі вийшла 37.847, що краще ніж для лінійної та квадратичної регресії. Але поведінка моделі дев’ятого степеня суперечить здоровому глузду – для автомобілів великої потужності тривалість розгону стрімко зростає зі збільшенням потужності. Крім того, функція вже не монотонно-спадна. Таким чином, модель вийшла заскладною, і замість того, щоб виявити тенденції в даних, вона апроксимувала шуми.

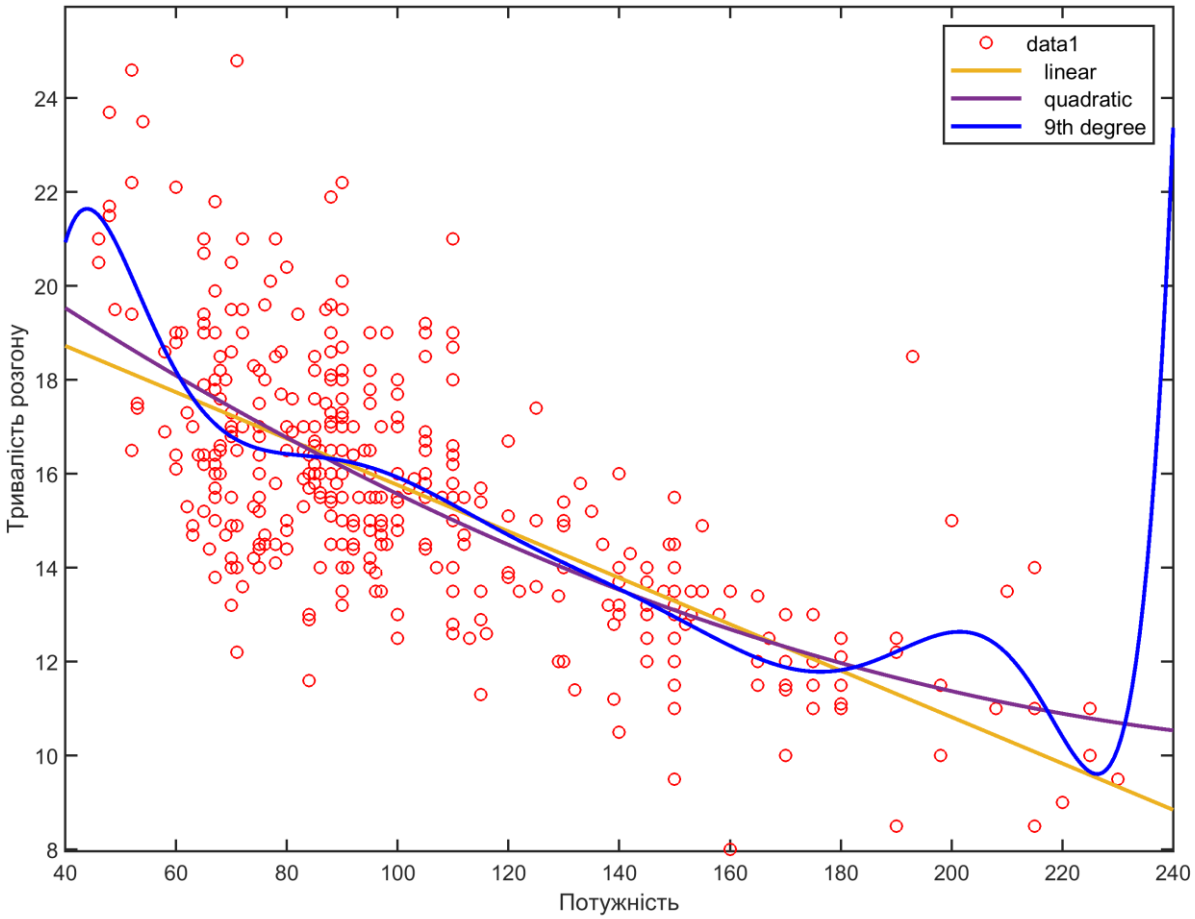


Рисунок 6 – Графік трьох регресійних залежностей у вікні plot

За складність модель можна «покарати», додавши деякий штраф до *RMSE*. Чим складніше модель, тим *RMSE* має бути більшим за одних і тих самих результатів апроксимації. Один із варіантів такого штрафування – так звана незміщена середня квадратична нев'язка:

$$RMSE_F = \sqrt{\frac{1}{M-N} \sum_{j=\overline{1,M}} \left(y_j - f(X_j, P)\right)^2}, \qquad (9)$$

де $P = (p_1, p_2, \ldots, p_N)$ - вектор коефіцієнтів регресійної моделі;

$M - N$ - число степенів свободи.

Нев'язки (3) та (9) сильно відрізняються, коли кількість параметрів регресійної моделі наближається до обсягу вибірки. Якщо даних багато, то відмінність між цими критеріями буде незначною, і за різницею між ними не визначити вийшла регресійна модель заскладною чи ні.

В командному режимі однофакторний регресійний аналіз на основі поліномів зручно проводити за допомогою функції polyfit. Функція викликається в такому форматі:

```
p=polyfit(x, y, n),
```

де **x** – вектор-стовпчик вхідних значень;

y – вектор-стовпчик вихідних значень;

n – порядок регресійного полінома;

p – коефіцієнти полінома.

1.2. Навчальна і тестова вибірки

Регресійні коефіцієнти визначають за усіма експериментальними даними тільки у тому випадку, коли наперед достовірно відома структура моделі. Іншими словами, коли з деяких причин ясно, що шукана залежність є лінійною, чи квадратичною, чи логарифмічною тощо. Якщо структура залежності наперед невідома, тоді слід діяти за принципом зовнішнього доповнення. Цей принцип найпростіше реалізувати, розділивши експериментальні дані на дві репрезентативні вибірки – навчальну та тестову. Далі згенеруємо множину моделей з різною структурою – лінійні, квадратичні, кубічні тощо. За навчальною вибіркою для кожної моделі-кандидату визначимо оптимальні значення коефіцієнтів. Потім кожну модель перевіримо на тестовій вибірці. Найкращою оберемо модель, для якої критерій (3) на тестовій вибірці є мінімальним.

Описаний підхід запобігає переускладненню моделі. На перший погляд здається, що чим складніша модель, тим вона точніша. Тобто, кубічна модель буде точніша за квадратичну, а квадратична – за лінійну. Якщо побудувати залежність нев'язки (3) на навчальній вибірці від складності моделі, наприклад, від загальної кількості регресійних коефіцієнтів, то вона буде монотонно-спадною (рис. 7). Коли кількість регресійних коефіцієнтів моделі дорівнюватиме M нев'язка на навчальній вибірці буде мінімальною. Але якщо перевірити отримані моделі на тестовій вибірці, то зазвичай виявляється, що зі збільшенням складності моделі нев'язка спочатку зменшується, а потім починає зростати. На рис. 7

оптимальною є регресійна модель з чотирма коефіцієнтами. Подальше ускладнення моделі призводить до втрати генералізуючих властивостей.

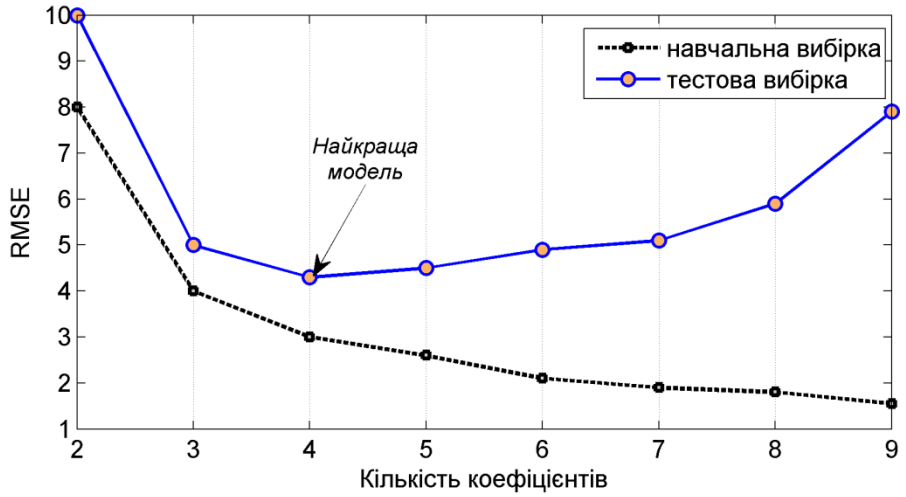


Рисунок 7 – Типові залежності нев'язки від складності моделі

Розглянемо як виконати регресійний аналіз в середовищі MATLAB з використанням навчальної та тестової вибірок. Для прикладу побудуємо двофакторну регресійну модель $y = f(x_1, x_2)$ про залежність паливної ефективності легкового автомобіля (y) від його маси (x_1) та року випуску (x_2). Це спрощений варіант задачі Auto MPG з UCI Machine Learning Repository. Паливна ефективність визначається кількістю миль, які можна проїхати на одному галоні палива (MPG – miles per gallon).

Експериментальні дані розіб'ємо на навчальну та тестову вибірки. В навчальну включимо непарні рядки даних, а в тестову – парні. Для перевірки репрезентативності вибірок розрахуємо математичне сподівання (mean) та середнє квадратичне відхилення (std) даних за кожною змінною. Результати перевірки (табл. 2) вказують на приблизно однакові значення цих показників для навчальної та тестової вибірок, тому вважаємо їх репрезентативними. Зображенням типу «сіль з перцем» на рис. 8 візуально підтверджується цей висновок.

Однофакторні залежності (рис. 9) вказують на сильну тенденцію зменшення паливної ефективності зі збільшенням маси автомобіля та слабку тенденцію зростання її для більш нових моделей. Відповідно, в лінійній моделі $y = a_0 + a_1 x_1 + a_2 x_2$ коефіцієнт a_1 має бути від'ємним, а a_2 – додатнім.

Таблиця 2 – Статистики даних

Вибірка	mean(x_1)	mean(x_2)	mean(y)	std(x_1)	std(x_2)	std(y)
Навчальна	2989.6	75.97	23.42	828.2	3.67	7.88
Тестова	2965.6	75.98	23.47	872	3.69	7.75

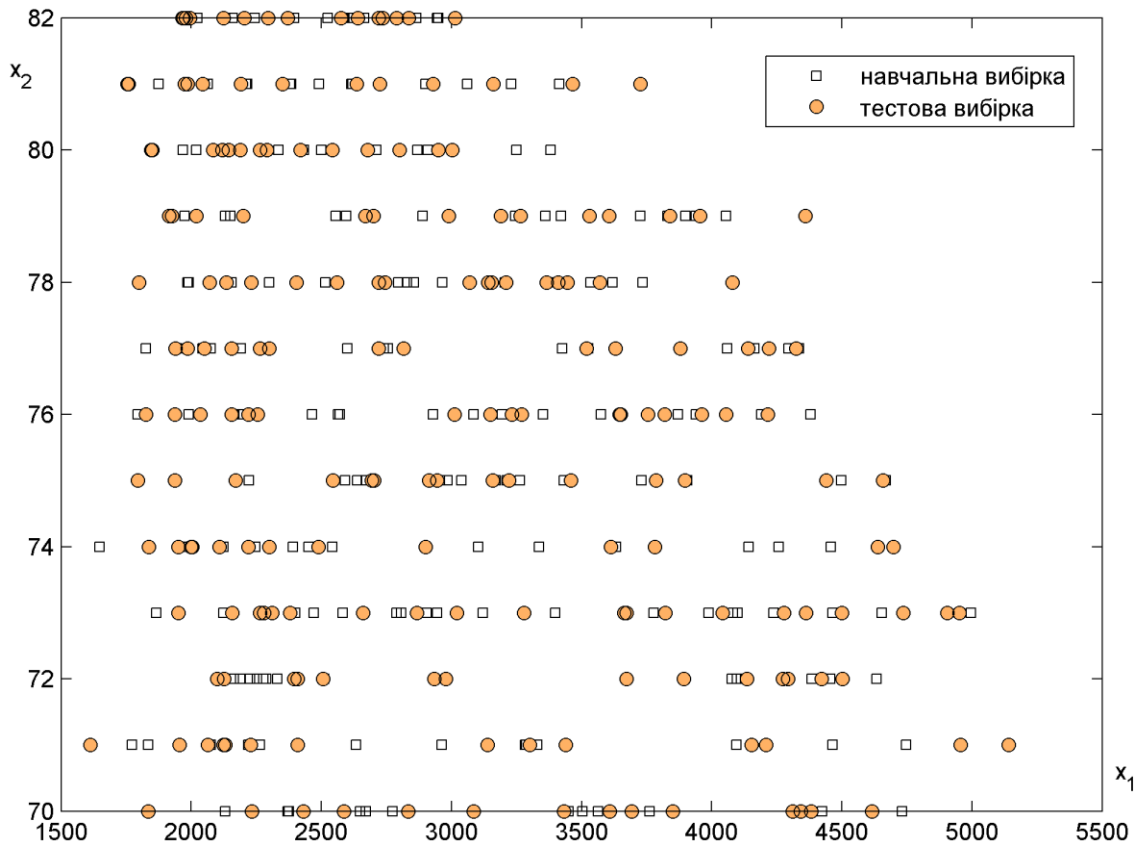


Рисунок 8 – Розподіл даних в навчальній та тестовій вибірках

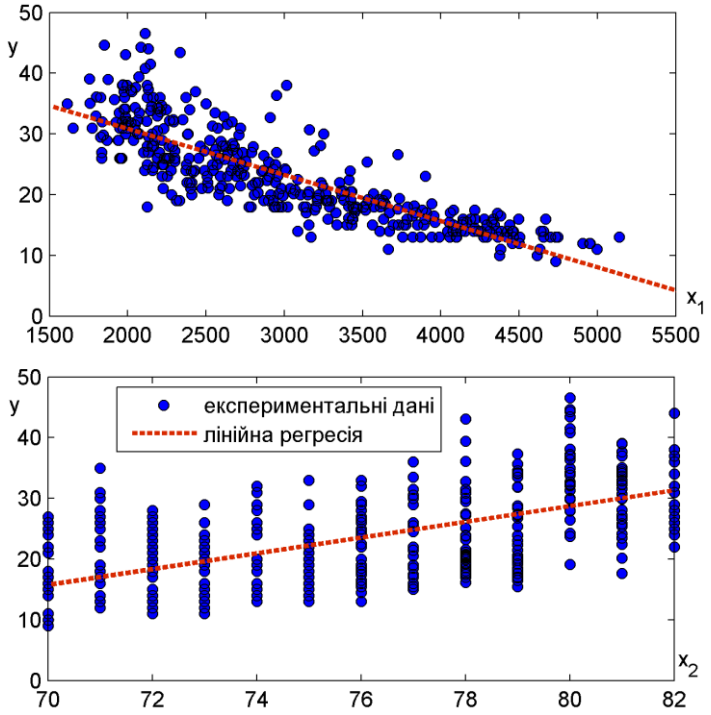


Рисунок 9 – Однофакторні залежності

Знайдемо коефіцієнти лінійної регресії за допомогою такої програми:

```
%-------------------------------------------------------------
%Двофакторний лінійний регресійний аналіз для задачі Auto MPG.
%-------------------------------------------------------------
%Зчитування даних:
[data, input_name] = loadgas;
vars=[4 6 7];
%Формування навчальної та тестової вибірок:
M=length(data);
tr_index=1:2:M;
test_index=2:2:M;
x1_tr= data(tr_index, vars(1));
x2_tr= data(tr_index, vars(2));
y_tr = data(tr_index, vars(3));
x1_test= data(test_index, vars(1));
x2_test= data(test_index, vars(2));
y_test = data(test_index, vars(3));
%Довжина навчальної вибірки:
M_tr=length(y_tr);
disp('Лінійний регресійний аналіз:')
X_tr=[ones(M_tr, 1) x1_tr x2_tr];
A=X_tr\y_tr
%Розрахунок нев'язки на навчальній вибірці:
```

```
pred_tr=X_tr*A;
RMSE_tr=norm(y_tr-pred_tr)/sqrt(M_tr)
%Розрахунок нев'язки на тестовій вибірці:
M_test=length(y_test);
X_test=[ones(M_test,1) x1_test x2_test];
pred_test=X_test*A;
RMSE_test=norm(y_test-pred_test)/sqrt(M_test)
%Побудова графіків:
subplot(2,2,1)
plot(y_tr, pred_tr, 'ko', 'markersize', 5)
hold on
plot([5 50],[5 50], 'r-', 'linewidth', 1)
xlabel('Експеримент')
ylabel('Теорія')
RMSE_str=num2str(RMSE_tr, 3);
title(['Лінійна регресія, RMSE_t_r=', RMSE_str])
subplot(2,2,2)
plot(y_test, pred_test, 'ko', 'markersize', 5)
hold on
plot([5 50],[5 50], 'r-', 'linewidth', 1)
xlabel('Експеримент')
ylabel('Теорія')
RMSE_str=num2str(RMSE_test, 3);
title(['Лінійна регресія, RMSE_t_e_s_t=', RMSE_str])
```

В результаті отримаємо регресійну модель $y = -15.37 - 0.007x_1 + 0.779x_2$ з такими нев'язками $RMSE_{tr}$=3.5 та $RMSE_{test}$=3.34. Як і очікувалось коефіцієнт при x_1 вийшов від'ємним, а при x_2 – додатнім. Для знаходження квадратичної регресійної моделі в програму додамо такі рядки:

```
disp('Квадратична регресія:')
X_tr=[ones(M_tr, 1) x1_tr x2_tr x1_tr.^2 x2_tr.^2
x1_tr.*x2_tr];
A=X_tr\y_tr
%Розрахунок нев'язки на навчальній вибірці:
pred_tr=X_tr*A;
RMSE_tr=norm(y_tr-pred_tr)/sqrt(M_tr)
%Розрахунок нев'язки на тестовій вибірці:
X_test=[ones(M_test,1) x1_test x2_test x1_test.^2 ...
       x2_test.^2 x1_test.*x2_test];
```

```
pred_test=X_test*A;
RMSE_test=norm(y_test-pred_test)/sqrt(M_test)
%Побудова графіків:
subplot(2,2,3)
plot(y_tr, pred_tr, 'ko', 'markersize', 5)
hold on
plot([5 50],[5 50], 'r-', 'linewidth', 1)
xlabel('Експеримент')
ylabel('Теорія')
RMSE_str=num2str(RMSE_tr, 3);
title(['Квадратична регресія, RMSE_t_r=', RMSE_str])
subplot(2,2,4)
plot(y_test, pred_test, 'ko', 'markersize', 5)
hold on
plot([5 50],[5 50], 'r-', 'linewidth', 1)
xlabel('Експеримент')
ylabel('Теорія')
RMSE_str=num2str(RMSE_test, 3);
title(['Квадратична регресія, RMSE_t_e_s_t=', RMSE_str])
```

В результаті виконання програми отримаємо регресійну модель $y = 166.7 - 0.001x_1 - 4.258x_2 + 0x_1^2 + 0.0382x_2^2 - 0.0003x_1x_2$ із значно меншими нев'язками: $RMSE_{tr}$=2.95 та $RMSE_{test}$=2.91. Порівнюючи теоретичні результати з експериментальними даними (рис. 10) бачимо, що для лінійної моделі характерні систематичні помилки - заниження прогнозованого показника для автомобілів з дуже високою та з дуже низькою паливною ефективністю. Для квадратичної моделі такі ефекти не спостерігаються – залишки майже рівномірно розподілені поміж автомобілів з різною паливною ефективністю.

Поверхня квадратичної регресійної моделі а також відхилення теоретичних результатів від експериментальних даних наведено на рис. 11. З нього видно, що модель не суперечить здоровому глузду – вона не містить горбів чи провалів, які неможливо змістовно інтерпретувати. Для побудови рис. 11 застосовано функції **surf**, **plot3** та **scatter3**.

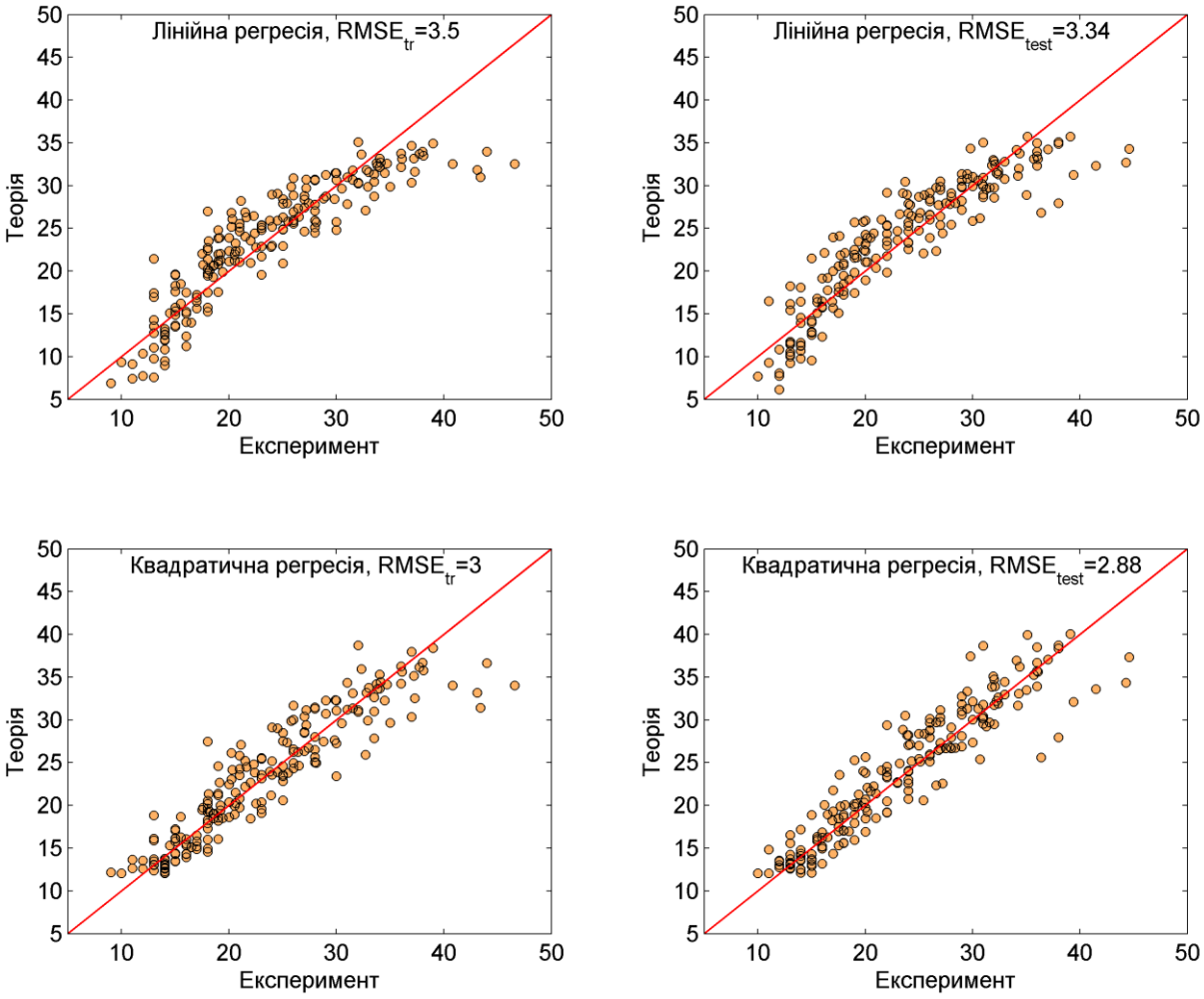


Рисунок 10 – Порівняння експериментальних значень паливної ефективності з результатами моделювання

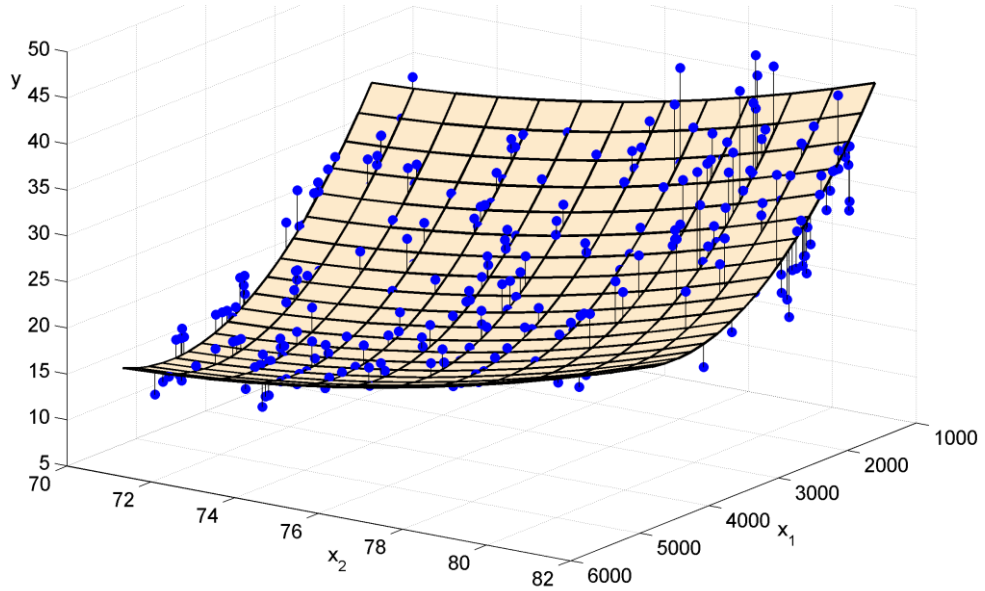


Рисунок 11 – Поверхня “входи – вихід” найкращої регресійної моделі та її відповідність експериментальним даним

Порівнюючи регресійну залежність з експериментальними даними бачимо, що між ними є відхилення. Часто буває, що графік регресійної залежності не проходить абсолютно точно по жодному спостереженню. Виникає задача виявлення коридору, в межах якого гарантовано буде знаходитися більша частка експериментальних даних. Відповідно, коефіцієнти в регресійній моделі мають задаватися не точково, а інтервалом. Такі інтервали називають довірчими. Ширина довірчого інтервалу залежить від довірчої ймовірності – ймовірності того, що результати спостереження не вийдуть за межі цього гарантованого коридору. Наприклад, якщо довірча ймовірність дорівнює 0.95, то 95% спостережень не вийдуть за межі гарантованого коридору. Для розрахунку довірчих інтервалів можна використати функцію regress таким чином:

```
[A, A_int]=regress(y, X, Alpha),
```

```
[A, A_int, r, rint, stats]=regress(y, X, Alpha),
```

де **A** – регресійні коефіцієнти;

A_int – довірчі інтервали коефіцієнтів;

r – вектор залишків (**y-X*A**);

rint – довірчий інтервал залишків **r**;

stats – вектор, що містить такі статистики: R^2 – коефіцієнт детермінації; f-критерій Фішера; рівень значущості регресійної моделі та оцінка дисперсії помилок;

y – вектор вихідних значень розміром $m \times 1$, де m – довжина вибірки даних;

X – матриця вхідних даних розміру $m \times n$, де n – кількість коефіцієнтів регресійної залежності; рядки з пропусками значень (**NaN**) під час регресійного аналізу не враховуються;

Alpha – ймовірність виходу за довірчий коридор.

Статистики розраховуються вірно, якщо **X** містить стовпчик з одиниць, тобто якщо в лінійній регресійній моделі є константа. Розрахунки здійснюються за припущенням, що помилки розподілені за нормальним законом.

Для одно- та дво- факторного регресійного аналізу можна застосовувати функцію fit або відповідний графічний модуль Curve Fitting Tool. Модуль запускається через верхнє меню APPS. Вікно цього модуля після завантаження даних про залежність паливної ефективності легкового автомобіля наведено на рис. 12. Тестова вибірка завантажена через пункт Specify validation test меню Fit. На рис. 12 також наведено результати лінійного регресійного аналізу, зокрема:

- коефіцієнти регресійної моделі та довірчі інтервали – в полі Results;
- критерії точності регресійної моделі *SSE* (2) та *RMSE* (8) на навчальній та тестовій вибірках – в полі Table of Fits;
- поверхня відклику та експериментальні дані.

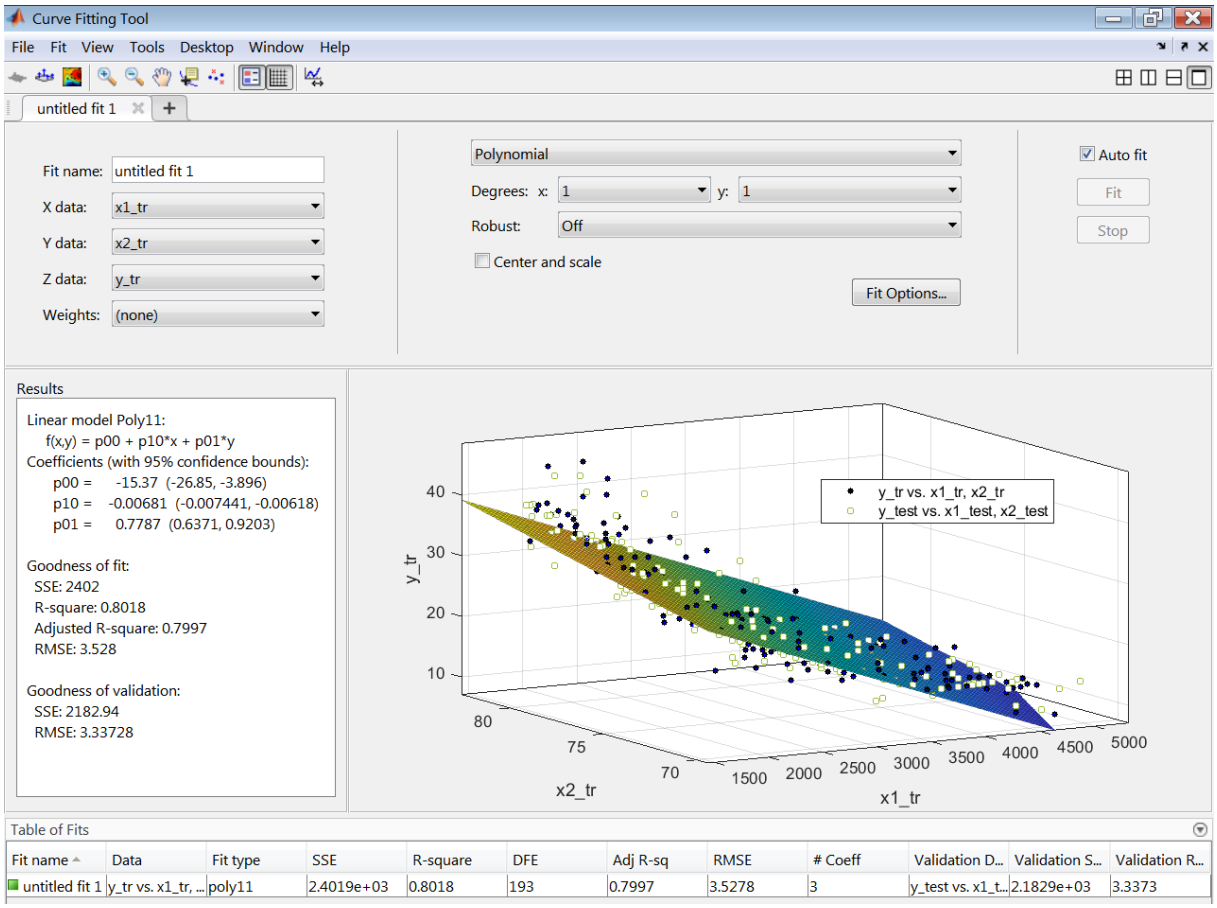


Рисунок 12 – Лінійний регресійний аналіз в модулі Curve Fitting Tool

В модулі Curve Fitting Tool можна проводити і нелінійний регресійний аналіз, задавши відповідні порядки при x_1 та x_2. Також можна задати діапазон можливих значень кожного коефіцієнту. Для прикладу на рис. 13 зображена процедура квадратичного регресійного аналізу без врахування

взаємодії змінних x_1 і x_2 та із забороною на додатне значення коефіцієнту при x_1^2. Окрім поліноміальної моделі для однофакторної регресії в Curve Fitting Tool можна використовувати інші типи: експоненційні, дробові, ряди Фур'є тощо. Можна апроксимувати і за власною моделлю, обравши пункт Custom Equation.

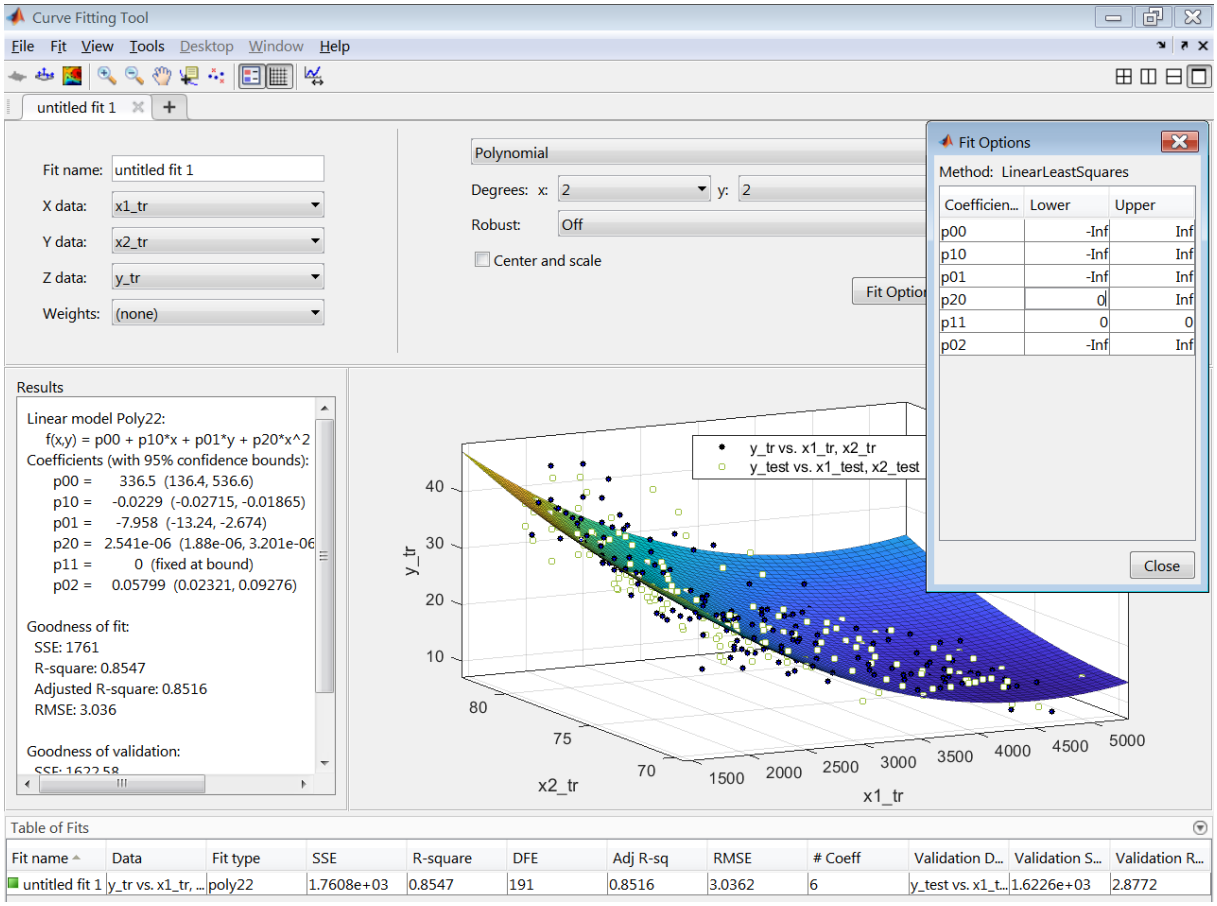


Рисунок 13 – Квадратичний регресійний аналіз в модулі Curve Fitting Tool

Curve Fitting Tool дозволяє побудувати не лише регресійну криву чи регресійну поверхню, але і довірчий коридор. Для прикладу на рис. 14 наведено довірчий коридор для квадратичної залежності паливної ефективності легкового автомобіля y від його маси x за довірчої ймовірності 0.95.

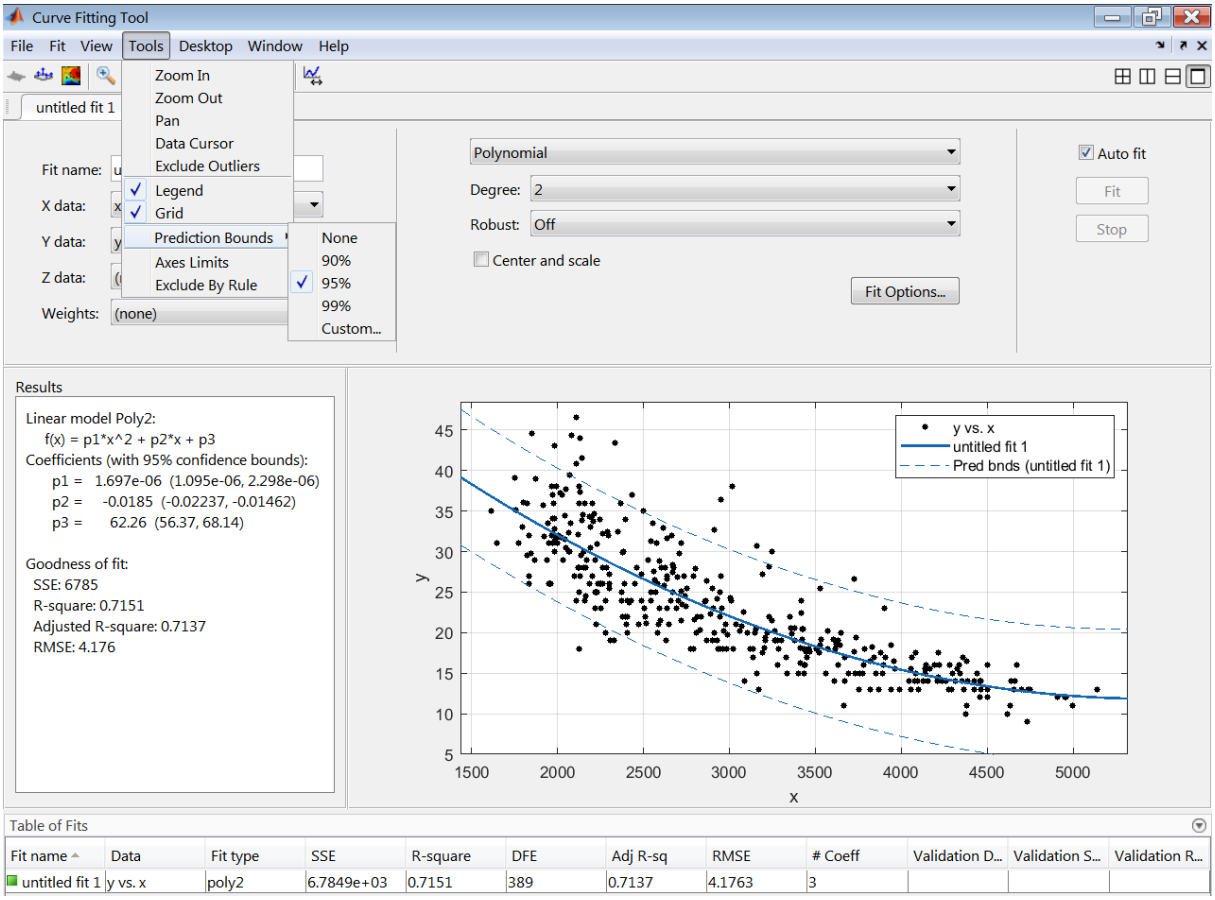


Рисунок 14 – Довірчий коридор однофакторної квадратичної регресії в модулі Curve Fitting Tool

Якщо регресійний аналіз потрібно зробити більше ніж за двома факторами, тоді Curve Fitting Tool не підходить. В цьому випадку підбір регресійних коефіцієнтів можна запрограмувати з використанням операції \ або функції regress. Інший варіант – викликати графічний модуль Regression Learner з верхнього меню APPS. Дані для Regression Learner задаються в іншому форматі, ніж для Curve Fitting Tool – вибірка даних має бути представлена єдиним масивом. Призначення з цього масиву вхідних та вихідної змінних здійснюється в самому Regression Learner. Модуль також містить трохи іншу бібліотеку регресійних моделей, ніж Curve Fitting Tool. Відрізняються і засоби візуалізації, зокрема можна зобразити відповідність між експериментальними та теоретичними результатами в форматі рис. 10.

1.3. Викиди та робастна регресія

Метод найменших квадратів чутливий до викидів – до експериментальних даних, які суттєво відрізняються від інших. Викиди

виникають як наслідок збою вимірювальних приладів, описок операторів, помилкової конвертації даних тощо. Наприклад, оператор поставив кому не в тому розряді десяткового числа або вказав ціну в доларах, замість гривень. Навіть 1 викид може відчутно змістити регресійну модель. Це відбувається через те, що залишок для викиду є великим. А за формулою (2) цей великий залишок підноситься до квадрату, і стає ще більшим. Щоб мінімізувати суму квадратів, необхідно в першу чергу зменшити такі великі залишки, тобто наблизити регресійну залежність до викиду. На рис. 14 червоним зображена лінійна регресія у випадку даних з одним викидом з координатами (8, 0). Лінія регресії суттєво відхилилася від загального тренду в даних.

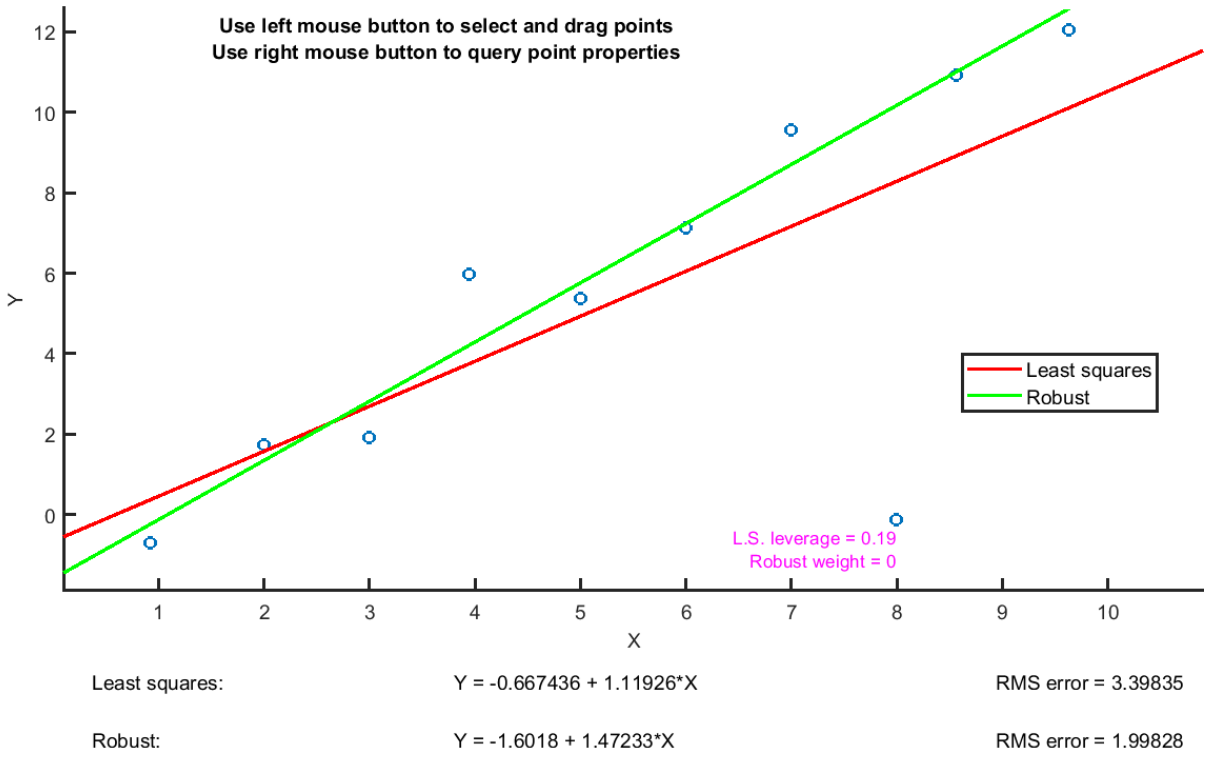


Рисунок 15 – Робастний регресійний аналіз в демо-модулі Robusdemo

Як здійснювати регресійний аналіз за даними з викидами? Якщо досліджуються одно- чи дво- факторні залежності, тоді викиди можна виявити візуально. А як бути з багатофакторними залежностями? Коли викид лише один, його виявити досить просто на основі таких міркувань. Чи мають змінитися регресійні коефіцієнти та *RMSE*, якщо з навчальної вибірки вилучимо одне спостереження? Звичайно, суттєвих змін не

повинно бути. Але якщо вилученим спостереженням є викид, тоді суттєво зменшиться *RMSE* та зміняться регресійні коефіцієнти. Відповідно, послідовно вилучаючи по одному спостереженню і здійснюючи регресійний аналіз на вибірці довжиною (*M*-1), можна виявити викид.

Якщо є кілька викидів, тоді суттєво зростає складність їх виявлення за допомогою вилучення з вибірки пар або трійок спостережень. В цьому випадку застосовують інший підхід, який полягає у зважуванні спостережень. Зважена нев'язка розраховується таким чином:

$$SWSE = \sum_{j=1,\,M} w_j \cdot \left(y_j - f(X_j)\right)^2,$$

де $w_j \in [0,1]$ – вага *j*-го спостереження.

Чим більше залишок для *j*-го спостереження, тим більше шансів, що воно є викидом. Відповідно, великий внесок залишку за цим спостереженням у нев'язку моделі слід скомпенсувати малим значенням w_j. Саме за такою ідеєю і працюють алгоритми робастної (*robust*) регресії. Робастна регресія виконується за кілька ітерацій. Спочатку здійснюється звичайний регресійний аналіз. Потім кожному спостереженню призначається вага за принципом, чим більший залишок, тим менша вага, і виконується регресійний аналіз за критерієм (2). Корекцію ваг та регресійний аналіз повторюють кілька разів. В результаті отримають робастну модель, яка нечутлива до викидів. Така модель на рис. 15 зображена зеленою лінією. Вона знехтувала викидом, призначивши для нього нульову вагу.

Існує кілька методів робастної регресії, які відрізняються правилами корекції ваг спостережень. Виконати робастний регресійний аналіз можна в Curve Fitting Tool за допомогою кнопки Robust. Доступні 2 методи: LAR (Least Absolute Residual method) та Bisquare (Biweight). Більше методів, а саме, 7, доступні, якщо використовувати функцію robustfit. Також в Curve Fitting Tool викиди можна вилучити вручну мишкою або вказавши діапазони викидів. Для цього в меню Tools слід обрати пункти Exclude Outliers та Exclude by Rule, відповідно.

1.4. Функції для регресійного аналізу в системі MATLAB

Для виконання регресійного аналізу та дослідження його результатів доцільно використовувати такі функції та команди:

find – знаходження номерів елементів масива, які задовольняють деяку умову;

fit – відновлення з даних одно- та дво- факторних залежностей;

hold on – продовження виведення графіки в поточне вікно без затирання попередніх зображень;

legend – підписування графіків;

max – значення та порядковий номер максимального елементу масива;

mean – середнє арифметичне координат вектора; для матриці – середнє арифметичне по кожному стовпчику;

min – значення та порядковий номер мінімального елементу масива;

norm – норма вектору (за замовчуванням евклідова відстань);

plot – побудова двовимірних графіків за точками;

plot3 – побудова трьохвимірних графіків за точками;

polyfit – розрахунок коефіцієнтів одновимірної поліноміальної регресії.

regress – розрахунок коефіцієнтів та статистик багатовимірної лінійної регресії;

robustfit – розрахунок коефіцієнтів та статистик робастної регресії;

scatter3 – виведення точок в трьохвимірному просторі;

setdiff – різниця множин;

sqrt – корінь квадратний;

std – середньоквадратичне відхилення координат вектора; для матриці – середньоквадратичне відхилення елементів по кожному стовпчику;

subplot – розбиття графічного вікна на комірки;

sum – сума координат вектора; для матриці – сума

елементів по кожному стовпчику;

surf – побудова зафарбованих поверхонь;

title – виведення заголовку графічного вікна;

unique – вилучення з масива дублюючих елементів;

xlabel – підписування вісі абсцис;

xlim – встановлення межі відображення графіки по вісі абсцис;

ylabel – підписування вісі ординат;

ylim – встановлення межі відображення графіки по вісі ординат;

zlabel – підписування вісі аплікат;

zlim – встановлення межі відображення графіки по вісі аплікат;

1.5. Питання для самоконтролю та розвитку

1. Що таке регресійний аналіз?
2. Яке відношення до регресійного аналізу має Карл Гаус?
3. Чому регресійна модель називається регресійною?
4. Що таке принцип зовнішнього доповнення?
5. Яким чином можна реалізувати принцип зовнішнього доповнення?
6. Навіщо дані розбивають на тестову та навчальну вибірку?
7. За якими критеріями перевіряють репрезентативність (однорідність) вибірок даних?
8. За якими критеріями слід обирати регресійну модель?
9. Яка складність алгоритму розрахунку коефіцієнтів регресії?
10. Як визначити достовірність коефіцієнтів регресії?
11. Що таке викиди в експериментальних даних і як вони впливають на результати регресійного аналізу.
12. Як під час регресійного аналізу врахувати різну достовірність початкових даних?

2. КЛАСИФІКАЦІЯ І КАТЕГОРИЗАЦІЯ

2.1. Ази

Задача класифікації полягає у віднесені об'єкта з ознаками $(x_1, x_2, ..., x_n)$ до одного класу з множини $\{l_1, l_2, ..., l_m\}$. Замість слова *ознаки* часто вживають слова *вхідні атрибути* або просто *атрибути*. З математичної точки зору класифікація – це відображення виду $X = (x_1, x_2, ..., x_n) \to y \in \{l_1, l_2, ..., l_m\}$. До класифікації зводяться різноманітні задачі прийняття рішень та розпізнавання образів в інженерному проектуванні, військовій справі, менеджменті, соціології, освіти, політиці, спорті, в медичній, технічній і економічній діагностиці тощо.

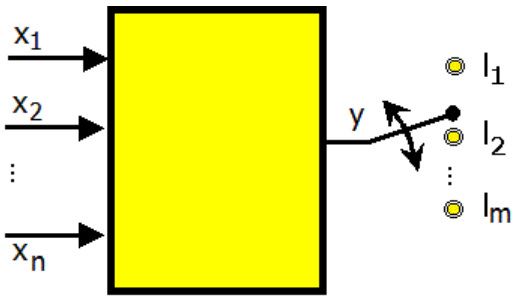


Рисунок 16 – Задача класифікації

В машинному навчанні найвідомішою задачею класифікації є флористична задача про фішерівські іриси. На цій задачі Рональд Фішер в 1936 р. протестував свій метод дискримінатного аналізу. Задача полягає у віднесені ірису до одного із трьох класів: ірис сетоса (Iris-setosa); ірис веселковий (Iris-versicolor); ірис віржиніка (Iris-virginica) (рис. 17). Вибірка складається із 150 рядків – по 50 на кожний клас. Вона доступна за посиланням: http://archive.ics.uci.edu/ml/datasets/Iris. Експериментальні дані зібрав Едгард Андерсон. Класифікація здійснюється за такими чотирма ознаками: x_1 – довжина чашолистка; x_2 – ширина чашолистка; x_3 –

довжина пелюстки; x_4 – ширина пелюстки. Двофакторні розподіли ірисів за класами наведено на рис. 18. Видно, що ірис сетоса лінійно відділяється від інших за ознаками x_1 та x_2, або за пороговим рівнем ознаки x_3 або x_4.

Iris-setosa Iris-versicolor Iris-virginica

Рисунок 17 – Фото ірисів з сайту http://www.signa.org

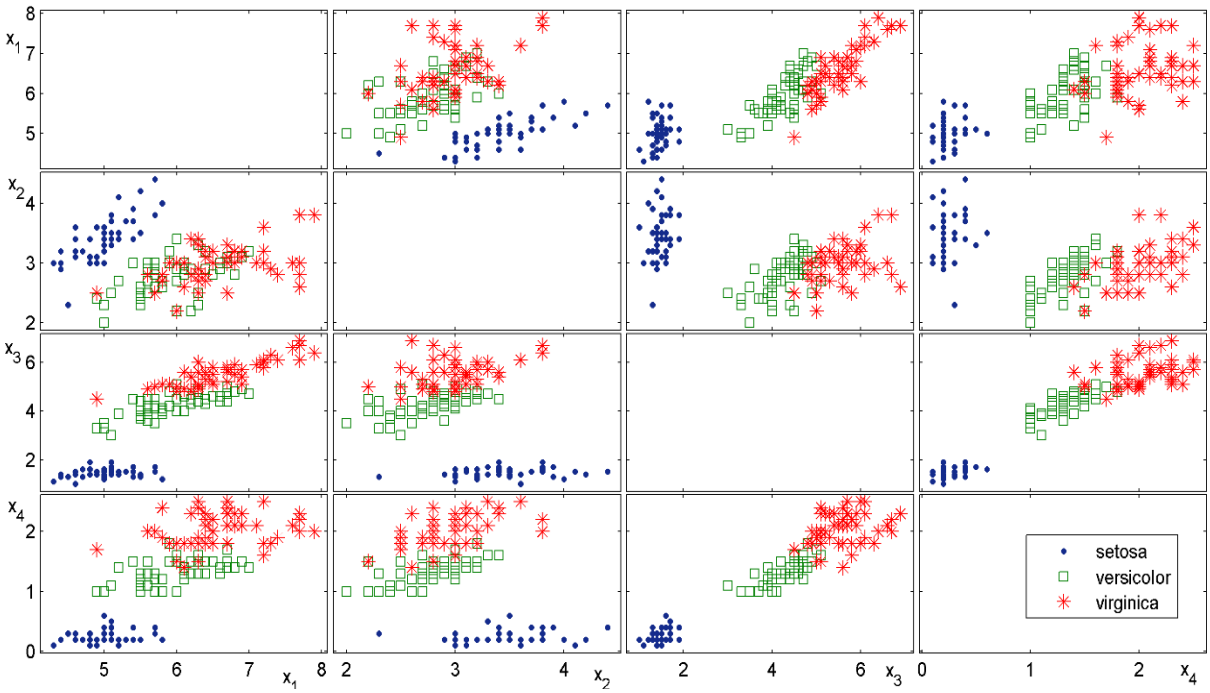


Рисунок 18 – Двофакторні розподіли ірисів за класами

Алгоритм, за яким здійснюється класифікація будемо називати класифікатором. Вхідні змінні, що подаються на класифікатор, можуть приймати значення з різних шкал. Найчастіше зустрічаються такі типи шкал даних:

числова, наприклад, діапазон $[0, 24]$ для оцінювання середньої тривалості перебування особи в соціальній мережі протягом доби;

категоріальна або номінальна, наприклад, множина {чоловіча, жіноча} для опису статі людини або множина {червоний, жовтий, зелений} для опису кольору. Частинним випадком категоріальної шкали є бінарна, коли атрибути приймають одне із двох можливих значень, наприклад, з множини {0, 1}, {ні, так}, {немає, є};

порядкова, в якій значення впорядковані якимось чином, наприклад, від меншого до більшого або від гіршого до кращого, наприклад, множина {незадовільно, задовільно, добре, відмінно} для оцінювання знань студента.

Над даними з числової шкали можна виконувати арифметичні операції. Для цієї шкали існують відношення еквівалентності та порядку, тобто над числовими даними можна виконати логічні операції дорівнює, більше та менше. Для категоріальної шкали допустиме лише одне відношення – відношення еквівалентності. Над даними з порядкової шкали можна виконувати логічні операції дорівнює, більше або менше, але арифметичні операції є недопустимі, навіть якщо для запису порядкових значень використовуються числа.

Існують і специфічні шкали зі значеннями у формі інтервалу, нечіткого числа чи ймовірнісного розподілу. Також, можливі задачі, коли один і той самий атрибут приймає значення з різних шкал, зазвичай, з порядкової та числової. Наприклад, рівень впливовості одного користувача може буде задано порядковим значенням «Високий», а іншого – деяким числом з діапазону [0, 100]. Специфічною є і лінгвістична шкала великої потужності, деякі елементи якої є залежними, наприклад, є повними чи частковими синонімами.

Класифікатори бувають параметричними та непараметричними. В параметричних класифікаторах наперед визначена модель прийняття рішень, наприклад, деяке рівняння границі класів. Задача проектування класифікатора полягає в знаходженні таких параметрів цієї моделі, які забезпечують найкращу якість прийняття рішень. В непараметричних класифікаторах структура моделі прийняття рішень визначається не суб’єктивно, а за деяким алгоритмом під час аналізу навчальної вибірки.

Останнім часом все більше уваги приділяються питанням створення багатозразкових та багатоярликових класифікаторів.

Багатозразковий класифікатор (multiple instance classifier) здійснює класифікацію не для одного об`єкту, а для кількох пов'язаних об'єктів одночасно. Сукупність таких пов'язаних об'єктів часто називають мішком (bag). Ключовою особливістю є те, що під час навчання чи функціонування, увесь цей мішок має бути віднесено до одного класу. В цьому принципова відмінність від багатократного застосування до мішка звичайного класифікатора, коли окремі елементи мішка можуть бути віднесені до різних класів. Така багатозразкова класифікація доцільна в задачах аналізу текстів в соціальних мережах. Наприклад, є група текстів одного автора, написаних в різних жанрах чи у різний час. Відповідно, під час класифікації увесь цей мішок текстів має бути віднесено до одного класу. Інший приклад – фотографії одного і того ж об'єкту чи події, які зроблені під різним кутом, за різного освітлення, у різний час, з різною кількістю учасників тощо. І всі вони мають бути віднесені до одного класу.

Багатоярликовий класифікатор (multi-label classifier) функціонує таким чином, що кожен об'єкт може бути віднесено не одного класу, а до кількох. Багатоярликову класифікацію також називають категоризацією. Наприклад, наукова стаття може бути віднесена до кількох наукових областей. Нахабний коментар в соціальній мережі може бути віднесено до таких 6 класів [88]: toxic – токсичний, severe toxic – дуже токсичний, obscene – непристойний, threat – погрозливий, insult – образливий та identity hate – зневажливий до ідентичності. Одночасно нахабний коментар може належати до кількох класів, наприклад, бути і непристойний, і погрозливим, і містити зневагу до ідентичності. Попарне суміщення класів нахабних коментарів наведено на рис. 19. На цьому рисунку площа фігури пропорційна кількості коментарів. Візуалізація зроблена за розподілом 16225 нахабних коментарів з Вікіпедії з даних kaggle-змагання "Toxic Comment Classification Challenge".

Класифікація може бути детермінована, рангова чи ймовірнісна (або нечітка). За детермінованої класифікації рішенням є лише клас до якого віднесено об'єкт. У випадку категоризації – це список класів. За рангової класифікації можливі класи впорядковуються, тобто вказується ранги класів. Обрізаючи цей ранговий список знизу виходимо на категоризацію.

Рангова класифікація за двох класів неможлива. За ймовірносної або нечіткої класифікації вказується числова міра належності до кожного класу. Для одноярликої класифікації сума ймовірностей належності до класів має дорівнювати одиниці. У випадку нечіткої класифікації умова нормування суми належностей на 1 необов'язкова. За багатоярликої класифікації ця різниця між нечіткою та ймовірнісною класифікацією зникає.

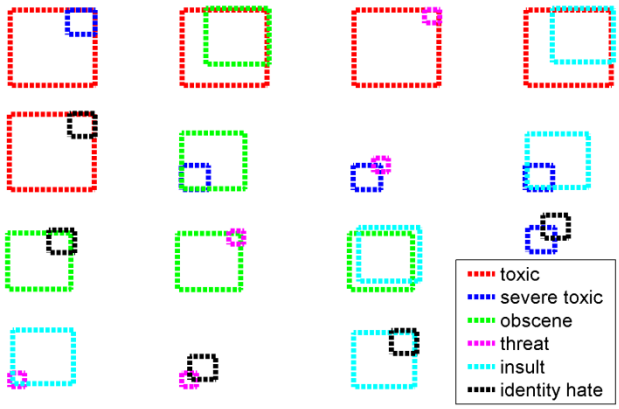


Рисунок 19 – Попарне суміщення класів нахабних коментарів в kaggle-задачі "Toxic Comment Classification Challenge"

На рис. 20 наведено приклад нечіткої категоризації п'яти документів на базисний словник ключових слів. Видно, що документ №5 одночасно належить до четвертого, шостого та восьмого класів, але з різними ступенями (μ). До третього, дев'ятого, десятого та одинадцятого класів не віднесено жодного документа.

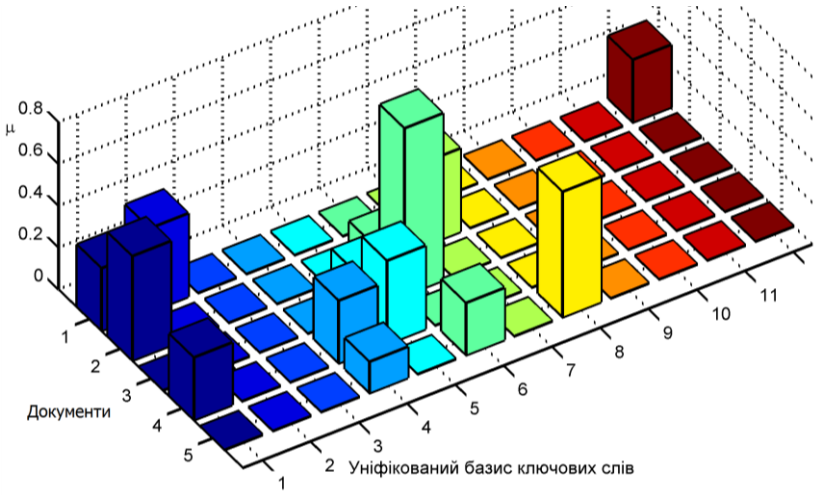


Рисунок 20 – Приклад нечіткої категоризації

Класифікатор будемо створювати лише за вибіркою експериментальних даних:

$$(X_r, y_r), \quad r = \overline{1, M}, \tag{10}$$

де $X = (x_{r1}, x_{r2}, ..., x_{rn})$;

$y_r \in \{l_1, l_2, ..., l_m\}$;

M – довжина вибірки.

У вибірці (10) вхідні змінні можуть приймати значення з різних шкал.

Якість класифікатора оцінюють за критеріями складності, вартості, інтерпретабельності та точності. *Складність* класифікатора визначають за кількістю елементарних операцій, які необхідно виконати для прийняття рішення. *Вартість* визначається витратами на збір початкових даних, необхідних для прийняття рішення. Кожен додатковий атрибут, який використовується для прийняття рішень, вимагає витрат деяких ресурсів – наприклад, проведення лабораторних досліджень для медичної діагностики. Тому, намагаються розробити класифікатор таким чином, щоб дорогі атрибути використовувати рідко. Інтерпретабельність пов'язують з наочністю моделі. Вона віддзеркалює наскільки процес прийняття рішень є зрозумілим фахівцям з предметної області, які не мають спеціальної кібернетичної підготовки. Точність класифікатора F зазвичай визначають за частотою помилок:

$$MCR(F) = \frac{\sum\limits_{j=\overline{1, M}} \Delta_j}{M},$$

де $\Delta_j = \begin{cases} 1, & якщо\ y_j \neq F(X_j) \\ 0, & якщо\ y_j = F(X_j) \end{cases}$.

2.2. Дерево рішень

Одним із найпростіших непараметричних класифікаторів є дерево рішень. Дерево рішень (*decision tree*) є однією з найпопулярніших моделей класифікації. Воно являє собою ієрархічний набір правил "Якщо – тоді – інакше" у формі орієнтованого дерева. Зазвичай використовують бінарне

дерево рішень, з кожної нетермінальної вершини якого виходять 2 дуги (рис. 21). На рисунку дуги направлені зверху вниз. Корінь дерева розміщено зверху. Листки дерева відповідають класам рішень, причому одному класу може належати кілька листків. На рис. 21 класу “Virginica” відповідають 2 листка, а класам “Setosa” та “Versicolor” – по одному. Нетермінальні вершини відповідають логічним умовам, за якими аналізуються вхідні атрибути. Щоб визначити клас деякого об’єкту потрібно в нетермінальних вершинах дерева відповісти на питання типу: “Значення ознаки *x* менше *A*?” або “Значення ознаки *x* дорівнює *A* або *B*”. Якщо відповідь “так”, здійснюється перехід до лівої вершини наступного рівня дерева, якщо “ні”– до правої вершини. Рух по дереву продовжується до тих пір, поки не потрапимо на один із його листків.

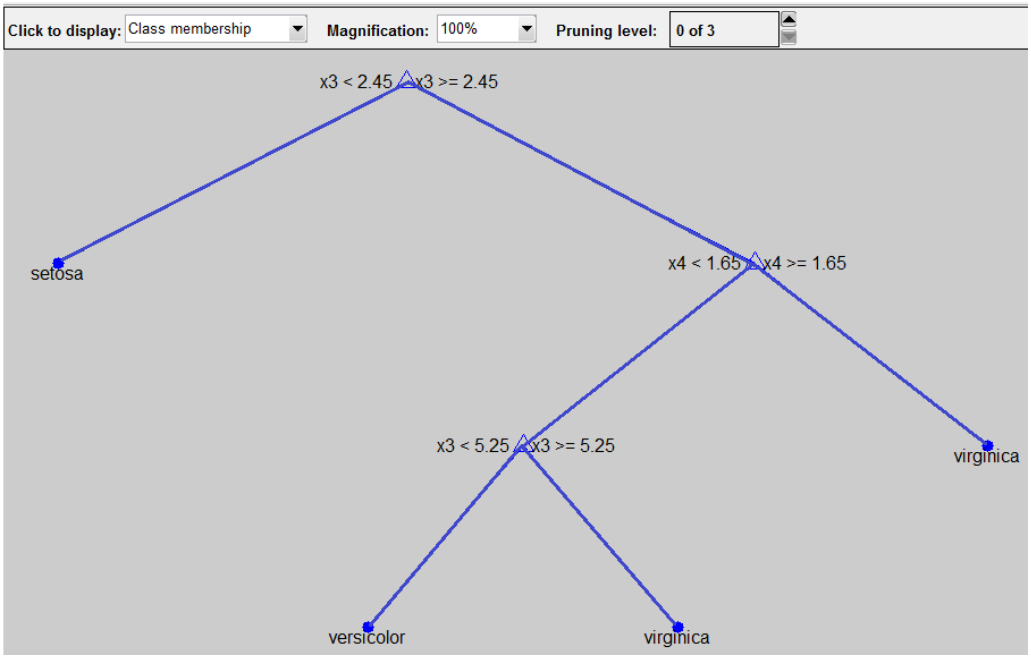


Рисунок 21 – Дерево рішень для класифікації ірисів в форматі MATLAB

Популярність дерев рішень для ідентифікації реальних залежностей обумовлена наочністю і зрозумілістю моделі, простотою процедури класифікації, а також можливістю використання як числових, так і порядкових та категоріальних атрибутів. Перевагою є і те, що синтез та оптимізація дерев рішень відбуваються швидко навіть для великих вибірок даних. При цьому, досліднику не потрібно вказувати, які атрибути є інформативні, а які – ні. Формування переліку інформативних атрибутів здійснюється покроково під час синтезу дерева рішень.

Для синтезу дерев рішень зазвичай використовують жадібні алгоритми. Це швидкі однопрохідні алгоритми, за якими дерево рішень будують від кореня до листків. Розпочинають з пенька – дерева з одного листка. На кожній ітерації 1 листок дерева розщеплюють – перетворюють на проміжну вершину з двома новими листками. При цьому на кожній ітерації обирається локально найкраще продовження, без аналізу наслідків прийнятого рішення. Найбільш відомими жадібними алгоритмами синтезу дерева рішень є CART та C4.5. Під час синтезу дерева рішень потрібно знати відповіді на такі 4 питання.

1. Як згенерувати варіанти логічних умов для листка, що розщеплюється?

2. Якими класами розмітити нові листки?

3. Яким чином відібрати кращий варіант продовження дерева?

4. Коли зупинитися?

Нижче розглянемо відповіді на поставлені питання. Розпочнемо з генерації варіантів логічної умови. В логічній умові фігурує лише один із n атрибутів – $x_1, x_2, ..., x_n$. Усі атрибути є незалежними, відповідно, має множина варіантів логічних умов з атрибутом x_1, з атрибутом x_2, …, з атрибутом x_n. Атрибути можуть задаватися на різних шкалах, відповідно процедури генерування варіантів будуть різними.

Розглянемо ситуацію, коли атрибут x_i, $i = \overline{1, n}$ задано на числовій шкалі, наприклад, $x_i \in [0, 10]$. Нам потрібно згенерувати логічну умову, яка розділяє інтервал $[0, 10]$ на дві частини. Варіантами логічної умови можуть бути такі: $x_i < 5$, $x_i < 6$, $x_i < 5.5$, $x_i < 5.499$, $x_i < 5.4999999999$ тощо. Скільки може бути таких варіантів – на перший погляд нескінчена кількість. Але нам потрібно не лише розбити інтервал, а зробити так, щоб логічна умова розділила експериментальні дані по двом листкам дерева рішення. Деяка частина даних з навчальної вибірки має потрапити на лівий листок, а деяка – на правий. Якщо логічні умови $x_i < 5.5$, $x_i < 5.499$ та $x_i < 5.4999999999$ призводять до однакового розподілу даних по листкам, тоді між ними немає різниці і їх можна розглядати як одну.

Для формування варіантів відсортуємо значення x_i для об’єктів навчальної вибірки, які потрапили в аналізовану проміжну вершину.

Позначимо відсортований ряд цих значень через $(a_1, a_2, ..., a_k)$. Тоді, усі варіанти логічної умови можна записати таким чином: $x_i < a$, де $a \in \left\{ \frac{a_1 + a_2}{2}, \frac{a_2 + a_3}{2}, ..., \frac{a_{k-1} + a_k}{2} \right\}$. За одного можливого значення змінної x_i розщеплення за цим атрибутом буде неможливим. За двох можливих значень змінної x_i буде 1 варіант логічної умови. Максимальна кількість варіантів дорівнює $M - 1$, що може бути тільки при формуванні першої проміжної вершини. При цьому в навчальні вибірці усі значення змінної x_i мають бути унікальними, тобто $k = M$. Максимальна кількість варіантів логічної умови для однієї вершини дорівнює $n \cdot (M - 1)$.

Розглянемо випадок, коли атрибут x_i задано на порядковій шкалі. Для формування правої частини логічної умови $x_i < a$ відсортуємо за зростанням значення x_i для об'єктів навчальної вибірки, які потрапили в проміжну вершину. Позначимо відсортований ряд цих значень через $(a_1, a_2, ..., a_k)$. Тоді $a \in \{a_2, a_3, ..., a_k\}$.

Якщо атрибут x_i задано на категоріальній шкалі, тоді сформуємо множину B значень x_i для об'єктів навчальної вибірки, що потрапили в проміжну вершину. Логічну умову в цій проміжній вершині запишемо так: $x_i \in A$, де A деяка непуста підмножина з B, причому $0 < |A| < |B|$. Серед будь-яких двох варіантів A_1 та A_2 не має бути «дзеркальних», для яких $A_1 \cap A_2 = \varnothing$ та $A_1 \cup A_2 = B$. Розглянемо приклад формування правої частини логічної умови. Нехай B={Банан, Лимон, Яблуко, Диня}, тоді можливими варіантами правої частини логічної умови будуть такі:

A_1={Банан};

A_2={Лимон};

A_3={Яблуко};

A_4={Диня};

A_5={Банан, Лимон};

A_6={Банан, Яблуко};

A_7={Банан, Диня}.

При появі нових листків їм потрібно поставити мітки, тобто вказати належність до того чи іншого класу. Очевидно, що 2 сусідніх листки

повинні мати різні мітки. Мітка відповідає тому класу, об'єкти якого домінують на листку. Наприклад, з навчальної вибірки на листок v_1 потрапило:

9 об'єктів класу l_1;

5 об'єктів класу l_2;,

2 об'єкти класу l_3;

А на листок v_2 потрапило:

4 об'єкти класу l_1;

2 об'єкти класу l_2;

8 об'єктів класу l_3.

Тоді листок v_1 помітимо як l_1, а листок v_2 як l_3.

Під час екстракції дерева рішень логічна умова в новій проміжній вершині формується так, щоб забезпечити екстремальне значення деякого локального критерія ефективності. Популярними критеріями є змішаність Джині (Gini impurity) та ентропія. Інколи застосовують і двійкове правило (twoing rule).

Чим сильніше перемішані на сусідніх листках об'єкти різних класів, тим гірша якість відповідної логічної умови. Змішаність Джині для вершини v, з якою пов'язані листки v_1 та v_2, розраховується так:

$$G(v) = \frac{N_1}{N}\left(1 - \sum_{j=\overline{1,m}} p_{1j}^2\right) + \frac{N_2}{N}\left(1 - \sum_{j=\overline{1,m}} p_{2j}^2\right),$$

де N_1 та N_2 кількість об'єктів навчальної вибірки, які потрапили на листки v_1 та v_2;

N – кількість об'єктів навчальної вибірки, які потрапили на проміжну вершину v, $N = N_1 + N_2$;

p_{1j} – частота потрапляння на листок v_1 об'єктів з класу l_j, $j = \overline{1,m}$;

p_{2j} – частота потрапляння на листок v_2 об'єктів з класу l_j, $j = \overline{1,m}$.

Ентропія – це міра безладу. Абсолютний порядок на листку дерева рішень буде, коли на нього потрапили об'єкти лише одного класу. Абсолютний безлад буде, коли на листку всі класи представлені рівномірно. Відповідно, слід обрати той варіант логічної умови, який

найсильніше покращує порядок, найбільше зменшує ентропію. Для цього спочатку розрахуємо ентропію навчальної вибірки:

$$I = -\sum_{j=\overline{1,m}} p_j \cdot \log_2(p_j),$$

де p_j – частота об'єктів класу l_j у вибірці, $j = \overline{1,m}$;

Далі розрахуємо ентропію кожного варіанту логічної умови *v*:

$$I(v) = -\frac{N_1}{N} \sum_{j=\overline{1,m}} p_{1j} \cdot \log_2(p_{1j}) - \frac{N_2}{N} \sum_{j=\overline{1,m}} p_{2j} \cdot \log_2(p_{2j}).$$

Кращою буде логічна умова, яка забезпечує максимальне зменшення ентропії:

$$\Delta I(v) = I - I_v \to \max.$$

За двійкового правила критерій розщеплення розраховується так:

$$T(v) = \frac{N_1 N_2}{N^2} \left(\sum_{j=\overline{1,m}} abs(p_{1j} - p_{2j}) \right)^2.$$

Максимальне значення $T(v)=1$ відповідає випадку, коли листки мають однакову кількість об'єктів і на кожному листку є об'єкти лише одного класу. За ентропійним правилом також обирається логічна умова, яка забезпечує приблизно однакову кількість об'єктів на сусідніх листках. Правило розщеплення за змішаністю Джині формує листки дерева таким чином, щоб ізолювати об'єкти найпоширенішого класу.

Приклад. В вершину *v* потрапило $N = 200$ об'єктів, по 100 класу l_1 та класу l_2. Можливі 2 варіанти логічної умови в вершині *v*, розподіли за якими наведено в табл. 3.

Таблиця 3 – Розподіл об’єктів по сусіднім листкам v_1 та v_2

Варіант логічної умови	v_1		v_2	
	l_1	l_2	l_1	l_2
I	5	50	95	50
II	5	90	95	10

Вершина v є абсолютно забрудненою – співвідношення між об’єктами різних класів складає 1:1. За першим варіантом логічної умови отримуємо в вершині v_1 рівень забруднення 1:10 та в вершині v_2 – приблизно 2:1. За другим варіантом логічної умови отримуємо рівень забруднення 1:18 в вершині v_1 та 9.5:1 в вершині v_2. Очевидно, що другий варіант є кращим. Порівняємо цей висновок з результатами вибору варіанту логічної умови за різними правилами розщеплення.

Змішаність Джині першого варіанта логічної умови дорівнює:

$$G_{\text{I}} = \frac{55}{200}\left(1-\left(\frac{5}{55}\right)^2-\left(\frac{50}{55}\right)^2\right)+\frac{145}{200}\left(1-\left(\frac{95}{145}\right)^2-\left(\frac{50}{145}\right)^2\right)=$$
$$= \frac{55}{200}\cdot 0.1653+\frac{145}{200}\cdot 0.4518 = 0.373.$$

Змішаність Джині другого варіанта логічної умови дорівнює:

$$G_{\text{II}} = \frac{95}{200}\left(1-\left(\frac{5}{95}\right)^2-\left(\frac{90}{95}\right)^2\right)+\frac{105}{200}\left(1-\left(\frac{95}{105}\right)^2-\left(\frac{10}{105}\right)^2\right)=$$
$$= \frac{95}{200}\cdot 0.0997+\frac{105}{200}\cdot 0.1723 = 0.1378.$$

Видно, що $G_{\text{II}} < G_{\text{I}}$, тому, як і очікувалось, другий варіант є кращим.

За ентропійним критерієм спочатку розрахуємо початкову ентропію:

$$I = -\frac{100}{200}\log_2\left(\frac{100}{200}\right)-\frac{100}{200}\log_2\left(\frac{100}{200}\right) = -0.5\cdot(-1)-0.5\cdot(-1) = 1.$$

Як і очікувалося, за повного безладу на проміжній вершині її ентропія максимальна.

Ентропія для першого варіанта логічної умови дорівнює:

$$
\begin{aligned}
I_{\mathrm{I}} &= -\frac{55}{200}\left(\frac{5}{55}\log_2\left(\frac{5}{55}\right)+\frac{50}{55}\log_2\left(\frac{50}{55}\right)\right)- \\
&- \frac{145}{200}\left(\frac{95}{145}\log_2\left(\frac{95}{145}\right)+\frac{50}{145}\log_2\left(\frac{50}{145}\right)\right)= \\
&= -0.275(0.09\cdot(-3.4739)+0.91\cdot(-0.1361))- \\
&- 0.725(0.66\cdot(-0.5995)+0.34\cdot(-1.5564))= \\
&= 0.275\cdot 0.4366+0.725\cdot 0.9249=0.7906.
\end{aligned}
$$

Ентропія для другого варіанта логічної умови дорівнює:

$$
\begin{aligned}
I_{\mathrm{II}} &= -\frac{95}{200}\left(\frac{5}{95}\log_2\left(\frac{5}{95}\right)+\frac{90}{95}\log_2\left(\frac{90}{95}\right)\right)- \\
&- \frac{105}{200}\left(\frac{95}{105}\log_2\left(\frac{95}{105}\right)+\frac{10}{105}\log_2\left(\frac{10}{105}\right)\right)= \\
&= -0.475(0.05\cdot(-4.3219)+0.95\cdot(-0.0740))- \\
&- 0.525(0.9\cdot(-0.152)+0.1\cdot(-3.3219))= \\
&= 0.475\cdot 0.2864+0.525\cdot 0.4690=0.3823.
\end{aligned}
$$

Зменшення ентропії становить:

$$\Delta I_{\mathrm{I}} = 1-0.7906=0.2094;$$

$$\Delta I_{\mathrm{II}} = 1-0.3823=0.6177.$$

Видно, що $\Delta I_{\mathrm{II}} > \Delta I_{\mathrm{I}}$, тому, другий варіант є кращим.

За двійкового правила значення критерію для першого та другого варіантів становлять:

$$T_{\mathrm{I}} = \frac{55\cdot 145}{4\cdot 200^2}\left(\left|\frac{5}{55}-\frac{95}{145}\right|+\left|\frac{50}{55}-\frac{50}{145}\right|\right)^2 = 0.0635;$$

$$T_{\mathrm{II}} = \frac{95\cdot 105}{4\cdot 200^2}\left(\left|\frac{5}{95}-\frac{95}{105}\right|+\left|\frac{90}{95}-\frac{10}{105}\right|\right)^2 = 0.811.$$

Видно, що $T_{\mathrm{II}} > T_{\mathrm{I}}$, тому, другий варіант є кращим.

Алгоритм синтезу дерева рішень збільшує кількість проміжних вершин до тих пір, поки не почне виконуватися один із критеріїв зупинки. Сьогодні переважно використовуються 5 критеріїв зупинки.

Перший критерій – досягнення максимальної глибини дерева, яка розраховується як довжина ланцюжка проміжних вершин від кореня до листка.

Другий критерій – досягнення максимальної складності дерева, яка розраховується як сумарна кількість проміжних вершин.

Третій критерій – мінімальна кількість об'єктів навчальної вибірки, які потраплять на один листок дерева. Наприклад, якщо потенційна проміжна вершина дерева рішень породжує листки, в один із яких попадає менше 10 об'єктів, тоді таке розбиття вважається недоцільним і розщеплення не фіксується. Відповідно за цим напрямком припиняється подальше розгалуження, і потенційна проміжна вершина залишається листком. Цей критерій трохи захищає від перенавчання – забороняє синтезувати правила класифікації для малої кількості об'єктів.

Четвертий критерій – мінімальна кількість об'єктів навчальної вибірки на листку, для виконання спроби його розщеплення. Наприклад, на листку не більше 30 об'єктів, тоді він в надалі не розщепляється. Цей критерій виконує ту саму мету, що і попередній.

П'ятий критерій – логічна неможливість подальшого розщеплення листка, наприклад, коли в нього потрапили об'єкти лише одного класу або об'єкти з однаковими значенням усіх ознак.

Синтезоване за жадібними алгоритмами дерево рішень як правило виходить громіздким та заскладним. Для такого дерева частота помилкової класифікації на тестовій вибірці зазвичай значно більша, ніж на навчальній. Для покращення класифікатора дерево рішень підрізають. Підрізання здійснюють або за рівнями – зменшуючи глибину дерева, або за вершинами - видаляючи окремі листки.

Типові залежності частоти помилкової класифікації (MCR) від розміру дерева рішень ($Complexity$) наведені на рис. 21. На навчальній вибірці залежність є монотонно спадною, тоді як на тестовій вибірці спостерігається екстремум. За принципом зовнішнього доповнення моделлю класифікації слід обрати дерево рішень з мінімальною кількістю помилок на тестовій вибірці. На рис. 22 це дерево позначено літерою "А".

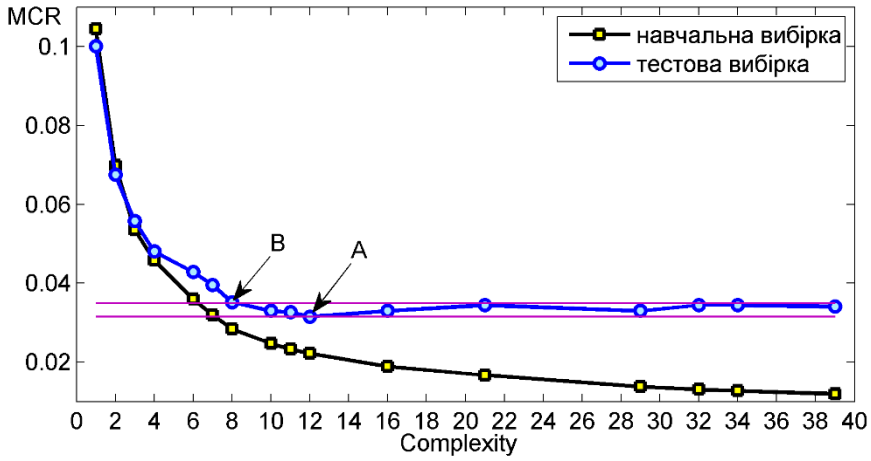


Рисунок 22 – Залежність частоти помилкової класифікації від кількості листків дерева для задачі Page Blocks Classification

Згідно до рис. 22 кращою моделлю класифікації відповідно до алгоритму CART буде обрано не дерево “A”, а більш просте дерево “B”. Обґрунтування такого вибору полягає в тому, що залежності на рис. 22 не є абсолютно достовірними, оскільки значення атрибутів містять деякі шуми. Крім того, криві на рис. 22 чутливі до способу розбиття експериментальних даних на навчальну та тестову вибірки. За іншого розбиття частота помилок класифікації могла б бути іншою. Відповідно на рис. 22 частоти помилкової класифікації оцінено з деякою похибкою. Ця похибка розраховується так:

$$\Delta = \sqrt{\frac{MCR \cdot (1 - MCR)}{M_{test}}}, \qquad (11)$$

де M_{test} – кількість об’єктів тестової вибірки.

За формулою (11) на рис. 22 проведено Δ-коридор, в межах якого класифікатори є статистично нерозрізненими за частотою помилки. Серед моделей однакової безпомилковості слід обирати найпростішу, якою і є класифікатор “B”. Початкове та оптимальне дерева рішень наведені на рис. 23 та 24.

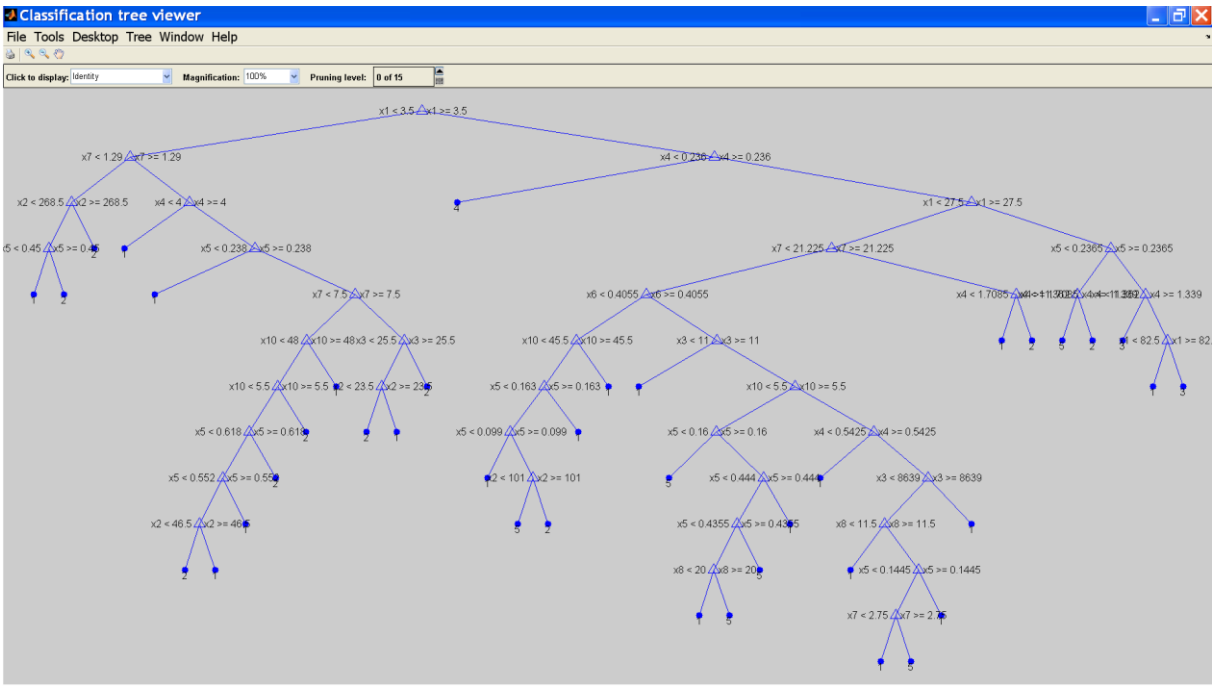


Рисунок 23 – Початкове дерево рішень для задачі Page Blocks Classification в головному вікні модуля Classification tree viewer

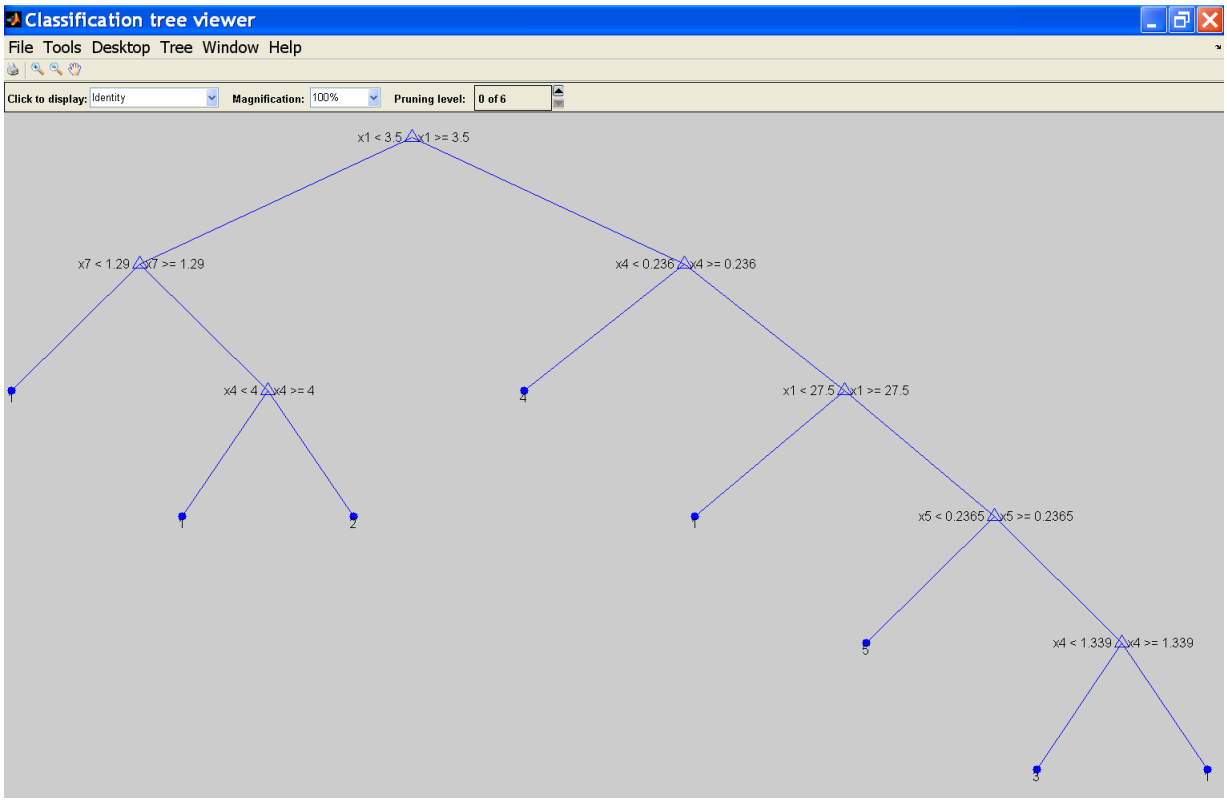


Рисунок 24 – Оптимальне дерево рішень для задачі Page Blocks Classification в головному вікні модуля Classification tree viewer

Аналізуючи класифікатори з рис. 23 та 24 бачимо, що оптимальне дерево не тільки забезпечує кращу безпомилковість, але і має в 5 разів менше логічних умов. Крім того, для класифікації за оптимальним деревом рішень потрібні значення лише 4 ознак, тоді як за початковим деревом – 9. Таким чином, оптимальне дерево краще за критеріями точності, складності, вартості та інтрепретабельності.

В багатьох практичних задачах експериментальні дані містять пропуски – тобто у об'єкта класифікації відсутнє значення деякої ознаки. Пропуски – достатньо поширене явища. Для великих вибірок відсутність пропусків може опосередковано свідчити, що дані підтасовували. Як же діяти за пропусків даних?

Перший варіант – вилучити об'єкти з пропусками з вибірки даних. Але так можна неприпустимо скоротити обсяг початкової інформації, що призведе до зниження достовірності моделі. Крім того, можливий "зсув" моделі, коли об'єкти з пропусками та без пропусків знаходяться в різних зонах факторного простору. Наприклад, в деяких містах опитування здійснювалося за старими анкетами, а в деяких - за новими, з додатковим запитанням. Відповідно, записи з старих анкет містять пропуски. Але вилучивши їх, ми суттєво спотворимо вибірку даних.

Другий варіант – вилучити з вибірки ознаки, що містить пропуски. Але тут можлива ситуація, коли за кожною ознакою хоча б в одному рядку вибірки даних є пропуски. Тоді виходить, що треба вилучити усі ознаки. Можлива й інша ситуація. Зібрана велика вибірка, наприклад, результати анкетування 10 000 осіб. І в одній анкеті респондент не відповів на якесь питання, наприклад, не вказав свій вік. Невже через 1 пропуск ігнорувати відповіді 9 999 респондентів? Очевидно, що шкода втрачати такі дані лише через 1 пропуск.

Третій варіант – замінити пропуски на значення "невідомо", тобто розширити шкалу вимірювань. Такий варіант можна застосувати, якщо пропуск стосується категоріальної ознаки.

Четвертий варіант – замінити пропуски ймовірнісним розподілом. Цей розподіл будується за наявними в вибірці значеннями проблемної ознаки. Сучасні алгоритми здатні синтезувати дерева рішень з урахуванням таких ймовірнісних розподілів випадкових величин.

В середовищі MATLAB для синтезу дерева рішень доцільно використати функцію **fitctree** таким чином:

```
T = fitctree(tr_x, tr_y),
```

де **T** – дерево рішень;

tr_x – вхідні значення навчальної вибірки;

tr_y – вихідні значення навчальної вибірки.

Якщо в даних є пропускі, то у відповідні чарунки матриці **tr_x** слід занести **NaN**. Якщо деякі ознаки є категоріальними, тоді необхідно навести їх перелік. Наприклад, якщо перша і третя ознаки є категоріальними, тоді **fitctree** викликаємо таким чином:

```
T = fitctree(tr_x, tr_y, 'CategoricalPredictors', [1 3]).
```

За замовченням дерево рішень синтезується на основі змішаності Джині. Інші правила розщеплення задаються так:

```
T = fitctree(tr_x, tr_y, 'SplitCriterion', 'deviance');

T = fitctree(tr_x, tr_y, 'SplitCriterion', 'twoing').
```

Інші важливі параметри алгоритму синтезу дерева рішень встановляються за допомогою таких ключів:

'MaxNumSplits' – максимальна кількість логічних умов в дереві;

'MinLeafSize' – мінімальна кількість об’єктів, для яких може бути сформований окремий листок дерева рішень (за замовченням – 1);

'MinParentSize' – мінімальна кількість об’єктів на листку, за якої можливе його подальше розщеплення (за замовченням – 10);

'Weights' – вектор вагових коефіцієнтів спостережень (за замовченням – 1).

Для демонстрації дерева рішень призначена функція view. Формат функції такий:

```
view(T, param, value),
```

де **T** – дерево рішень;

param – назва параметра;

value – значення параметра.

Допустимі назви параметрів:

'mode' – режим виводу – текстовий (**'text'**) або графічний (**'graph'**);

'PruneLevel' – кількість рівнів підрізання дерева.

Для прикладу, виконаємо таку програму:

```
%Завантажуємо дані задачі класифікації ірисів:
load fisheriris;
%Створюємо дерево рішень:
T = fitctree(meas, species, 'PredictorNames', ...
    {'SL' 'SW' 'PL' 'PW'});
view(T);
```

В результаті на екрані отримуємо дерево рішень у вигляді такого списку правил:

```
Decision tree for classification
1  if PL<2.45 then node 2 elseif PL>=2.45 then node 3 else setosa
2  class = setosa
3  if PW<1.75 then node 4 elseif PW>=1.75 then node 5 else
versicolor
4  if PL<4.95 then node 6 elseif PL>=4.95 then node 7 else
versicolor
5  class = virginica
6  if PW<1.65 then node 8 elseif PW>=1.65 then node 9 else
versicolor
7  class = virginica
8  class = versicolor
9  class = virginica
```

Якщо викликати **view(T, 'mode', 'graph')**, з'явиться графічне вікно, яке зображено на рис. 25. Вікно має лічильник Pruning Level за допомогою якого можна інтерактивно підрізати дерево рішень. Дії у цьому вікні не змінюють дерево рішень з робочої області. При виділені вершини курсором миші виводиться інформація згідно до обраного режиму з списку Click to display. Можливі такі режими: Identity – порядковий номер вершини та відповідне правило чи клас; Variable range – межі, що їх можуть приймати атрибути в вершині; Class membership – кількість об'єктів кожного класів, що потрапили у вершину; Estimated probability – ймовірність появи кожного класу в вершині (рис. 26).

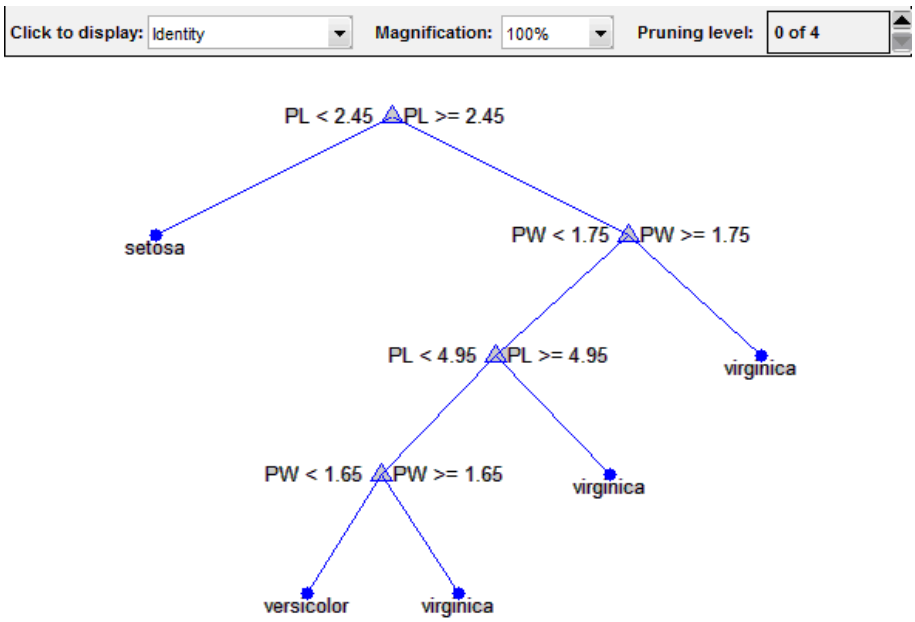


Рисунок 25 – Початкове дерево рішень для задачі класифікації ірисів в головному вікні модуля Classification tree viewer

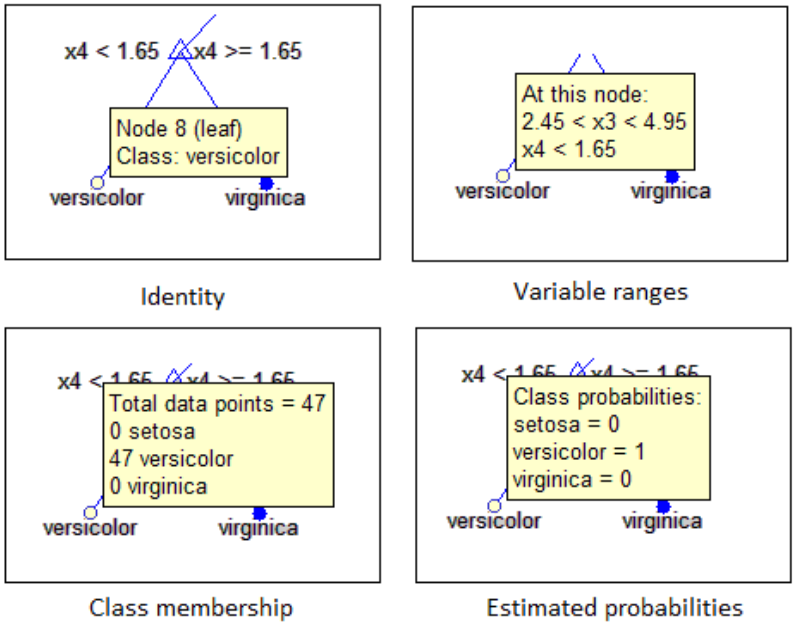


Рисунок 26 – Додаткова інформація по листу дерева за різних режимів зі списку Click to display

Підрізати дерево можна фунцією **`prune`** таким чином:

```
T2 = prune(T1, param, value),
```

де **T2** – підрізане дерево рішень;

T1 – початкове дерево рішень;

param – назва параметра;

value – значення параметра.

Допустимі значення для **param** є такими: **'level'** – підзрізання за рівнем, **'nodes'** – підзрізання за вершинами. У випадку **'level'** слід вказати на скільки рівнів зменшиться дерево рішень. Для **'nodes'** слід вказати порядкові номери проміжних вершин, які необхідно вилучити з дерева рішень. Проміжні та термінальні вершини дерева пронумеровано порівнево. Номери проміжних вершин можна визначити з графічного вікна, яке створюється за допомогою функції **view**. Список проміжних вершин дерева **T** можна отримати таким чином: **unique(T.Parent(2:end)).**

Класифікація за деревом рішень здійснюється функцією **predict**, яку можна викликати таким чином:

```
y = predict(T, X),
```

```
[y, prob, node_number, class_number] = predict (T, X),
```

де **T** – дерево рішень;

X – матриця вхідних атрибутів;

y – результати класифікації об’єктів з **X;**

prob – матриця ймовірностей належності до класів;

node_number – результати класифікації у формі номерів вершин дерева рішень;

class_number – результати класифікації у формі порядкових номерів класів рішень.

Окрім командного режиму, дерева рішень в середовищі MATLAB можна синтезувати та дослідити в спеціальному модулі Classifier Learner. Модуль запускається через верхнє меню APPS.

2.3. Метрики точності

Обирати дерево рішень за критерієм *MCR* доцільно у випадку однакової ціни помилок класифікації різних типів. На практиці

зустрічаються задачі, коли ціни помилок різних типів суттєво різняться. Наприклад, ціна помилку першого роду (пропуск цілі) в кілька разів більша за ціну помилки другого роду (хибна тривога). Інший приклад, розпізнавання купюр в терміналі. Очевидно, що помилка “50 грн → 10 грн” в 4 рази дорожча, ніж помилка “20 грн → 10 грн”. За помилок різних типів мінімізують ризик – математичне сподівання збитків від помилкової класифікації. В цьому випадку вважають відомими ціни помилок кожного типу, які зазвичай задають платіжною матрицею.

Платіжною називається наступна квадратна матриця:

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \vdots & & & \\ c_{m1} & c_{m2} & \cdots & c_{mm} \end{bmatrix}, \qquad (12)$$

де c_{ij} – ціна помилки типу $l_i \to l_j$, коли замість вірного рішення l_i під час класифікації помилково обрано рішення l_j, $i = \overline{1, m}$, $j = \overline{1, m}$.

Усі елементи головної діагоналі матриці (12) дорівнюють нулю – $c_{ii} = 1$, $i = \overline{1, m}$, що вказує на відсутність штрафу за правильну класифікацію.

Для розрахунку ризику окрім платіжної матриці слід знати і матрицю сплутувань, яка записується таким чином:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \vdots & & & \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix}, \qquad (13)$$

де p_{ij} – ймовірність події $l_i \to l_j$, $i = \overline{1, m}$, $j = \overline{1, m}$.

За відомих матриць (12) та (13) ризик розраховується таким чином:

$$R = \sum_{i=\overline{1, m}} \sum_{j=\overline{1, m}} p_{ij} \cdot c_{ij}.$$

В середовищі MATLAB для синтезу дерева рішень за відомої платіжної матриці (**`CostMatrix`**) функцію **`fitctree`** викликають таким чином:

```
T = fitctree(tr_x, tr_y, 'cost', CostMatrix).
```

Для розрахунку матриці сплутувань та її візуалізації в середовищі MATLAB є функції **confusionmat** та **confusionchart**. Функція **confusionmat** розраховує матрицю сплутувань в абсолютних значеннях – підраховує кількість випадків $l_i \to l_j$, $i = \overline{1, m}$, $j = \overline{1, m}$. Проілюструємо використання цих функцій такою програмою, що синтезує дерево рішень для задачі про іриси:

```
load fisheriris;
T = fitctree(meas, species);
y_pred=predict(T,meas);
confusion=confusionmat(species, y_pred)
confusionchart(confusion);
%Розрахуємо ймовірності сплутувань:
conf_prob= confusion./sum(sum(confusion))
```

В результаті виконання програми отримаємо матриці сплутувань для кількості подій та їх ймовірностей:

```
confusion =
    50     0     0
     0    47     3
     0     0    50
conf_prob =
    0.3333         0         0
         0    0.3133    0.0200
         0         0    0.3333
```

Матриця сплутувань в абсолютних значеннях, що візуалізована командою **confusionchart**, показана на рис. 27.

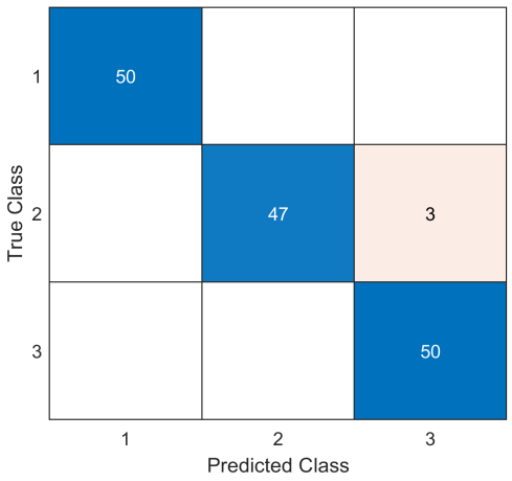


Рисунок 27 – Матриця сплутувань для дерева рішень з рис. 25

Для багатьох задач відомо, що помилки різних типів мають різну ціну. Але співвідношення між цінами помилок не так просто встановити, як для задачі розпізнання купюр. Наприклад, під час оцінювання знань студентів за шкалою {“2”, “3”, “4”, “5”} зустрічаються 12 типів помилок. Ціна помилок типу $5 \to 2$ значно вища за помилки типу $3 \to 4$ чи $4 \to 3$, але словосполучення “значно вище” у платіжну матрицю так просто не занести. За таких умов в постановку задачі навчання класифікатора можна ввести обмеження на допустимі рівні помилок найбільш небезпечних типів. У випадку лише двох типів помилок мінімізують частоту помилок другого роду при обмежені на рівень помилок першого роду.

Часто зустрічаються незбалансовані задачі класифікації, коли частоти різних класів суттєво різні. Наприклад, в kaggle-задачі “Toxic Comment Classification Challenge” навчальна вибірка складається із 159751 текстового коментаря, з них 143526 (89.8%) є нейтральними. За таких умов точність тривіального класифікатора, який завжди відносить коментар до нейтрального, становитиме 0.898. Очевидно, що тривіальний класифікатор нікому не потрібен – він не виявляє жодного нахабного коментаря. У таких випадках доцільно використовувати збалансовану точність, яка розраховується як середнє значення безпомилковості за кожним класом. На основі матриці сплутувань (13) збалансована точність розраховується як середнє значення елементів головної діагоналі:

$$A_{balanced} = \frac{1}{m} \sum_{i=\overline{1,m}} p_{ii} .$$

Окрім простоти та змістовної інтерпретації, збалансована точність придатна для оцінювання класифікаторів у випадку багатьох класів. Це додаткова перевага цієї метрики у порівнянні з такими як f1-критерій (f1-score) та коефіцієнт кореляції Метьюса (Matthews correlation coefficient).

2.4. Ансамблі

Особливість дерев рішень полягає в тому, що границі класів у просторі вхідних атрибутів є прямокутними. Тому для складних задач з іншими границями класів застосування дерев рішень недоцільне. Одним із

шляхів підвищення достовірності є використання не одного класифікатора, а ансамблю (колективу) класифікаторів, за яким можна реалізувати довільні границі розділу класів. Є багато підходів до колективного прийняття рішень, наприклад, створення моделі, за якою обирається класифікатор для кожної області факторного простору. Тобто для кожного класифікатора визначається область його компетенції. Інший підхід полягає в класифікації одного і того ж об'єкту одночасно кількома класифікаторами. Кожен класифікатор видає своє рішення, які потім агрегують. Найпростішим способом агрегування є вибір рішення, за який проголосувало найбільше класифікаторів. Згідно до теореми Кондорсе про присяжних, у випадку двох класів рішень ймовірність безпомилковості зростає зі збільшенням кількості учасників голосування при виконанні двох умов. Перша умова – ймовірність помилки кожного окремого класифікатора має бути більшою за 0.5. Друга умова – диверсифікація класифікаторів - помилки різних класифікаторів мають бути незалежними. Відповідно, класифікатори в ансамблі можуть бути і слабкими (*weak*), тобто з великою частотою помилок. Тому, просте дерево рішень є хорошим кандидатом в члени ансамблю.

Одним із найбільш ефективних методів синтезу ансамблю класифікаторів є бустинг дерев рішень. Бустинг (boosting) – це метод синтезу високоточного класифікатора з низькоточних простих класифікаторів. За одну ітерацію бустингу в ансамбль додається 1 класифікатор, який забезпечує найбільше зростання безпомилковості. При цьому попередні класифікатори не видаляються. Таким чином, бустинг має ознаки жадібного алгоритму. Дерево рішень на кожній ітерації синтезується з деякої підмножини навчальної вибірки за типовими алгоритмами. Ключовою особливістю бустингу є адаптивність формування навчальної підвибірки на кожній ітерації. Підвибірка формується випадково, але так, щоб більше шансів потрапити в неї мали проблемні об'єкти. Під проблемними розуміються об'єкти, які частіше за інші хибно класифікуються деревами рішень з поточного ансамблю. Таким чином, кожне нове дерево ансамблю намагається виправити помилки попередніх. Бустинг часто реалізують алгоритмами AdaBoost та arc-x4.

Під час вирішення практичних задач виявлено, що ефект перенавчання ансамблю класифікаторів, синтезованих за бустингом дерев

рішень, спостерігається рідко. Тобто, зі збільшенням кількості дерев в ансамблі зменшується кількість помилкових рішень як для навчальної, так і для тестової вибірок. Збільшуючи розмір ансамблю можна досягти нульової частоти помилок на навчальній вибірці. Якщо продовжити додавати в ансамбль нові дерева, тоді ще протягом кількох ітерацій частота помилок на тестовій вибірці може спадати. Але тут слід пам'ятати, що безпомилковість класифікатора – це не єдиний критерій якості моделі прийняття рішень. При збільшенні ансамблю погіршуються інші критерії – зростає обчислювальна складність та втрачається наочність. Крім того, в модель додаються нові вхідні змінні, яких не було в попередніх деревах рішень. Відповідно, під час експлуатації класифікатора зростають витрати на отримання початкових даних через встановлення нових вимірювальних приладів в задачах технічної діагностики чи проведення додаткових лабораторних обстежень пацієнтів в задачах медичної діагностики.

2.5. Функції для дерев рішень в системі MATLAB

Для роботи з деревом рішень доцільно використовувати такі функції:

`fitctree` – синтез дерева рішень з даних;

`view` – візуалізація дерева рішень**;**

`predict` – класифікація за деревом рішень**;**

`prune` – підрізання дерева рішень**;**

`confusionmat` – підрахунок матриці сплутувань;

`confusionchart` – підрахунок матриці сплутувань;

`histc` - підрахунок частот значень категоріальної змінної

`cell2mat` – перетворення списку в масив;

`str2num` – перетворення даних з символьного формату у числові;

`max` – значення та порядковий номер максимального елементу масиву;

`min` – значення та порядковий номер мінімального елементу масиву;

`setdiff` – різниця множин**;**

`find` – знаходження номерів елементів масиву, які задовольняють деяку умову;

`unique` – вилучення з масиву дублюючих елементів;

`gplotmatrix` – побудова двофакторних розподілів класів.

2.6. Питання для самоконтролю та розвитку

1. В чому принципова відмінність числової та категоріальної шкал?
2. Яка складність жадібного алгоритму синтезу дерева рішень?
3. Як змінюється складність перебору між сусідніми ітераціями алгоритми синтезу дерева рішень?
4. Як визначити платіжну матрицю для задачі класифікації позичальників під час прийняття рішень щодо кредитування?
5. Чи потрібно, щоб помилки класифікації за окремими деревами ансамблю були корельованими?
6. Чи можна в ансамбль класифікаторів включати дерева рішень, які синтезовано за різними алгоритмами?
7. Наведіть змістовні приклади помилок першого та другого родів для реальних задач класифікації?
8. Скільки існує різних типів помилок для задачі класифікації з трьома класами?
9. Які труднощі визначення платіжної матриці для реальних задач прийняття рішень?
10. Як оцінити середні витрати для здійснення класифікації за деревом рішень, якщо відома вартість вимірювання кожної ознаки?
11. Чи можна точність дерева рішень оцінювати за логарифмом втрат (log-loss)?
12. Чи можна точність дерева рішень оцінювати площею під кривою помилок (AUC-ROC)?

3. КЛАСТЕРИЗАЦІЯ

3.1. Ази

Кластеризація - це об'єднання об'єктів в групи (кластери) на основі схожості ознак для об'єктів однієї групи і відмінностей між об'єктами з різних груп, що відповідає навчанню без учителя. Кластеризація включає в себе наступні етапи: виділення ознак, визначення метрики, розбиття об'єктів на групи та представлення результатів. Для початку необхідно вибрати ознаки, які характеризують об'єкти. Ними можуть бути кількісні ознаки – координати, висота, довжина тощо або якісні ознаки – колір, статус, військове звання, тощо. Далі варто спробувати зменшити розмірність простору ознак, тобто виділити найбільш важливі атрибути об'єктів. Зменшення розмірності прискорює процес кластеризації і в ряді випадків дозволяє візуально оцінювати її результати. Вихідною інформацією для кластеризації є матриця спостережень:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & & & \\ x_{M1} & x_{M2} & \dots & x_{Mn} \end{bmatrix},$$

в кожному рядку якої записано значення n атрибутів одного із M об'єктів. Кластеризація полягає в розбитті об'єктів з $\mathbf{X}$ на кілька підмножин (кластерів), в яких об'єкти між собою більш схожі, ніж з об'єктами з інших кластерів. У метричному просторі «схожість» зазвичай визначають через відстань. Відстань може розраховуватися як між об'єктами – рядками матриці $\mathbf{X}$, так і від цих об'єктів до прототипів кластерів. Найчастіше координати прототипів заздалегідь невідомі, їх знаходять одночасно з розбивкою даних на кластера.

Існує багато методів кластеризації. Їх класифікують за тим, чи визначено кількість кластерів заздалегідь, до початку кластеризації чи ні. В

останньому випадку кількість кластерів знаходять за розподілами початкових даних під час виконання алгоритму. До таких методів відносять метод дендрограм та субтрактивну кластеризацію. Серед методів із заданою кількістю кластерів часто використовують метод *k*-середніх (*k*-means). Цей метод також називають *c*-середніх. Саме таку назву будемо використовувати в подальшому.

Кластеризація може бути ієрархічною або планарною. Планарна кластеризації здійснюється на одному рівні – “об’єкти – кластера”, тобто кожний об’єкт приписують до якогось кластера. За ієрархічної кластеризації рівнів може бути кілька. На найнижчому рівні об’єкти розподілять за кластерами першого рівня. На другому рівні об’єднуються деякі кластера. На третьому рівні об’єднуються між собою кластера другого рівня, або кластера першого та другого рівнів. На будь-якому рівні об’єднуватися можуть не лише кластера, але і до кластерів додаватися окремі об’єкти. Найбільш відомим методом ієрархічної кластеризації є метод дендрограм. Приклад ієрархічної кластеризації 30 документів за цим методом наведено на рис. 28.

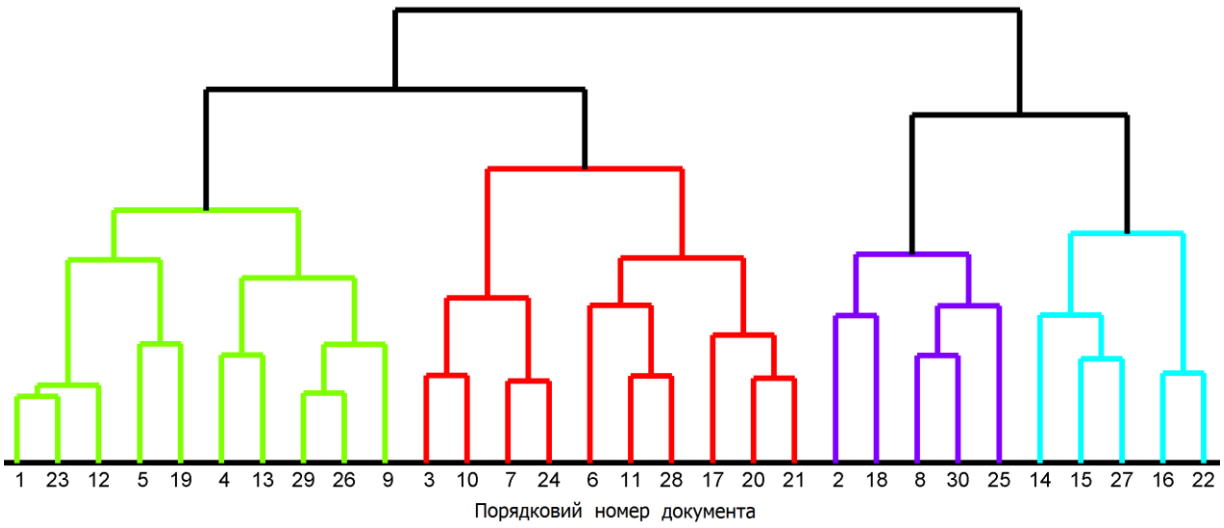


Рисунок 28 – Приклад ієрархічної кластеризації

Методи кластеризації поділяють на чіткі і нечіткі. Чіткі методи кластеризації розбивають множину об'єктів **X** на декілька підмножин з порожнім попарним перетином. При цьому будь-який об'єкт з **X** належить тільки одному кластеру. Нечіткі методи кластеризації дозволяють одному і тому ж об'єкту одночасно належати кільком, або навіть усім кластерам, але

з різними ступенями. Нечітка кластеризація в багатьох ситуаціях більш «природна», ніж чітка, наприклад, для об'єктів, розташованих на межі кластерів. Проілюструємо цю тезу на «метелику» – добре відомому в теорії кластеризації прикладі. «Метелик» складається із 15 об'єктів, двовимірне зображення яких нагадує однойменну комаху (рис. 29а). За чіткої кластеризації (рис. 29б і в) виходять два кластери із 7 і 8 об'єктів. На рисунку об'єкти першого кластера позначені трикутниками, а другого – квадратами. Симетричний «метелик» за чіткої кластеризації розбивається на два несиметричних кластера. За нечіткої кластеризації (рис. 29г) проблемний восьмий об'єкт, розташований в центрі «метелики», одночасно належить двом симетричним кластерам з одним і тим же ступенем. На цьому рисунку розмір маркерів пропорційний ступеню належності об'єкта кластеру. Кластеризація здійснена за алгоритмом нечітких *c*-середніх (fuzzy *c*-means).

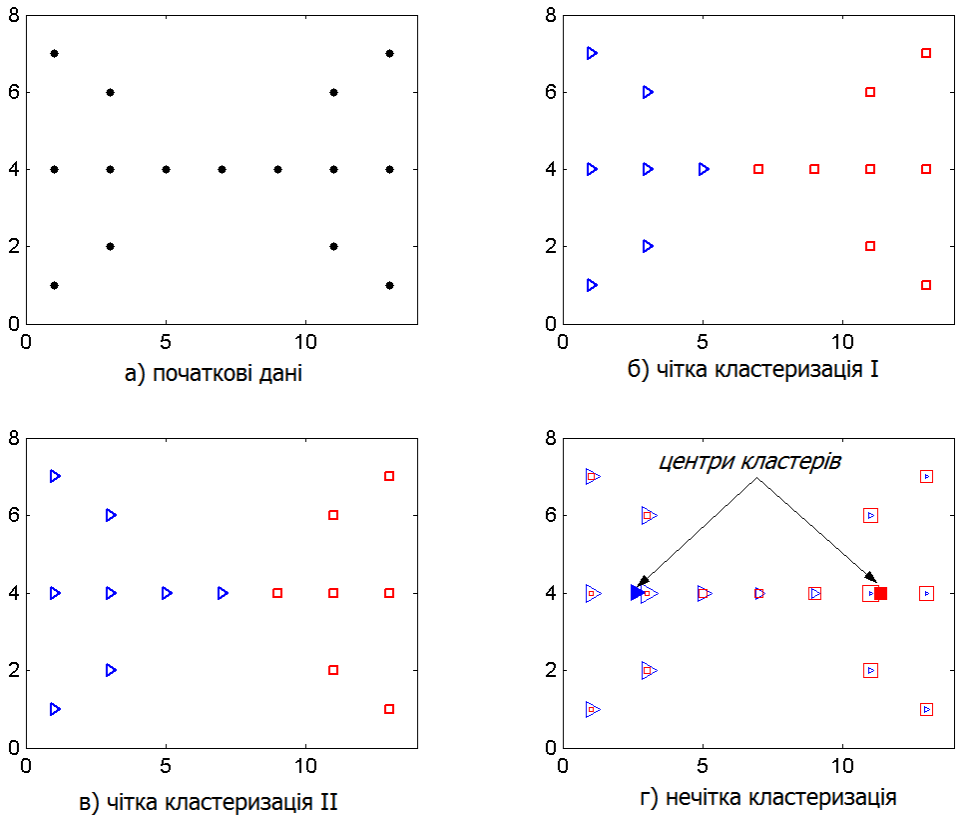


Рисунок 29 – Кластеризація «метелика»

Подальший виклад матеріалу стосуватиметься методів планарної кластеризації: алгоритму *c*-середніх, алгоритму нечітких *c*-середніх та субтрактивного алгоритму гірської кластеризації.

3.2. Кластеризація за алгоритмом *c*-середніх

В результаті кластеризації за алгоритмом *c*-середніх початкова множина об'єктів **X** розбивається на підмножини $A_1, A_2, ..., A_c$, з наступними властивостями:

$$\bigcup_{i=\overline{1,c}} A_i = \mathbf{X}, \tag{14}$$

$$A_i \cap A_j = \varnothing, \quad i, j = \overline{1,c},\ i \neq j, \tag{15}$$

$$\varnothing \subset A_i \subset \mathbf{X}, \quad i = \overline{1,c}. \tag{16}$$

Умова (14) вказує, що всі об'єкти повинні бути розподілені за кластерами. При цьому, за умовою (15) кожен об'єкт повинен належати тільки одному кластеру. За умовою (16) жоден із кластерів не може бути порожнім або містити усі об'єкти. Кількість кластерів $c \in \{2, 3, ..., M-1\}$ задається до початку роботи алгоритму.

Задачу кластеризації зручно формулювати використовуючи характеристичну функцію. Характеристична функція приймає значення 0, якщо елемент не належить кластеру, та значення 1, якщо елемент належить кластеру. Використовуючи характеристичну функцію кластера опишемо такою матрицею розбиття:

$$\mathbf{U} = [\varphi_{ki}],\ \varphi_{ki} \in \{0,1\},\ \ k = \overline{1,M},\ \ i = \overline{1,c},$$

де *k*-й рядок матриці **U** вказує на належність об'єкта $X_k = (x_{k1}, x_{k2}, ..., x_{kn})$ до кластерів $A_1, A_2, ..., A_c$.

Матриця **U** має такі властивості:

$$\sum_{i=\overline{1,c}} \varphi_{ki} = 1, \quad k = \overline{1,M}, \tag{17}$$

$$0 < \sum_{k=\overline{1,M}} \varphi_{ki} < M, \quad i = \overline{1,c}. \tag{18}$$

Для оцінювання якості розбиття використовується критерій розкиду, зазвичай у формі суми відстаней від об'єктів до центру свого кластера. За евклідового простору цей критерій розраховується так:

$$\sum_{i=1,\,c}\ \sum_{X_k \in A_i} \|V_i - X_k\|^2 . \qquad (19)$$

де $A_i = \{X_p :\ \varphi_{pi} = 1,\quad p = \overline{1, M}\}$ – i-й кластер;

$V_i = \dfrac{1}{|A_i|} \sum_{X_k \in A_i} X_k$ – центр i-го кластера;

$|A_i|$ – кількість об'єктів кластера A_i.

Кластеризацію об'єктів **X** можна сформулювати як таку задачу оптимізації: знайти матрицю **U**, яка мінімізує критерій (19). Дискретний характер чіткого розбиття обумовлює негладкість цільової функції, що ускладнює знаходження оптимальної кластеризації.

В середовищі MATLAB кластеризація за методом *c*-середніх реалізована функцією **kmeans**. Обов’язковими аргументами функції є матриця спостережень **X** та кількість кластерів. Функція повертає вектор, що містить номер кластера кожного об’єкта. Проілюструємо роботу цієї функції на прикладі кластеризації ірисів:

```
load fisheriris;
clusters=kmeans(meas, 3, 'Display', 'iter');
iris=mode([clusters(1:50) clusters(51:100)
clusters(101:150)]);
disp('Кількість помилок для Setosa:')
sum([clusters(1:50)~=iris(1)])
disp('Кількість помилок для Versicolor:')
sum([clusters(51:100)~=iris(2)])
disp('Кількість помилок для Virginica:')
sum([clusters(101:150)~=iris(3)])
```

В результаті виконання програми отримаємо таку інформацію:

```
iter       phase          num           sum
     1         1          150       79.2971
     2         1            2       78.8514
Best total sum of distances = 78.8514
```

```
Кількість помилок для Setosa:
ans =
     0

Кількість помилок для Versicolor:
ans =
     2

Кількість помилок для Virginica:
ans =
    14
```

Перший блок видачі, який активовано аргументами **'Display', 'iter'**, є покроковим протоколом кластеризації. Остання колонка видачі містить інформацію про значення критерію оптимальності. Другий блок видачі – кількість ірисів кожного виду, яких призначено до чужих кластерів. Найбільша кількість помилок – 14 для Iris Virginica.

Кластеризацію можна проводити не лише за евклідовою метрикою, але і за іншими. Для кластеризації ірисів за манхетенською метрикою (сумою абсолютних відхилень) **kmeans** викличемо таким чином:

```
clusters=kmeans(meas, 3, 'Distance', 'Cityblock');
```

В результаті отримуємо:

```
Кількість помилок для Setosa:
ans =
     0

Кількість помилок для Versicolor:
ans =
     2

Кількість помилок для Virginica:
ans =
    15
```

У базовому алгоритмі *c*-середніх відстань між об'єктом $X = (x_1, x_2, ..., x_n)$ та центром кластера $V = (v_1, v_2, ..., v_n)$ розраховується через евклідову норму: $D^2 = \|X - V\|^2$. У кластерному аналізі застосовуються і інші норми, серед яких, окрім згадуваної манхетенської, часто використовується діагональна норма і норма Махалонобіса. У

загальному вигляді норму можна задати через симетричну додатно визначену матрицю **B** розміром $n \times n$:

$$\|X - V\|_{\mathbf{B}}^2 = (X - V) \cdot \mathbf{B} \cdot (X - V)^{\mathsf{T}},$$

де T - операція транспонування.

Для евклідової норми матриця **B** являє собою одиничну матрицю:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}.$$

За евклідової норми кластера виділяються у вигляді гіперсфер.

Для діагональної норми матриця **B** задається так:

$$\mathbf{B} = \begin{bmatrix} \omega_1 & 0 & \dots & 0 \\ 0 & \omega_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \omega_n \end{bmatrix}.$$

Елементи головної діагоналі матриці інтерпретуються як ваги координат. За діагональної норми виділяються кластера у вигляді гіпереліпсоідов, орієнтованих уздовж координатних осей.

Для норми Махалонобіса матриця **B** розраховується через коваріаційну матрицю від **X**:

$$\mathbf{B} = \mathbf{R}^{-1},$$

де $\mathbf{R} = \frac{1}{M} \sum_{k=\overline{1, M}} (X_k - \overline{X})^{\mathsf{T}} \cdot (X_k - \overline{X})$ - коваріаційна матриця;

$\overline{X} = \frac{1}{M} \sum_{k=\overline{1, M}} X_k$ - вектор середніх значень даних.

За норми Махаланобіса кластери мають вигляд гіперелліпсоідов з довільною орієнтацією. Приклади изоліній за різних норм показані на рис. 30.

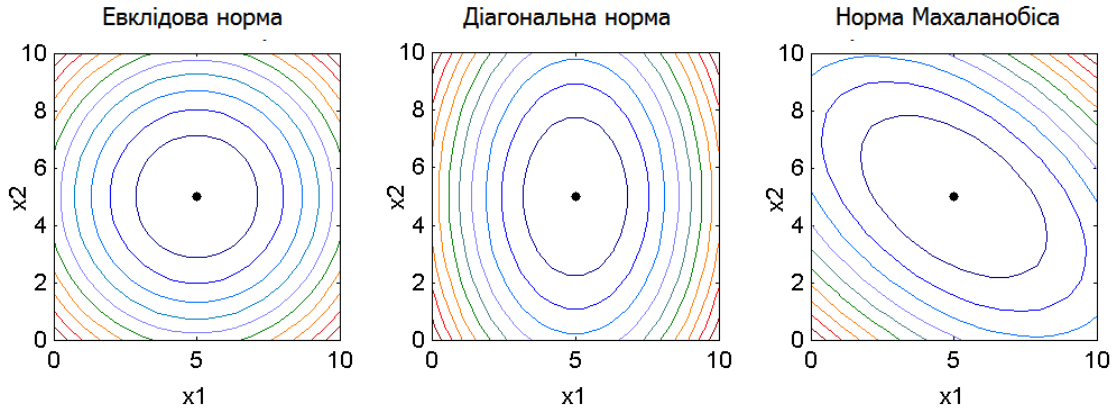


Рисунок 30 – Ізолінії відстаней за різних норм

3.3. Кластеризація за алгоритмом нечітких *c*-середніх

Нечіткі кластери опишемо матрицею нечіткого розбиття, в якій *k*-й рядок містить ступені належності об'єкта $X_k = (x_{k1}, x_{k2}, ..., x_{kn})$ кластерам $A_1, A_2, ..., A_c$:

$$\mathbf{F} = [\mu_{ki}], \quad \mu_{ki} \in [0,1], \quad k = \overline{1,M}, \quad i = \overline{1,c}, \tag{20}$$

Єдиною відмінністю матриць **F** та **U** є те, що за нечіткого розбиття ступінь належності об'єкта кластеру приймає значення з інтервалу [0, 1], а за чіткому – з двоелементної множини {0, 1}. Аналогічні (17) – (18) умови для матриці нечіткого розбиття записуються так:

$$\sum_{i=\overline{1,c}} \mu_{ki} = 1, \quad k = \overline{1,M}, \tag{21}$$

$$0 < \sum_{k=\overline{1,M}} \mu_{ki} < M, \quad i = \overline{1,c}, \tag{22}$$

Нечітке розбиття дозволяє просто вирішити проблему об'єктів, розташованих на межі двох кластерів – їм призначають однакові ступеня належності, наприклад, по 0.5. Недолік нечіткого розбиття проявляється під час роботи з об'єктами, віддаленими від центрів всіх кластерів. Віддалені об'єкти мають мало спільного з будь-яким кластером, тому інтуїтивно хочеться призначити їм малі ступеня належності. Однак, за

умовою (21) сума ступенів належності дорівнює 1, як у віддалених, так і у близьких до центрів кластерів об'єктів.

Для оцінювання якості нечіткого розбиття використовується такий критерій:

$$\sum_{i=\overline{1,c}} \sum_{k=\overline{1,M}} (\mu_{ki})^m \cdot \|V_i - X_k\|^2, \tag{23}$$

де $V_i = \dfrac{\sum_{k=\overline{1,M}} (\mu_{ki})^m \cdot X_k}{\sum_{k=\overline{1,M}} (\mu_{ki})^m}$ – центри нечітких кластерів;

$m \in [1, \infty)$ – коефіцієнт щільності кластерів; чим більше m, тим розмитішим виходить нечітке розбиття.

Найбільш популярний метод мінімізації критерію (23) – алгоритм нечітких с-середніх – FCM. Він базується на методі невизначених множників Лагранжа. Алгоритм знаходить локальний оптимум, тому виконання його з різних початкових точок може призвести до різних результатів.

Алгоритм нечітких с-середніх складається з таких кроків.

Крок 1. Встановити параметри алгоритму: c – кількість кластерів; m - коефіцієнт щільності кластерів; ε – параметр зупинки алгоритму.

Крок 2. Випадковим чином згенерувати матрицю нечіткого розбиття **F**, що задовольняє умови (21) – (22).

Крок 3. Розрахувати центри кластерів за формулою:

$$V_i = \frac{\sum_{k=\overline{1,M}} (\mu_{ki})^m \cdot X_k}{\sum_{k=\overline{1,M}} (\mu_{ki})^m}, \quad i = \overline{1,c}.$$

Крок 4. Розрахувати відстані між об'єктами з **X** та центрами кластерів:

$$D_{ki} = \sqrt{\|X_k - V_i\|^2}, \; k = \overline{1,M}, \; i = \overline{1,c}.$$

Крок 5. Перерахувати елементи матриці нечіткого розбиття для всіх $k = \overline{1,M}$ та $i = \overline{1,c}$:

якщо $D_{ki} > 0$, тоді $\mu_{ki} = \dfrac{1}{\left(D_{ik}^2 \cdot \sum\limits_{j=\overline{1,c}} \dfrac{1}{D_{jk}^2} \right)^{1/(m-1)}}$;

якщо $D_{ki} = 0$, тоді $\mu_{kj} = \begin{cases} 1, \text{ якшо } j = i \\ 0, \text{ якшо } j \neq i \end{cases}$, $j = \overline{1, c}$.

Крок 6. Перевірити умову $\left\| \mathbf{F} - \mathbf{F}^* \right\|^2 < \varepsilon$, де $\mathbf{F}^*$ – матриця нечіткого розбиття на попередній ітерації алгоритму. Якщо «так», то перейти до кроку 7, інакше – до кроку 3.

Крок 7. Кінець.

В середовищі MATLAB кластеризація за методом нечітких *c*-середніх реалізована функцією **`fcm`**. Обов'язковими аргументами функції є матриця спостережень **X** та кількість кластерів. Функція повертає матрицю належностей кожного об'єкта до кластерів, а також центри нечітких кластерів. Проілюструємо роботу цієї функції на прикладі кластеризації метелика:

Приклад. Початкові дані для кластеризації задано в табл. 4. Графічне зображення цих даних наведено на рис. 29а. Кластеризуємо ці дані за таких параметрах алгоритму нечітких с-середніх: *c*=2 та *m*=2. Після 8-ми ітерацій одержимо нечітке розбиття з табл. 5. Значення критерію (23) для цього нечіткого розбиття дорівнює 82.94.

Таблиця 4 – Початкові дані для кластеризації

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_1	1	1	1	3	3	3	5	7	9	11	11	11	13	13	13
x_2	1	4	7	2	4	6	4	4	4	2	4	6	1	4	7

Таблиця 5 – Результати кластеризації

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
μ_{k1}	0.91	0.98	0.91	0.95	1	0.95	0.86	0.5	0.12	0.05	0	0.05	0.09	0.02	0.09
μ_{k2}	0.09	0.02	0.09	0.05	0	0.05	0.12	0.5	0.88	0.95	1	0.95	0.91	0.98	0.91

Результати нечіткої кластеризації показані на рис. 29г (розмір маркерів пропорційний ступеню належності об'єкта кластеру). Тривимірні зображення нечітких кластерів наведені на рис. 31.

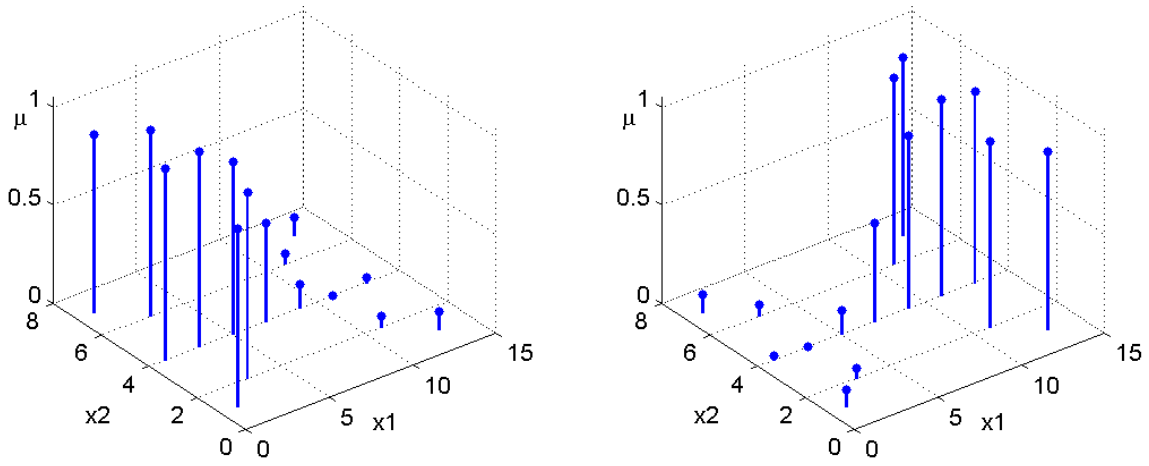


Рисунок 31 – Тривимірне зображення нечітких кластерів

В середовищі MATLAB є 2 демо-програми з кластеризації за методом нечітких *c*-середніх: fcmdemo та irisfcm. Програма fcmdemo ілюструє застосування алгоритму нечітких *c*-середніх для задач кластеризації. Вона виводить на екран інтерактивне вікно (рис. 32) з можливістю вибрати набір даних для кластеризації, встановити параметри алгоритму нечітких *c*-середніх, вивести результати кластеризації, в тому числі і графіки функцій належності нечітких кластерів. Значення цільової функції на кожній ітерації алгоритму кластеризації виводяться в робочу область MATLAB.

Графічне вікно fcmdemo містить 7 верхніх типових меню, область візуалізації, меню вибору даних, меню установки параметрів алгоритму нечітких *c*-середніх, кнопку запуску кластеризації Start. Меню вибору даних розташоване в правому верхньому куті графічного вікна. Користувач може вибрати один з п'яти демо-наборів даних Data Set 1 – Data Set 5 або завантажити свої дані. Демо-набори генеруються програмою fcmdemo при кожному завантаженні даних. Для завантаження своїх даних необхідно вибрати опцію Custom ... та вказати відповідний файл даних. Дані повинні бути записані в файлі порядково, тобто кожен об'єкт необхідно описати одним рядком, що містить значення двох ознак.

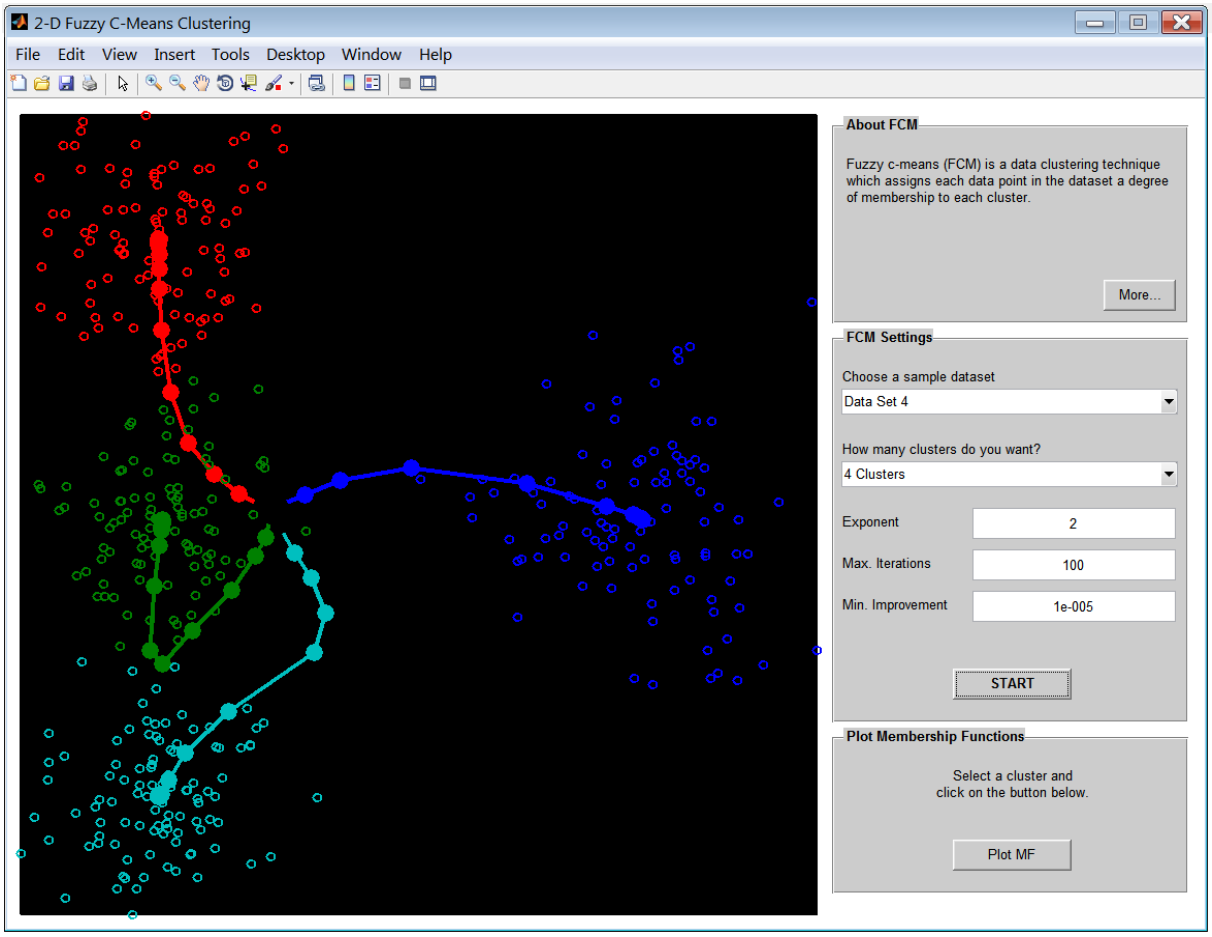


Рисунок 32 – Вікно демонстраційної програми fcmdemo

У центрі графічного вікна розташована область візуалізації, в якій виводяться об'єкти кластеризації та знайдені центри кластерів. Маркери у вигляді кіл відповідають об'єктам кластеризації, а маркери у вигляді дисків –центрам кластерів. Кластера виділяються різними кольорами. Спочатку центри всіх кластерів розташовуються в середині області візуалізації. Потім, під час виконання алгоритму центри нечітких кластерів покроково зміщуються на свої місця. Траєкторії центрів кластерів зображуються суцільними лініями. Після кожної ітерації алгоритму кольору об'єктів змінюються, відповідно до їх належності кластерам.

Для управління візуалізацією використовуються такі кнопки: Label Data – дозвіл / заборона виділення кольором належності об'єктів кластерам; Clear Traj. – очистка попередніх траєкторій руху центрів кластерів; MF Plot – виведення функції належності центрів кластерів. Для виведення графіка функції належності необхідно обрати центр кластера і натиснути кнопку MF Plot. Приклад функції належності центру кластера показана на рис. 33.

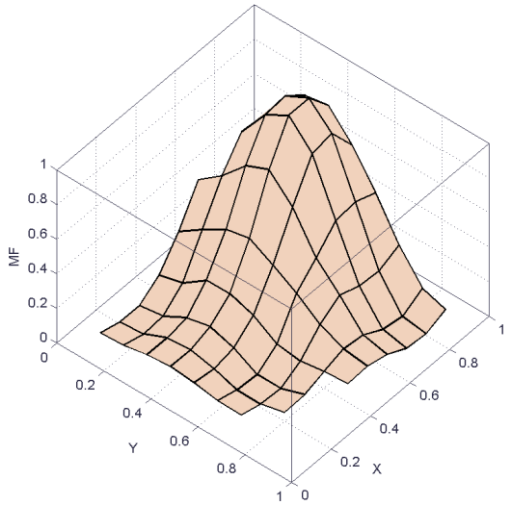


Рисунок 33 – Функція належності нечіткого кластера в програмі fcmdemo

В програмі fcmdemo користувач може вибрати через меню кількість кластерів (від 2 до 9), а також встановити значення таких параметрів алгоритму нечітких *c*-середніх: коефіцієнт щільності кластерів (поле Expo); максимальну кількість ітерацій алгоритму (поле Iterat.); мінімальне допустиме зменшення цільової функції за одну ітерацію алгоритму (поле Improv.).

Демонстраційна програма irisfcm ілюструє застосування алгоритму нечітких *c*-середніх для кластеризації ірисів. На рис. 34 наведені двовимірні розподіли ірисів по класах. Іриси Iris Setosa зображені трикутниками, Iris Versicolor – точками і Iris Virginica – хрестиками. Центри кластерів позначено цифрами 1, 2 та 3. Кластеризація здійснюється за алгоритмом нечітких c-середніх з такими параметрами: коефіцієнт щільності кластерів – 2; кількість кластерів – 3; максимальна кількість ітерацій алгоритму – 100; мінімальне зменшення цільової функції за одну ітерацію – 0.000001. Окрім центрів кластерів демо-програма irisfcm виводить і криву навчання у вигляді графіка цільової функції на ітераціях алгоритму (рис. 35).

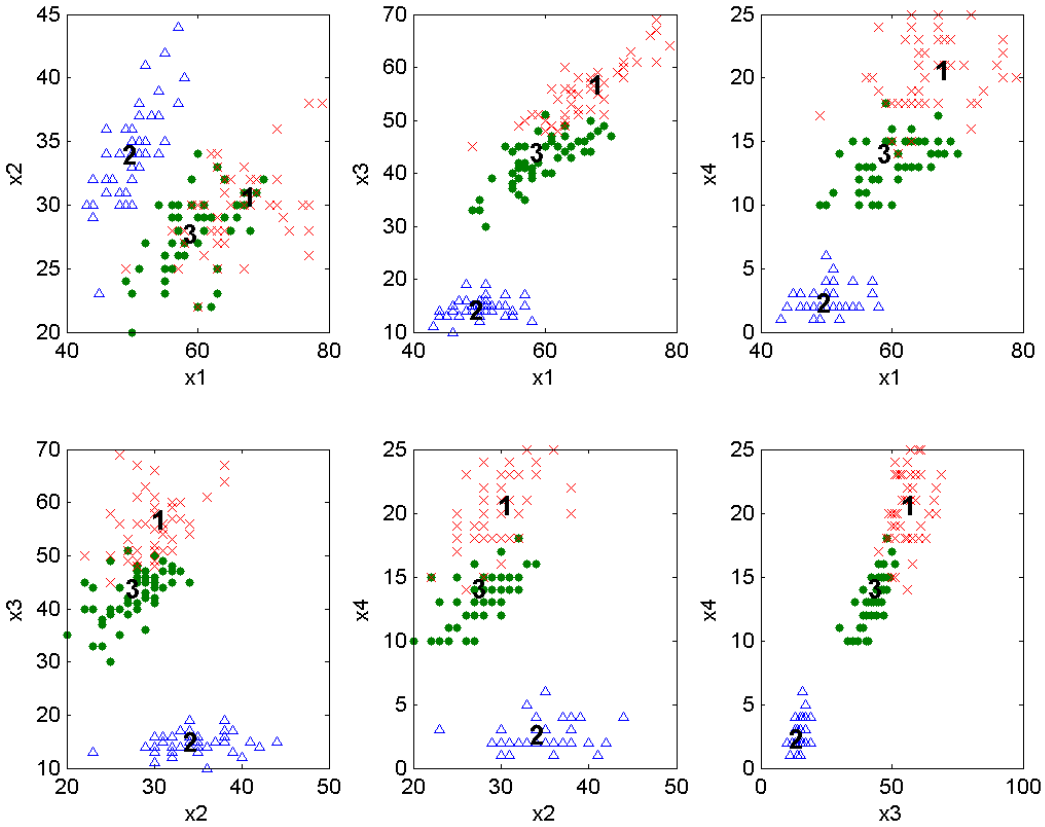


Рисунок 34 – Центри нечітких кластерів ірисів в програмі irisfcm

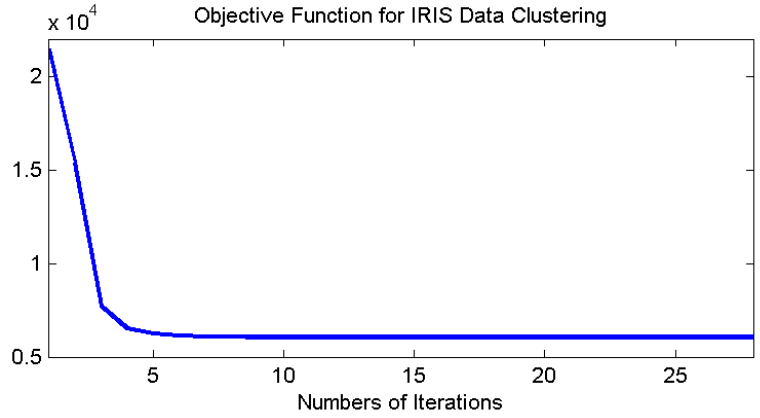


Рисунок 35 – Динаміка якості кластеризації ірисів

В алгоритмі нечітких *c*-середніх найважливішим параметром є кількість кластерів. Правильно вибрати кількість кластерів для реальних завдань без будь-якої апріорної інформації про структури в даних досить складно. Існує два формальних підходи до вибору числа кластерів. Перший підхід заснований на критерії компактності і віддаленості отриманих кластерів. Логічно припустити, що за вірного вибору кількості кластерів дані будуть розбиті на компактні і добре віддалені один від

одного групи. Існує кілька критеріїв оцінювання компактності кластерів, однак питання про те, як формально і достовірно визначити правильність вибору кількості кластерів для довільного набору даних все ще залишається відкритим. Для алгоритму нечітких *c*-середніх як критерій компактності кластерів можна використовувати коефіцієнт Ксі–Бені (Xie–Beni):

$$\chi = \frac{\sum_{i=1,\,c}\ \sum_{k=1,\,M} (\mu_{ik})^m \cdot \|X_k - V_i\|^2}{M \cdot \min_{i \neq j} \left(\|X_k - V_i\|^2 \right)}.$$

За другим підходом розпочинають за великої кількості кластерів, а потім послідовно об'єднують схожі суміжні кластера. При цьому застосовують різні формальні критерії схожості кластерів.

Важливим чинником успішної кластеризації є вибір релевантної метрики. Кожен тип метрики продукує кластера певної форми. За евклідової метрики форма кластерів близька до сферичної. На рис. 36 наведено приклад кластеризації за методом нечітких *c*-середніх з використанням евклідової метрики: ліворуч зображені початкові об'єкти; праворуч показані результати нечіткої кластеризації. Центри нечітких кластерів позначені символами '+'. Вісім ізоліній функцій належності нечітких кластерів побудовані для значень 0.67, 0.71, 0.75, 0.79, 0.83, 0.87, 0.91 та 0.95.

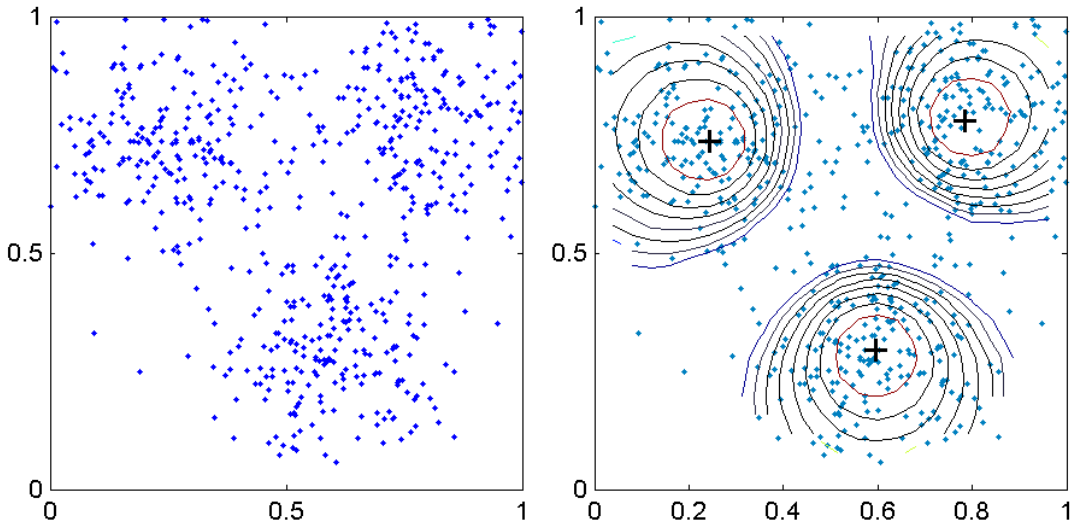


Рисунок 36 – Нечітка кластеризація за евклідової метрики

Для деяких наборів даних можна на око виділити скупчення об'єктів у вигляді різних геометричних фігур: сфер, еліпсоїдів різної орієнтації, ланцюжків тощо. В результаті кластеризації за алгоритмом з фіксованою метрикою форма усіх кластерів виходить однаковою. Алгоритми кластеризації ніби нав'язують даними невластиву їм структуру, що призводить не тільки до неоптимальних, але іноді і до принципово неправильних результатів. Для усунення цього недоліку запропоновано кілька методів, серед яких виділимо алгоритм Густавсона-Кеселя (Gustafson-Kessel).

Алгоритм Густавсона-Кеселя використовує адаптивну норму для кожного кластера, тобто для кожного *i*-го кластера існує своя норм-породжуюча матриця $\mathbf{B}_i$. За цим алгоритмом виділяються кластера різної геометричної форми, так як оптимізуються не тільки координати центрів кластерів і матриця нечіткого розбиття, але також метрики. Критерій оптимальності (28) лінійний відносно $\mathbf{B}_i$, тому для отримання ненульових рішень, вводяться деякі обмеження на норм-породжуючі матриці. В алгоритмі Густавсона–Кеселя це таке обмеження на значення визначника норм-породжуючих матриць:

$$\left|\mathbf{B}_i\right| = \beta_i, \ \beta_i > 0, \ i = \overline{1, c} \ .$$

Оптимальне рішення знаходять за допомогою методу невизначених множників Лагранжа. Алгоритм Густавсона-Кеселя має значно більшу обчислювальну трудомісткість в порівнянні з алгоритмом нечітких *c*-середніх.

3.4. Кластеризація за гірським методом

Гірський метод кластеризації не потребує завдання кількості кластерів. Кількість кластерів визначається за розподілом даних. Алгоритм відноситься до субтрактивної кластеризації (від англ. subtractive). Спочатку формується перший кластер і з вибірки вилучаються усі об’єкти першого кластера. Далі за новим розподілом даних формується другий кластер і з вибірки вилучаються усі його об’єкти. Так продовжується до тих пір, поки усі об’єкти не будуть розподілені за кластерами.

На першому кроці гірської кластеризації визначають точки, які можуть бути центрами кластерів. На другому кроці для кожної такої точки розраховується значення потенціалу, що показує можливість формування кластера в її околі. Чим густіше розташовані об'єкти в околі прототипу центра кластера, тим вище значення його потенціалу. На третьому кроці ітераційно вибираються центри кластерів серед точок з максимальними потенціалами. Розглянемо гірський метод кластеризації докладніше.

На першому кроці необхідно сформувати потенційні центри кластерів. Для алгоритму гірської кластеризації число потенційних центрів кластерів (Q) має бути кінцевим. Такими центрами можуть бути як об'єкти кластеризації (рядки матриці **X**), так і вузлові точки в просторі атрибутів. В останньому випадку діапазони зміни вхідних ознак розбивають на декілька інтервалів. Провівши через точки розбиття прямі, паралельні координатним осях, отримуємо сітку. Вузли цієї сітки і відповідатимуть центрам потенційних кластерів. Позначимо через q_r - кількість значень, які можуть приймати центри кластерів по i-й координаті, $i = \overline{1, n}$. Тоді кількість можливих кластерів дорівнюватиме: $Q = q_1 \cdot q_2 \cdot \ldots \cdot q_n$.

На другому кроці алгоритму розраховується потенціал центрів кластерів за такою формулою:

$$P(Z_h) = \sum_{k=\overline{1, M}} \exp\left(-\alpha \cdot D(Z_h, X_k)\right), \quad h = \overline{1, Q},$$

де $Z_h = \left(z_{1,h}, z_{2,h}, .., z_{n,h}\right)$ - потенційний центр h-го кластера, $h = \overline{1, Q}$;

α - додатня константа;

$D(Z_h, X_k)$ - відстань між потенційним центром кластера (Z_h) та об'єктом кластеризації (X_k). В евклідовому просторі ця відстань розраховується так:

$$D(Z_h, X_k) = \sqrt{\left\| Z_h - X_k \right\|^2}.$$

За двох ознак ($n = 2$), графічне зображення розподілу потенціалу буде являти собою поверхню, що нагадує гірський рельєф (рис. 37). Звідси і назва - гірський метод кластеризації.

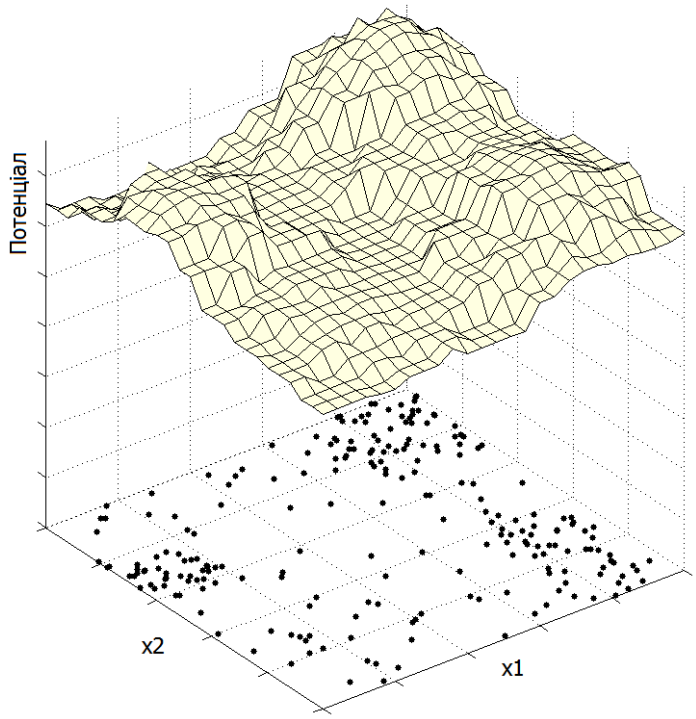


Рисунок 37 – Розподіл потенціалу під час кластеризації за гірським методом

На третьому кроці алгоритму центрами кластерів призначають координати «гірських вершин». Центром першого кластера призначають точку з найбільшим потенціалом. Зазвичай, найвища вершина оточена декількома досить високими піками. Призначення центром наступного кластера точки з максимальним потенціалом серед решти вершин, призвело б до виділення великої кількості близько розташованих центрів кластерів. Отже, перед вибором наступного центру кластера необхідно виключити вплив тільки що знайденого кластера. Для цього потенціал можливих центрів кластерів перераховують наступним чином: від поточного потенціалу віднімають внесок центру щойно знайденого кластера. Перерахунок потенціалу відбувається за такою формулою:

$$P_2(Z_h) = P_1(Z_h) - P_1(V_1) \cdot \exp\left(-\beta \cdot D(Z_h, V_1)\right), \quad Z_h \neq V_1, \quad h = \overline{1, Q},$$

де $P_1(.)$ - потенціал на першій ітерації;

$P_2(.)$ - потенціал на другій ітерації;

V_1 - центр першого знайденого кластера - $V_1 = \underset{Z_1, Z_2, ..., Z_Q}{\arg} \max\left(P_1(Z_1), P_1(Z_2), ..., P_1(Z_Q)\right)$;

β – додатна константа.

Центр другого кластера обирають за максимумом оновленого потенціалу:

$$V_2 = \underset{Z_h:\ Z_h \neq V_1,\ h=\overline{1,Q},}{\arg} \max\left(P_2(Z_1), P_2(Z_2), ..., P_2(Z_Q)\right).$$

Потім знову перераховується значення потенціалів:

$$P_3(Z_h) = P_2(Z_h) - P_2(V_2) \cdot \exp\left(-\beta \cdot D(Z_h,\ V_2)\right),\ \ Z_h \neq V_1,\ \ Z_h \neq V_2,\ \ h = \overline{1, Q}.$$

Ітераційна процедура виділення центрів кластерів триває до тих пір, поки максимальне значення потенціалу перевищує певний поріг.

В середовищі MATLAB гірська кластеризація реалізована функцією **subclust**. Обов'язковими аргументами функції є матриця спостережень **X**. Додатковими аргументами є параметри алгоритму кластеризації. Функція викликається у такому форматі:

```
[centers, sigmas] = subclust (X, radii, xBounds, options),
```

де **centers** – центри знайдених кластерів;

sigmas – радіуси знайдених кластерів;

X – матриця спостережень **X**;

radii – вектор радіусів кластерів за кожною ознакою; за малих значень **radii** функція виокремлює багато дрібних кластерів;

xBounds – діапазони атрибутів, які необхідні для масштабування даних на одиничний гіперкуб;

options – параметри кластеризації;

options(1) – коефіцієнт липкості, значення за замовченням – 1.25. Значення **options(1)*radii** використовується для визначення близьких до центру кластера об'єктів. Ці об'єкти вважаються такими, що належать кластеру і вилучаються із подальшого аналізу;

options(2) – коефіцієнт прийняття, значення за замовченням – 0.5. Якщо відношення потенціалів вузлової точки та центру першого кластера більше за цей коефіцієнт, тоді вузлова точка включається до списку можливих центрів кластерів;

options(3) – коефіцієнт відторгнення, значення за замовченням – 0.15. Якщо вузлова точка відхилена за коефіцієнтом прийняття, тоді вона може потрапити у список потенціальних центрів кластерів за двох умов: 1) точка розташована далеко від вже знайдених кластерів; 2) відношення її потенціалу до потенціалу центра першого кластера більше за коефіцієнт відторгнення;

options(4) – управління виводом на екран проміжних даних. За замовченням дані не виводяться.

Гірську кластеризацію також можна виконати в модулі Findcluster. Основне графічне вікно модуля Findcluster з прикладом кластеризації ірисів зображено на рис. 38. Призначення специфічних полів модуля є таким: Influence Range - важливість ознак (radii); Squash - коефіцієнт липкості; Accept Ratio - коефіцієнт прийняття та Reject Ratio - коефіцієнт відторгнення.

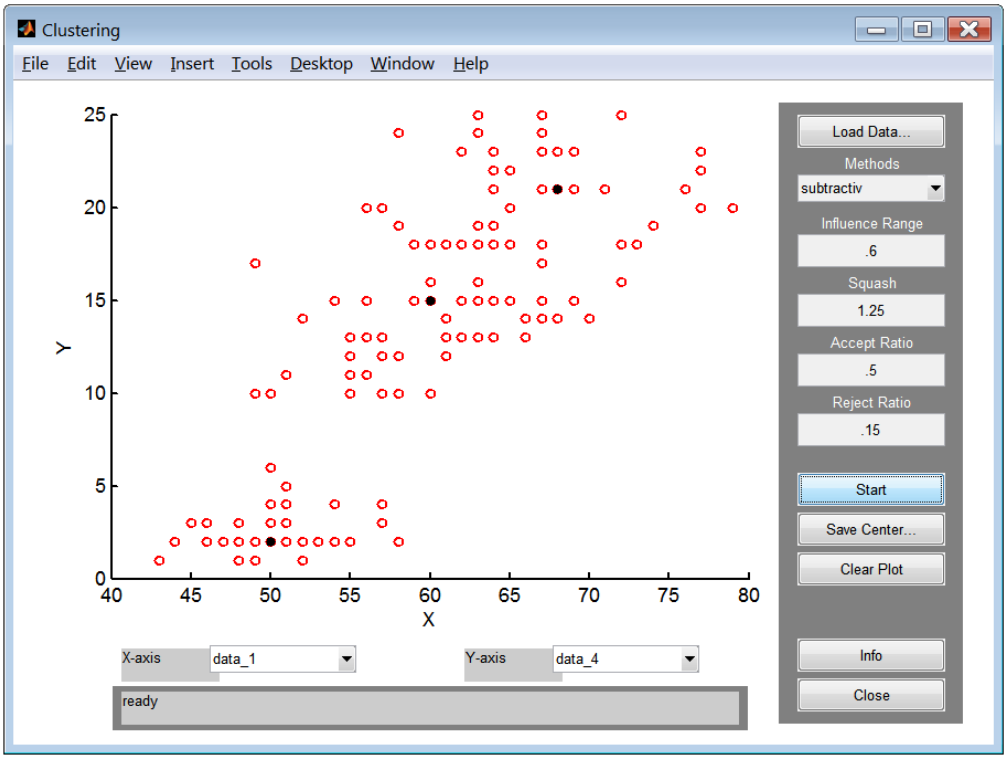


Рисунок 38 – Гірська кластеризація в модулі Findcluster

3.5. Функції для кластеризації в системі MATLAB

Для кластеризації даних доцільно використовувати такі функції:

clusterdata – ієрархічна кластеризація з різноманітними параметрами;

dbscan – кластеризація зашумлених просторових даних на основі функції щільності за методом DBSCAN (Density-Based Spatial Clustering of Applications with Noise);

dendrogram – графічне представлення дерева ієрархічної кластеризації;

fcm – кластеризація за методом нечітких *c*-середніх;

fitgmdist – кластеризації на основі гаусової суміші з максимізією апостеріорної ймовірності кластерів;

gplotmatrix – побудова двофакторних розподілів класів;

kmeans – кластеризація за методом *c*-середніх**;**

kmedoids – кластеризація за робастним аналогом методу *c*-середніх**;**

linkage – розрахунок дерева ієрархічної кластеризації**;**

spectralcluster – спектральна кластеризація;

subclust – гірська кластеризація**.**

3.6. Питання для самоконтролю та розвитку

1. В чому принципова відмінність класифікації та кластеризації?
2. Яка обчислювальна складність алгоритмів кластеризації?
3. Чи можна кластеризувати об’єкти з категоріальними атрибутами?
4. Як врахувати різну важливість ознак під час кластеризації?
5. Яким чином результати кластеризації можна використати для побудови класифікаторів?
6. Яким чином результати кластеризації можна використати для регресійного аналізу?
7. Для географічних даних якого типу доцільно використовувати манхетенську відстань?
8. Чи може у кластер входити лише 1 об’єкт?
9. Як оцінити схожість кластерів?
10. Яку метрику обрати для об’єктів з порядковими атрибутами?
11. Яку метрику обрати для об’єктів з атрибутами різних типів?
12. Для вирішення яких практичних задач використовується кластеризація?

Література

1. Андреев И.М. Описание алгоритма CART / И.М. Андреев // Exponenta Pro: Математика в приложениях. – 2004. – №3–4. – С. 48-53.

2. Воронцов К.В. Комбинаторный подход к оценке качества обучаемых алгоритмов // Математические вопросы кибернетики. – 2004. – Т. 13. – С. 5-36.

3. Городецкий В.И. Методы и алгоритмы коллективного распознавания / Городецкий В.И., Серебряков С.В. // Автоматика и телемеханика. – 2008. – №11. – С. 3-40.

4. Дьяконов А.Г. Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab (практикум на ЭВМ кафедры математических методов прогнозирования) / А.Г. Дьяконов. – МАКСПресс, 2010. – 278 с. – Режим доступу: http://www.machinelearning.ru/wiki/images/7/7e/Dj2010up.pdf

5. Ивахненко А.Г. Долгосрочное прогнозирование и управление сложными системами / А.Г. Ивахненко. –К.: Техніка, 1975. – 312 с.

6. Начало работы с MATLAB / Пер. с англ. Конюшенко В.В. – Softline Co., 2014. – 74 с. Режим доступу: http://www.exponenta.ru/educat/free/matlab/gs.pdf .

7. Орлов А.И. Прикладная статистика. Учебник. / А.И. Орлов. – М.: Издательство “Экзамен”, 2004. – 656 с.

8. Штовба С.Д., Галущак А.В. Ідентифікація багатофакторних залежностей за допомогою баз знань. Лабораторний практикум: електронний навчальний посібник. – Вінниця: Вінницький національний технічний університет, 2016. – 96 с.

9. Штовба С.Д., Мазуренко В.В. Інтелектуальні технології ідентифікації залежностей. Лабораторний практикум: електронний навчальний посібник. – Вінниця: Вінницький національний технічний університет, 2014. – 113 с.

10. Штовба С.Д. Проектирование нечетких систем средствами MATLAB. М.: Горячая линия - Телеком, 2007.- 288 с

11. Bache K. UCI Machine Learning Repository / Bache K. Lichman M.. Irvine: University of California, School of Information and Computer Science. 2014. – Режим доступу: http://archive.ics.uci.edu/ml .

12. Carbonneau M.A. Multiple instance learning: A survey of problem characteristics and applications / M.-A. Carbonneaua, V. Cheplygina, E. Grangera, G. Gagnon // Pattern Recognition. – 2018. – Vol. 77. – P. 329–353.

13. Kuncheva L. Combining pattern classifiers: methods and algorithms / L. Kuncheva. – John Wiley & Sons, 2004. – 350 p.

14. .Shtovba S., Shtovba O., Petrychko M. Detection of Social Network Toxic Comments with Usage of Syntactic Dependencies in the Sentences / Proc. of the Second International Workshop on Computer Modeling and Intelligent Systems, Zaporizhzhia, Ukraine, April 15-19, 2019. CEUR Workshops Proceeding, Vol. 2353. – 2019. – P. 313-323

15. Zambon, M., Lawrence, R., Bunn, A., Powell, S. Effect of alternative splitting rules on image processing using classification tree analysis // Photogrammetric Engineering & Remote Sensing. – 2006. – Vol. 72, №1, P. 25-30.

16. Zhang M. L., A review on multi-label learning algorithms / M.L. Zhang, Z. H. Zhou // IEEE transactions on knowledge and data engineering. – 2014. – Vol. 26. – №.8. – P. 1819-1837.

Навчальне видання

Штовба Сергій Дмитрович
Козачко Олексій Миколайович

Machine Learning: стартовий курс

Електронний навчальний посібник

Друкується в авторській редакції.

Оригінал-макет підготовлено авторами.

Формат 29,7х42¼.
Гарнітура Times New Roman

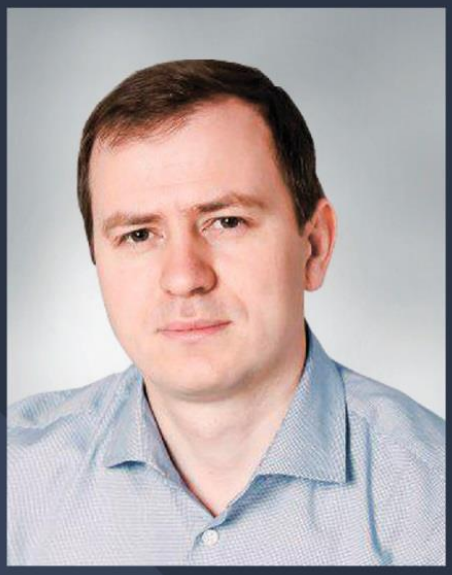
Вінницький національний технічний університет
навчально-методичний відділ ВНТУ
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, к. 2201.
Тел. (0432) 59-87-36.
Свідоцтво суб'єкту видавничої справи
серія ДК № 3516 від 01.07.2009 р.

Штовба
Сергій Дмитрович

Професор, доктор технічних наук, професор кафедри комп`ютерних систем управління Вінницького національного технічного університету.
В 1993 р. закінчив Вінницький політехнічний інститут за спеціальністю «Конструювання та технологія електронно-обчислювальних засобів».
В 1997 р. захистив кандидатську дисертацію з математичного моделювання, а в 2009 р. – докторську дисертацію з інформаційних технологій. Автор 9 книг та понад 100 статей в галузі інформаційних систем та штучного інтелекту.

www.shtovba.vk.vntu.edu.ua;
shtovba@vntu.edu.ua;

www.shtovba.vinnitsa.com
shtovba@gmail.com



Козачко
Олексій Миколайович

Доцент, кандидат технічних наук, доцент кафедри системного аналізу, комп`ютерного моніторингу та інженерної графіки Вінницького національного технічного університету.
В 2002 р. закінчив Вінницький національний технічний університет за спеціальністю «Комп'ютеризовані системи управління і атоматика».
В 2006 р. захистив кандидатську дисертацію з математичного моделювання та обчислювальних методів. Автор 4 книг та понад 40 статей в галузі інформаційних технологій.

www.kozachko.vk.vntu.edu.ua

aleksey.kozachko@gmail.com