Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Л.Я. Кулаковський, А.В. Босак, В.О. Броницький

МЕТОДИ РОЗПІЗНАВАННЯ ОБРАЗІВ В ЕЛЕКТРОТЕХНІЧНИХ СИСТЕМАХ

Рекомендації до виконання розрахункової роботи

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня доктора філософії
за освітньою програмою «Електроенергетика, електротехніка та електромеханіка»
спеціальності G3 Електрична інженерія

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2026

УДК 621.311
В19

Автори: *Кулаковський Леонід Ярославович*, канд. техн. наук, доц.
Босак Алла Василівна, канд. техн. наук, доц.
Броницький Вадим Олегович, канд. техн. наук, доц.

Рецензент *Дерев'янко, Д.Г.*, д.т.н., доцент, завідувач кафедри АЕ

Відповідальний редактор *Торопов А.В.* , канд. техн. наук, доц.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 8 від 04.06.2026 р.)
за поданням вченої ради навчально-наукового інституту енергозбереження та енергоменеджменту
(протокол № 10 від 28.05.2026 р.)

Кулаковський Л. Я.

В19 Методи розпізнавання образів в електротехнічних системах. [Електронний ресурс] : рек. до виконання розрахунк. роботи : навч. посіб. для здобувачів ступеня д-ра філософії за освіт. програмою «Електроенергетика, електротехніка та електромеханіка» спец. G3 Електрична інженерія / Л. Я. Кулаковський, А. В. Босак, В. О. Броницький; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2026. – 59 с.

У представленому навчальному посібнику викладено сучасні підходи та практичні методи розпізнавання образів, що застосовуються для аналізу, діагностики та класифікації режимів роботи електроенергетичних систем. Особливу увагу приділено практичній реалізації повного циклу побудови системи розпізнавання образів – від роботи з сирими даними (трифазні напруги та струми) до формування підсумкових моделей та оцінювання їх ефективності. Посібник містить опис типових задач, що охоплюють моніторинг стану обладнання, виявлення гойдань потужності, класифікацію електричних пошкоджень, діагностику дефектів машин, аналіз профілів навантаження та кластеризацію аварійних подій. Запропоновано покрокову методику виконання розрахунково-графічної роботи, що включає аналіз даних, побудову ознак, використання методів розпізнавання образів, оцінювання якості моделей та інтерпретацію отриманих результатів. Навчальний посібник призначений для здобувачів ступеня доктора філософії за спеціальністю G3 Електрична інженерія освітньої програми "Електроенергетика, електротехніка та електромеханіка". Поряд із цим, посібник буде також корисним науковцям, що працюють в області автоматизації, Smart Grid-технологій та проводять аналіз режимів енергосистем і застосовують машинне навчання у технічних задачах.

УДК 621.311

Реєстр. № НП XX/XX-XXX. Обсяг 2,7 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

Перелік умовних скорочень

COMTRADE – стандарт формату реєстрації аварійних подій у енергосистемах.
CNN – Convolutional Neural Network, згорткова нейронна мережа.
DAQ – Data Acquisition System, система збору даних.
DBSCAN – Density-Based Spatial Clustering of Applications with Noise, алгоритм кластеризації за щільністю.
ENTSO-E – European Network of Transmission System Operators for Electricity, європейська мережа операторів систем передачі.
FFT – Fast Fourier Transform, швидке перетворення Фур'є.
GMM – Gaussian Mixture Model, модель гаусових сумішей.
k-NN – k-Nearest Neighbors, метод k найближчих сусідів.
LR (Logistic Regression) – логістична регресія.
ML – Machine Learning, машинне навчання.
MSE – Mean Squared Error, середньоквадратична помилка.
PCA – Principal Component Analysis, метод головних компонент.
P_mean, P_std – average active power and its standard deviation, середнє значення та стандартне відхилення активної потужності.
PMU – Phasor Measurement Unit, пристрій вимірювання фазорів.
PSCAD – Power Systems Computer-Aided Design, середовище моделювання електроенергетичних систем.
PU / pu – per unit, відносна система одиниць.
PRPD – Phase Resolved Partial Discharge, фазорозділені часткові розряди.
RBF – Radial Basis Function, радіально-базисна функція (ядро SVM).
ReLU – Rectified Linear Unit, нелінійна функція активації.
RF (Random Forest) – ансамбль дерев рішень.
RMS – Root Mean Square, середньоквадратичне значення.
SVM – Support Vector Machine, машина опорних векторів.
STFT – Short-Time Fourier Transform, короткочасне перетворення Фур'є.
V1, V2 – positive-sequence and negative-sequence components, складові прямої та зворотної послідовностей.
Z_est – Estimated Impedance, оцінений імпеданс мережі.
ВДЕ – відновлювані джерела енергії.
КЗ – коротке замикання.

ВСТУП

У сучасних електроенергетичних системах стрімке зростання складності мереж, поява великої кількості силової електроніки, розвиток Smart Grid-технологій та широке впровадження систем моніторингу створюють необхідність застосування інтелектуальних методів аналізу даних. Одним із ключових напрямів такого аналізу є розпізнавання образів, що дозволяє автоматично ідентифікувати режими роботи, виявляти аномалії, прогнозувати небезпечні стани та приймати рішення в реальному часі.

Електроенергетика – це галузь, де точність і швидкість обробки інформації мають критичне значення. Короткі замикання, гойдання потужності, коливання напруги, нестійкість генераторів, асиметрія фаз – усі ці явища мають характерні ознаки у часових, частотних та просторових доменах. Традиційні алгоритми релейного захисту часто не здатні охопити повний спектр сучасних режимів, особливо у випадках складної динаміки, шумів, перехідних процесів та взаємодії елементів системи. Саме тому методи машинного навчання, статистичного аналізу, обробки сигналів та інтелектуальних систем стають невід'ємною складовою сучасного енергетичного інженерного інструментарію.

Для України питання впровадження інтелектуальних систем у енергетику має особливу актуальність. Енергосистема країни зазнає значних навантажень через:

- інтеграцію до європейської мережі ENTSO-E,
- збільшення частки ВДЕ (сонячних і вітрових електростанцій) у загальному балансі,
- потребу у підвищенні стійкості системи до аварійних режимів,
- необхідність швидкого та надійного відновлення енергопостачання,
- роботу в умовах форс-мажорних викликів та нестабільних мережевих конфігурацій.

У цих умовах системи моніторингу, діагностики та прогнозування, побудовані на методах розпізнавання образів, стають важливими елементами безпеки й ефективності. Вони дають змогу:

- своєчасно виявляти аварійні режими та потенційні загрози,
- відокремлювати небезпечні стани від нормальних коливань,
- підвищувати надійність релейного захисту,
- оптимізувати керування обладнанням,
- покращувати якість електричної енергії.

Для аспірантів і молодих науковців володіння методами розпізнавання образів стає фундаментальною компетентністю. Вони відкривають можливості для:

- побудови новітніх цифрових систем захисту та автоматизації,
- дослідження поведінки електроенергетичних об’єктів у перехідних режимах,
- розробки алгоритмів селективного й адаптивного захисту,
- створення гнучких інтелектуальних енергетичних систем (Intelligent Power Systems).

Сучасні енергетичні компанії, наукові центри та цифрові підстанції потребують фахівців, які здатні працювати з великими масивами даних, будувати моделі класифікації, здійснювати очистку й нормалізацію сигналів, аналізувати часові ряди та застосовувати машинне навчання для підтримки критичних інженерних рішень.

Запропонована методика допоможе сформувати ці компетентності, даючи студентам і аспірантам можливість на практиці опанувати сучасні підходи до аналізу енергосистем, побудови ознак, класифікації режимів та створення цифрових моделей для підвищення надійності і безпеки електропостачання.

Метою цих методичних вказівок є формування в аспірантів цілісної системи знань і практичних навичок щодо застосування методів розпізнавання образів для аналізу, діагностики та класифікації режимів роботи електроенергетичних систем.

Методичка спрямована на те, щоб забезпечити оволодіння сучасними підходами до:

- опрацювання вимірювальних даних електричних величин;
- очищення, нормалізації та підготовки ознак для машинного навчання;
- побудови моделей класифікації нормальних та аварійних режимів;
- аналізу поведінки трифазних сигналів у часовій, частотній та векторній областях;
- використання інтелектуальних алгоритмів для підвищення надійності релейного захисту та систем моніторингу.

Особлива увага приділяється вивченню способів автоматизації виявлення небезпечних станів, таких як гойдання потужності, асиметрія та короткі замикання, а також методам контролю якості даних, що є критично важливими для побудови достовірних систем розпізнавання.

Методичні вказівки мають на меті підготувати здобувачів до здійснення наукової та інженерної діяльності у сфері цифрових енергетичних технологій, розвитку Smart Grid, підвищення стійкості та безпеки електроенергетичних мереж України, а також до використання машинного навчання й інтелектуальних систем у реальних промислових застосуваннях.

Основним **завданням розрахункової роботи** є розробка системи розпізнавання образів на основі наданих або сформованих даних, виконати їх очищення, побудову інформативних ознак, навчання кількох моделей класифікації або кластеризації та оцінити їхню ефективність із використанням стандартних метрик якості. За результатами роботи студент повинен продемонструвати навички повного циклу підготовки даних і побудови моделей розпізнавання. Здобувач повинен реалізувати повний цикл побудови системи розпізнавання – від первинного аналізу даних до створення моделі та оцінки її ефективності.

Можливі варіанти тематик завдань до розрахункової роботи

Блок 1 – Моніторинг стану електротехнічного обладнання (High-Voltage, Machines, Drives)

Задача 1. Виявлення симетричних несправностей під час гойдання потужності

Постановка задачі

Необхідно класифікувати режим електроенергетичної системи:

- нормальний режим,
- гойдання потужності (Power Swing),
- симетричні короткі замикання (three-phase fault).

Необхідно визначити, чи можна розмежувати PS і КЗ за вибраними ознаками.

Дані:

- Тимчасові ряди струмів та напруг (Ia, Ib, Ic, Va, Vb, Vc)
- Частота вибірки: 1–10 kHz (підходить і PMU 50/100 кадров)
- Можуть бути надані:
    - реальні дані PMU / COMTRADE
    - згенеровані дані з моделювання (Simulink, PSCAD)

Методи:

- PCA для виділення ключових ознак гойдань
- SVM, Decision Trees / Random Forest для класифікації «гойдання / КЗ»
- CNN для класифікації спектрограм або траєкторій у dq0
- Кластеризація DBSCAN для виділення «аномальних» режимів

Очікуваний результат

Побудувати модель, яка:

- розмежовує гойдання та симетричні КЗ
- визначає найінформативніші ознаки
- дає оцінку точності (confusion matrix)

Що потрібно виконати

1. Завантажити/згенерувати дані режимів.
2. Побудувати ознаки: миттєві величини, dq0 координати, спектральні характеристики.
3. Виконати класифікацію щонайменше двома методами.
4. Порівняти точність методів і зробити висновки.
5. Оцінити, чи можна застосовувати вибраний метод до системи релейного захисту.

Задача 2. Виявлення гойдання потужності

Постановка задачі

Визначити тип гойдання:

- стійке (stable swing)
- нестійке (unstable swing)
- помилкове спрацювання релейного захисту (optional)

Дані:

- Тимчасові ряди активної та реактивної потужності

- Фазорні траєкторії (Voltage-Current trajectory)
- Дані різних сценаріїв PS

Методи:

- Класифікація за фазовими траєкторіями (shape recognition)
- K-Means або Gaussian Mixture Models–кластеризація траєкторій
- LSTM або GRU для класифікації часових рядів

Очікуваний результат

Здобувач визначає тип гойдання з точністю ≥ 80%.

Що потрібно виконати

1. Перетворити дані у фазові траєкторії або P–Q площину.
2. Провести кластеризацію.
3. Побудувати класифікатор.
4. Порівняти моделі та визначити оптимальний підхід.

Задача 3. Класифікація видів електричних пошкоджень

Постановка задачі

Потрібно розпізнати тип пошкодження:

- однофазне КЗ
- двофазне КЗ
- двофазне КЗ на землю
- трифазне КЗ
- нормальний режим

Дані:

- Струми та напруги фаз
- Миттєві значення, комплексні значення, похідні
- COMTRADE-файли реальних аварій (може надати викладач)

Методи:

- Random Forest
- SVM
- CNN (для спектрограм струмів/напруг)
- Кластеризація K-Means для попереднього аналізу

Очікуваний результат

Модель визначає тип КЗ з точністю ≥ 85%.

Що потрібно виконати

- Провести первинну обробку (фільтрація, нормалізація)
- Виділити ознаки (амплітуди, симетричні складові, гармоніки)
- Навчити модель
- Побудувати confusion matrix
- Написати висновок, які ознаки найбільш інформативні

Блок 2. Діагностика електричних машин

Задача 4. Діагностика дефектів ротора асинхронної машини

Постановка задачі

Класифікувати стан двигуна на:

- справний
- зламані стрижні ротора
- ексцентриситет
- міжвиткове замикання

Дані:

- Струм статора (сигнал MCSA)
- Вібраційні сигнали (optional)
- Спектри і часові ряди

Методи:

- PCA + кластеризація
- SVM / Random Forest
- 1D CNN для класифікації сигналів
- Autoencoder для виявлення аномалій

Очікуваний результат

Точність ≥ 75–85%.

Виявлення інформативних частот (1×fs, 2×fs, sidebands).

Що потрібно виконати

1. Провести FFT, визначити основні частотні компоненти.
2. Побудувати модель класифікації.
3. Визначити, які компоненти спектра найбільш важливі.
4. Порівняти моделі.

Задача 5. Виявлення деградації ізоляції високовольтного обладнання

Постановка задачі

Розпізнати стадії деградації:

- нормальний стан
- рання деградація
- часткові розряди
- критичний стан

Дані:

- Сигнали часткових розрядів
- Вартографія або PRPD-схеми
- Напруги та струми

Методи:

- K-Means для кластеризації патернів ЧР
- DBSCAN для виявлення аномалій
- Мережі CNN для класифікації PRPD-карт

Очікуваний результат

Визначення класу стану обладнання.

Що потрібно виконати

- Побудувати PRPD-карти
- Провести кластеризацію точок
- Навчити модель класифікації
- Сформулювати діагностичний висновок

Блок 3. Управління та технологічні процеси

Задача 6. Класифікація режимів електропривода

Постановка задачі

Визначити режими:

- холостий хід
- номінальне навантаження
- перевантаження
- аварійний режим

Дані:

- Струми приводів
- Швидкість
- Напруга
- Вібрація

Методи:

- SVM
- K-NN
- Нейронні мережі
- Кластеризація t-SNE для виявлення структур даних

Очікуваний результат

Студент формує модель, що ідентифікує режими в реальному часі.

Що потрібно виконати

- Зібрати/згенерувати дані
- Виділити ознаки (RMS, crest factor, THD, ін.)
- Побудувати модель
- Порівняти 3 різні алгоритми

Блок 4. Кластеризація

Задача 7. Кластеризація аварійних подій у мережі

Постановка задачі

Знайти групи подібних аварій у мережі.

Дані:

- Часові ряди аварій
- Значення струмів, напруг, імпеданси
- Метадані подій

Методи:

- K-Means
- DBSCAN
- Hierarchical clustering

Очікуваний результат

Аварії поділено на кілька груп з поясненням.

Що потрібно виконати

- Провести нормалізацію даних
- Обрати метрику відстані

- Побудувати дендрограму (якщо потрібно)
- Формально описати характер груп

Задача 8. Кластеризація графіків навантаження

Постановка задачі

Поділити споживачів на групи за характером навантаження.

Дані:

- Годинні/хвилинні профілі споживання
- Статистичні характеристики навантаження

Методи:

- DTW (Dynamic Time Warping)
- K-Means
- SOM (Self-Organizing Maps)

Очікуваний результат

Кілька кластерів типових профілів навантаження.

Що потрібно виконати

- Обрати метрику для тимчасових рядів
- Побудувати кластеризацію
- Зробити візуалізацію центрів кластерів
- Пояснити, що означає кожен кластер

Методика виконання роботи

Методика виконання розрахунково-графічної роботи спрямована на формування в студента вмінь:

- працювати з даними роботи електротехнічних систем та комплексів, проводити їх аналіз, зокрема аналізувати часові ряди напруги та струму для моніторингу роботи обладнання;
- виділяти інформативні ознаки;
- застосовувати методи класифікації та кластеризації;
- коректно інтерпретувати результати з погляду електроенергетики та релейного захисту.

У рамках розрахунково-графічної роботи студенти повинні:

- отримати базове розуміння фізики процесів;
- навчитися обробляти та аналізувати вимірювальні сигнали;
- виділяти інформативні ознаки;
- застосовувати методи класифікації та кластеризації для розмежування нормального режиму, гойдань і КЗ, оцінки стану обладнання, підтримки функцій релейного захисту;
- оцінювати ефективність побудованих моделей;
- формувати технічні висновки.

Орієнтовний план виконання розрахункової роботи

І. В першому розділі розрахунково-графічної роботи аспірантам необхідно описати процес або режими роботи електроенергетичної системи для яких

необхідно розробити систему розпізнавання образів. Необхідно розкрити задачу і цілі дослідження, описати фізичний зміст процесу, навести необхідні формули, що розкривають роботу обладнання процесу. Для цього повинні бути наведені короткі теоретичні відомості, дані необхідні для аналізу процесу, опис фізичної природи цих процесів і того, як вони проявляються у вимірювальних сигналах. Також потрібно обгрунтувати необхідність застосовування класифікації чи кластеризації процесів, явищ, ознак для покращення роботи системи в цілому та основні цілі дослідження.

II. Наступним етапом має бути характеристика та короткий аналіз даних, що мають бути використані для побудови моделі розпізнавання образів. Необхідно дати визначення кожній ознаці та параметру, що використовуються в датасеті, навести одиниці виміру, можливий діапазон зміни даних (за необхідності) та описати яким чином дані були отримані і можуть бути отримані в майбутньому (контрольні виміри, показники виробничих приборів, умов експлуатації тощо).

III. Третім етапом має бути ознайомлення з даними. В Jupiter Notebook необхідно завантажити дані провести перегляд структури датасету, кількості вибірок та міток. При необхідності можна провести аналіз динаміки зміни даних, ідентифікацію стрибків та різного роду викидів тощо. Також повинна бути проведена попередня обробка данів (Data Preprocessing**)** – згладжування, обчислення ковзних RMS, медіани**,** дисперсій, аналіз даних на нормальний закон розподілу, розрахунок t-критерія Стьюдента, Фішера (за необхідності) тощо. Тобто бажана наявність статистичної оцінки даних.

IV. Четвертим етапом дослідження має бути очищення данних (Data Cleaning). Потрібно очистити дані відповідно до коротких теоретичних відомостей про очищення данних наведених в попередньому розділі даних методичних вказівок в залежності від стуктури та типів даних для дослідження. Після цього за необхідності має проводитись нормалізація ознак, зокрема і за допомогою масштабування.

V. На основі отриманих даних на 5 етапі необхідно проводити виділення ознак (feature engineering). Для цього можна побудувати гістограми основних ознак, теплову карту кореляцій, розрахунок критерія Пірсона. При необхідності формуванні цільових функцій та вирішення задач регресійного аналізу можливе застосування методу групового урахування аргументів та інших методів для виділення інформативних ознак.

VI. На 6 етапі необхідно сформувати вибірки для моделі (тренувальну, тестову, за необхідності контрольну) та застосування методів розпізнавання образів значну частину яких описано в цій методичці для виконання розрахункової роботи. Під час побудови моделей необхідно проводити оцінювання якості моделей за відповідними метриками (точність (accuracy), повнота (recall), F1-міра тощо).

VII. На сьомому етапі необхідно провести інтерпритацію результатів, зокрема, візуалізація кластерів або проєкцій PCA, описати фізичний зміст отриманих кластерів або класів, пояснити чому було зроблено вибір певного методу розпізнавання образів, чому його використання є ефективним та було

застосовано для вирішення поставленої в роботі цілі. По проведеному дослідженні необхідно зробити висновок.

Робота виконується в текстовому редакторі Word із добавлення частини python скрипта, що був застосований в середовищі Notebook Jupiter. Оформлений нотбук з використання мови Markdown для виділення етапів дослідження, пояснені певних частин скрипту, візуалізації також повинен бути доданий до Звіту виконаному в текстовому редакторі Word. Тобто розрахунково-графічна робота має складатись із процесу побудови моделі в notebook Jupiter (файл формату .ipynb) та Звіту по виконання роботи в текстовому редакторі Word. Офоромлення Звіту має бути виконано згідно із нормативів оформлення студентських та наукових робіт і включатиме в себе титульний аркуш та результат виконання всіх 7 (за необхідності) етапів дослідження наведених в методичних вказівках.

Короткі теоретичні відомості про методи розпізнавання образів

Сучасні системи розпізнавання образів використовують математичні та алгоритмічні методи, що дозволяють обробляти великі масиви даних, виділяти ознаки, аналізувати просторові або часові патерни та приймати рішення щодо режиму роботи чи класу об’єкта.

Нижче наведено короткий опис основних методів, які застосовуються в задачах класифікації та кластеризації режимів електроенергетичних систем.

1. Метод головних компонент (PCA – Principal Component Analysis)

PCA – це метод зменшення розмірності, який перетворює вихідні ознаки у новий ортогональний простір, де перша компонента пояснює найбільшу дисперсію даних, друга – наступну найбільшу, і так далі.

Основні властивості PCA

- Зменшує кількість ознак без значної втрати інформації.
- Видаляє корельованість між ознаками.
- Робить дані більш зручними для візуалізації (2D/3D).
- Дозволяє побачити приховану структуру даних.

PCA зменшує розмірність ознак шляхом знаходження ортогональних напрямків (головних компонент), у яких дисперсія даних максимальна.

Нехай маємо матрицю центруваних даних

$$X \in \mathbb{R}^{N \times d},$$

де N – кількість об’єктів (векторів ознак); d – кількість ознак, X – матриця, де і-й рядок – це вектор ознак Xi.

тоді:

1. Обчислюється коваріаційна матриця:

$$C = \frac{1}{N-1} X^{\top} X_{\mathrm{c}},$$

де C – коваріаційна матриця $d \times d$, $X^\top$– транспонована центрована матриця даних

Кожен елемент C_{ij} показує, як ознаки і та j змінюються разом.

2. Розв'язується задача власних значень:

$$Cv_i = \lambda_i v_i.$$

де v_i – власний вектор (напрямок головної компоненти); λ_i– власне значення (пов'язане з дисперсією уздовж компоненти).

Чим більше λ_i, тим більшу варіацію пояснює ця компонента. Головні компоненти – власні вектори v_i, відсортовані за спаданням λ_i.

3. Трансформація даних:

$$Z = XV_k,$$

де V_k– матриця перших k власних векторів, Z – дані у просторі головних компонент

Кожен рядок Z– це нові ознаки після зменшення розмірності.

Python: бібліотека та мінімальний приклад

```
import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
Z = pca.fit_transform(X)   # X – матриця ознак
```

2. SVM – Машина опорних векторів (Support Vector Machine)

SVM – один із найефективніших методів класифікації. Ідея полягає у пошуку гіперплощини, що максимально розділяє класи.

Переваги SVM:

- Ефективний на даних малої та середньої розмірності.
- Стійкий до викидів.
- За допомогою ядрових функцій (RBF, polynomial) може розділяти дані у нелінійних задачах.

Що він робить?

- Створює межу з максимальним відступом (margin) між класами.
- Опорні вектори – точки, що визначають цю межу.

Математична постановка

Для бінарної класифікації SVM шукає розділяючу гіперплощину:

$$w^\top x + b = 0$$

де x – вектор ознак (наприклад, [V_{rms_a}, V_{1_mag}, $P_{\_mean}$, env_std]); w– вектор ваг, перпендикулярний до гіперплощини; b– зміщення (інтерсепт).

Гіперплощина розділяє класи в ознаковому просторі.

Оптимізаційна задача:

$$\min_{w,b} \frac{1}{2} \| w \|^2$$

за умов:

$$y_i(w^\top x_i + b) \geq 1.$$

де y_i – правильна мітка класу для вектора x_i: традиційно $y_i \in \{-1, +1\}$;
$\| w \|^2$– квадрат довжини вектора ваг (чим менший – тим ширший відступ).

Для нелінійних задач застосовують ядро $K(x_i, x_j)$, найпоширеніше – RBF:

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

де $K(x, x')$– значення ядра між двома точками (міра подібності), γ– параметр "гладкості" ядра, $\| x - x' \|$ – евклідова відстань між двома векторами ознак

RBF-ядро перетворює дані в нескінченновимірний простір, де класи стають лінійно відокремними.

Python: бібліотека та мінімальний приклад

```
from sklearn.svm import SVC
model = SVC(kernel='rbf', C=1.0, gamma='scale')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

3. Decision Trees та Random Forest

3.1. Decision Tree – Дерево рішень

Дерево рішень – це модель, що приймає рішення шляхом послідовних логічних розгалужень за критеріями ознак.

Переваги:

- Інтерпретованість.
- Легко відслідковувати логіку класифікації.
- Працюють з нелінійними залежностями.

Недолік:

- Схильність до перенавчання.

Для задач класифікації дерево використовує ентропію, тобто виконує функцію інформаційного критерію.

$$H(p) = -\sum_i p_i \log_2 p_i,$$

де $H(p)$ – ентропія вузла; міра невизначеності або "безладу" у розподілі класів;
p_i – частка (ймовірність) елементів класу iii у даному вузлі дерева

Чим більш рівномірний розподіл класів – тим більша ентропія.

Також може бути використаний індекс Джині:

$$G = 1 - \sum_i p_i^2 ,$$

де G – коефіцієнт Джині (міра нечистоти вузла).

Якщо значення $G = 0$ – у вузлі лише один клас (ідеальна чистота), якщо $G = 0.5$ – змішана структура (наприклад, 50/50 два класи). Розбиття обирається за максимальним приростом інформації:

$$\Delta = H_{\text{parent}} - \sum_k \frac{N_k}{N} H_k,$$

де Δ – приріст інформації після розбиття вузла; H_{parent} – ентропія до розбиття (у "батьківському" вузлі); H_k– ентропія k-го дочірнього вузла; N_k– кількість елементів у k-му дочірньому вузлі; N– загальна кількість елементів у батьківському вузлі.

Величина $\frac{N_k}{N}$ – вага (частка) цього дочірнього вузла. Чим більше Δ – тим краще розбиття (менше змішування класів).

Python: бібліотека та мінімальний приклад

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=5)
tree.fit(X_train, y_train)
```

3.2. Random Forest – Випадковий ліс

Random Forest – ансамбль багатьох дерев, кожне з яких навчається на випадковій підмножині даних і ознак.

Переваги:

- Висока точність.
- Низька чутливість до шуму.
- Автоматична оцінка важливості ознак.
- Практично завжди перевершує окреме дерево.

Ансамбль M дерев, які будуються на різних вибірках з випадковими підмножинами ознак.

Формула агрегування для класифікації:

$$\hat{y} = \text{mode}\big(h_1(x), h_2(x), \ldots, h_M(x)\big),$$

де $\hat{y}$ – Підсумковий прогноз моделі, тобто клас, який алгоритм вважає правильним для об'єкта (Це рішення отримане шляхом голосування всіх дерев ансамблю) x – Вектор ознак, який подається на вхід моделі. Це можуть бути: RMS-значення, симетричні складові, THD, P–Q ознаки, інші фічі, h_i – Прогноз i-го дерева у складі Random Forest, Кожне дерево h_i – це окремий класифікатор, побудований на випадковій підвибірці даних (bootstrap sampling),

випадковій підмножині ознак. M – загальна кількість дерев у лісі (наприклад, 100, 300 або 1000). Чим більше дерев тим стабільніша модель та нижчий ризик перенавчання, mode(·) – функція мода – знаходить найбільш популярний клас серед прогнозів дерев. Вона вибирає той клас, за який проголосувала більшість дерев.

У моделі Random Forest кожне дерево $h_i(x)$генерує власний прогноз класу для вхідного вектора ознак x. Після цього всі прогнози об'єднуються за принципом більшості: функція mode(·)визначає клас, який зустрівся найчастіше серед $h_1(x), h_2(x), \dots, h_M(x)$. Цей клас і є підсумковим рішенням моделі, позначеним як $\hat{y}$.

Python: бібліотека та мінімальний приклад

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=300, random_state=7)
rf.fit(X_train, y_train)

4. Логістична регресія (Logistic Regression)

Логістична регресія (LR) – це один з базових та найбільш інтерпретованих методів класифікації. Незважаючи на назву, метод не виконує регресію – він оцінює ймовірність належності об'єкта до певного класу, використовуючи логістичну функцію (сигмоїду) для відображення лінійної комбінації ознак у діапазон (0, 1).

У загальному вигляді для двокласової задачі:

$$P(y = 1 \mid x) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}},$$

де *x* – вектор ознак (feature vector) розміру d; y – цільова змінна (клас). Для бінарної задачі $y \in \{0,1\}$; *w* – вектор ваг (коефіцієнтів моделі), b – скаляр – зміщення (bias), інколи називають вільним членом, σ(z) – логістична функція (сигмоїда), P(y=1|x) – ймовірність того, що об'єкт із ознаками xналежить до класу 1.

У мультикласовій задачі застосовується softmax-регресія:

$$P(y = k \mid x) = \frac{e^{w_k^T x}}{\sum_{j=1}^{K} e^{w_j^T x}},$$

де K – кількість класів; k – індекс конкретного класу, k∈{1,…,K}.

Логіка роботи логістичної регресії:

1. Маніпулює лінійними комбінаціями ознак. Модель будує гіперплощини, які відділяють класи у просторі ознак.
2. Оцінює ймовірність. На виході подає не просто клас, а ймовірність належності до кожного класу.
3. Навчається за допомогою оптимізації логістичної втрати (log-loss). Модель знаходить ваги w, які мінімізують кількість «поганих» передбачень.

4. Підтримує регуляризацію (L1/L2). Це допомагає уникати перенавчання, особливо коли ознак багато або коли вони корелюють між собою.

Python: бібліотека та мінімальний приклад

logreg = LogisticRegression(max_iter=500, multi_class='multinomial')
logreg.fit(X_train, y_train)

5. CNN – Згорткові нейронні мережі (Convolutional Neural Networks)

CNN – клас глибинних нейронних мереж, які особливо добре працюють з даними, що мають просторово-часову структуру.

Хоча CNN історично створені для обробки зображень, вони чудово працюють з:

- часовими рядами,
- частотними спектрами,
- перетвореними сигналами (STFT, спектрограми, P–Q траєкторії).

Чому CNN ефективні?

- Згорткові фільтри виявляють локальні патерни.
- Глибокі шари формують ієрархії ознак.
- Менше параметрів, ніж у звичайних MLP.

CNN побудовані на згорткових фільтрах. 2D/1D згортка:

$$(f * g)[n] = \sum_k f[k]\, g[n-k]$$

де $f[k]$ – вхідний сигнал (наприклад, часовий ряд напруги); $g[k]$ – згортковий фільтр (kernel), який виявляє певний патерн; $(f * g)[n]$ – результат згортки в точці n.

Для 2D:

$$(F * K)(i,j) = \sum_m \sum_n F\,(i-m, j-n) K(m,n)$$

де $F(\mathrm{i},\mathrm{j})$ – значення вихідного образу (наприклад, спектрограми) в координаті; $K(m,n)$ – ядро (фільтр), який "вичитує" локальні патерни; $(F * K)(i,j)$– значення після згортки у точці (i,j).

Основні шари CNN:

- Convolution (Згортка). Цей шар застосовує фільтри (kernel), які «ковзають» по сигналу або зображенню, виділяючи локальні патерни, такі як градієнти, зміни амплітуди чи характерні форми сигналу. Він формує нові представлення даних, що підсилюють важливі особливості входу.
- ReLU ((Rectified Linear Unit). ReLU – це нелінійна функція активації, яка пропускає лише додатні значення, а від'ємні зануляє. Вона додає моделі нелінійності, що дозволяє CNN виявляти складніші залежності у даних.
- Pooling (max/avg). Pooling-шар зменшує розмірність даних, агрегуючи інформацію на локальних ділянках (наприклад, шляхом вибору

максимуму або середнього). Це робить модель стійкішою до шумів і зменшує кількість параметрів.

- Fully Connected (Повнозв'язний шар). У цьому шарі кожен нейрон з'єднаний з усіма виходами попереднього шару, що дозволяє моделі узагальнювати виділені CNN-ознаки та приймати фінальне рішення щодо класу. Це «класичний» етап класифікації після згорткових шарів.

Python: бібліотека та мінімальний приклад

```
import torch
import torch.nn as nn
class CNN1D(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv1d(in_channels=1, out_channels=8, kernel_size=5)
        self.relu = nn.ReLU()
        self.fc = nn.Linear(8*L_out, 3)  # 3 класи
    def forward(self, x):
        x = self.relu(self.conv(x))
        x = torch.flatten(x, 1)
        return self.fc(x)
```

6. K-Means та Gaussian Mixture Models (кластеризація)

6.1. K-Means

Алгоритм K-Means шукає центри кластерів, мінімізуючи суму квадратів відстаней точок до центроїдів.

Переваги:

- Простий та швидкий.
- Добре працює при чітких, опуклих кластерах.

Недоліки:

- Погано працює із складними формами кластерів.
- Потребує вказувати k – кількість кластерів.

Алгоритм мінімізує суму квадратів відстаней до центроїдів:

$$J = \sum_{i=1}^{N} \| x_i - \mu_{c(i)} \|^2,$$

де $c(i)$– індекс кластера точки, μ_c– центр кластера.

Python: бібліотека та мінімальний приклад

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=7)
labels = kmeans.fit_predict(X)
```

6.2. Gaussian Mixture Models (GMM)

GMM моделює дані як суміш багатьох багатовимірних нормальних розподілів.

Переваги:

- М'яка (probabilistic) класифікація.
- Кластери можуть мати різну форму та орієнтацію.
- Дає оцінку належності до кожного кластеру.

Дані моделюються сумішшю багатовимірних нормальних розподілів:

$$p(x) = \sum_{k=1}^{K} \pi_k \ \mathcal{N}(\, x \mid \mu_k, \Sigma_k \,),$$

де K– кількість компонент (кластерів) у моделі; π_k– вагові коефіцієнти (мікси), що вказують, яку частку даних займає k-та компонента; μ_k– вектор середніх значень k-го кластеру;

Параметри знаходяться методом EM (Expectation-Maximization).

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3, covariance_type='full', random_state=7)
labels = gmm.fit_predict(X)
```

Узагальнені дані щодо вибору методу розпізнавання образів в електротехнічних системах наведено в табл. 1.

Таблиця 1 – Особливості вибору методу для енергетичних задач

Метод	**Переваги**	**Недоліки**	**Де застосовувати**
PCA	Зменшення розмірності, візуалізація, декореляція	Лише лінійні співвідношення	Попередній аналіз, виявлення патернів
SVM	Висока точність, стійкість	Повільний на великих наборах	Чітке розділення режимів
Decision Trees	Інтерпретованість	Перенавчання	Просте дерево логік
Random Forest	Стабільність, важливість ознак	Більше пам'яті	Кращий базовий класифікатор
CNN	Робота з сигналами/образами	Потребує більше даних	Осцилографічні сигнали, спектри
K-Means	Швидкість	Потрібно k	Попередня кластеризація
GMM	Гнучкі кластери	Важче підібрати параметри	P–Q траєкторії, аномалії

Розглянуті методи охоплюють повний спектр задач розпізнавання образів: від визначення структури даних (PCA) і кластеризації (K-Means, GMM) до побудови потужних класифікаторів (SVM, Random Forest, CNN).

В енергетичних системах, де режими мають складну фізичну динаміку, а рішення приймаються в реальному часі, ці методи забезпечують основу для:

- виявлення гойдань потужності,
- діагностики аварійних режимів,
- автоматизації релейного захисту,
- прогнозування нестійкості.

Короткі теоретичні відомості про очищення данних (Data Cleaning, Data Preprocessing)

Очищення даних (Data Cleaning, Data Preprocessing) є критичним етапом у побудові будь-якої системи розпізнавання образів.

Якість вхідних даних прямо визначає здатність моделі:

- коректно навчатися,
- узагальнювати закономірності,
- уникати хибних рішень (особливо важливо в задачах релейного захисту та автоматизації).

У необроблених даних часто присутні:

- пропуски,
- шум,
- викиди,
- некоректні одиниці виміру,
- дублікати,
- аномальні сплески,
- несумісні масштаби ознак.

Основна мета очищення даних – привести вибірку до якісного, впорядкованого, однорідного вигляду, який адекватно відображає фізичну природу процесів та дозволяє машинним моделям ефективно працювати.

Основні типи проблем у даних

Проблеми, що найчастіше зустрічаються у вимірювальних даних (особливо – у системах енергетики, зокрема PMU/DAQ сигналів):

1. Пропущені значення (Missing Values)

Причини:

- збій сенсора,
- втрати пакетів,
- неточності експорту,
- некоректне зчитування каналів.

Наслідки:

- недійсність статистичних оцінок,
- спотворення векторів ознак,
- помилки у навчанні моделей.

Робота з пропущеними значеннями
Стратегії обробки:
а) Видалення рядків із критично важливими пропусками
Наприклад, якщо відсутня label або record_id.
б) Імпутація числових ознак

- медіаною (робастно до викидів),
- середнім значенням (якщо розподіл рівномірний),
- інтерполяцією (для часових рядів).

в) Імпутація категоріальних ознак

- модою,
- або спеціальним тегом "__MISSING__".

г) Видалення колонок з надмірною кількістю пропусків

Типове правило: видалити колонку, якщо

$$\frac{\text{к-ть NaN}}{\text{к-ть рядків}} > 0.4$$

Такі ознаки зазвичай не несуть інформації.

2. Дублікати

Можуть виникати при:

- повторному експорті,
- об’єднанні датасетів,
- генерації синтетичних вікон.

Дублікати створюють зміщення в розподілах, що погіршує узагальнення моделей.

Усунення дублікатів

df = df.drop_duplicates()

Особливо важливо для синтетичних або агрегованих вибірок, щоб уникнути зміщення моделей.

3. Викиди (Outliers)

Типові джерела:

- різкі сплески під час перемикання,
- аперіодична складова під час КЗ,
- паразитні фронти через дискретизацію,
- збій АЦП або перешкоди.

Викиди можуть:

- перекручувати статистичні ознаки,
- бути помилково класифікованими як аварійні режими,
- різко погіршувати навчання моделей.

Обробка викидів. Підходи:

- обмеження інтервалу (clipping) за квантилями 1–99 %,
- заміна на медіанне значення в межах кластеру чи класу,
- лог-перетворення для сильно розтягнутих розподілів.

Приклад:

$$x' = \min(\max(x, Q_{0.01}), Q_{0.99})$$

4. Некоректні одиниці виміру

Найбільш критична проблема в енергосистемах:

- вимірювання у В, а записано у кВ,
- або у pu (per-unit),
- або після неправильного масштабування (наприклад, ділення/множення на 10 або 1000).

Приклади:

- напруги ~0.23 замість ~230 → запис у кВ;
- RMS ~1.0 → можливо система у pu.

Контроль одиниць виміру. Один із найважливіших етапів для енергетичних систем.

Контроль фазних RMS-напруг

У мережі 230 В очікуємо:

$$180 \leq V \leq 260,$$

якщо середнє ≈ 0.2–0.4 → ймовірно кВ; середнє ≈ 1.0 → можливо pu.

Контроль потоків потужності

$$P(t), Q(t)$$

мають відповідати характерному діапазону системи (для PMU в лінії високої напруги – МВт/Mvar).

Контроль симетричних складових

У нормальному режимі:

$$|V_1| \text{ велике}, |V_2| \approx 0, |V_0| \approx 0$$

У несинусоїдності/несиметрії:

$$|V_2| \uparrow, |V_0| \uparrow$$

5. Несумісні масштаби ознак

У ML-моделях це спричиняє:

- домінування великих за масштабом ознак,
- погіршення збіжності алгоритмів,
- перекіс вагових коефіцієнтів.

Тому необхідно проводити масштабування ознак та нормалізацію.

Через різні одиниці і діапазони ознак моделі потребують масштабування:

Standartization (Z-score) або Z-score стандартування (або нормалізація) – це перетворення змінної, при якому кожне значення xприводиться до безрозмірної величини, що показує, на скільки стандартних відхилень воно віддалене від середнього:

$$x' = \frac{x - \mu}{\sigma}$$

де x– початкове значення ознаки; μ– середнє значення (mean) цієї ознаки; σ– стандартне відхилення (standard deviation); x'– стандартизоване значення.

Min–Max Scaling (мін–макс нормалізація) – це спосіб масштабування даних, при якому кожне значення ознаки приводиться до діапазону [0,1] (або іншого заданого діапазону) шляхом лінійного перетворення:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

де x– початкове значення ознаки; $x_{\min}$– мінімальне значення ознаки в датасеті; $x_{\max}$– максимальне значення ознаки в датасеті; x'– нормалізоване значення у межах [0,1].

Для енергетичних задач Z-score зазвичай працює краще (робастність до великих амплітуд RMS).

6. Невідповідність очікуваній фізиці

Приклади:

- RMS напруг виходить поза діапазоном 180–260 В у мережах 230 В,
- імпеданс Z_est не узгоджується з режимом,
- позитивна послідовність V1 некоректно мала у нормальному режимі,
- високі значення THD, які не відповідають реальній системі.

При очищенні даних можуть застосовуватись візуальні методи первинного аналізу, зокрема:

Гістограми основних ознак

Дозволяють побачити:

- аномальні хвости,
- двомодальні розподіли (ознака змішування режимів),
- проблеми масштабу.

Теплова карта кореляцій (Correlation Heatmap)

Виявляє надлишкові ознаки.
Якщо

$$| r(\text{feature}_i, \text{feature}_j) | > 0.95,$$

то одна з ознак може бути видалена без втрати інформації.
У енергетиці часто:

- Vrms_a, Vrms_b, Vrms_c сильно корельовані → достатньо однієї;
- P_mean і V1_mag*I1_mag можуть дублювати інформацію про потужність;
- симетричні складові V0,V1,V2 рідко корельовані між собою – і це добре.

Для сучасних моделей розпізнавання образів (SVM, RandomForest, Boosting, Neural Nets) якість даних визначає:

- стабільність меж прийняття рішень,
- здатність моделі розрізняти режими,
- захист від хибних спрацювань,

- точність у рідкісних класах (fault).

Недостатньо очищені дані можуть призвести до:

- помилок типу false positive (помилкові аварії),
- false negative (пропущені аварії),
- нестійких моделей, що змінюють рішення при малих змінах сигналу.

Очищення даних – це не технічна рутина, а обов'язковий елемент інженерного мислення, який:

- гарантує відповідність вимірів фізичним законам,
- захищає моделі розпізнавання образів від помилок,
- підвищує точність і надійність,
- є фундаментом для подальших етапів – кластеризації, класифікації, побудови логік РЗіА.

У задачах енергосистем, де помилки можуть призвести до хибних відключень або недовідключень, коректне очищення даних має першочергове значення.

Приклад виконання розрахункової роботи

Формулювання. Задача 1. Виявлення симетричних несправностей під час гойдання потужності

У реальних системах більшість подій – від нормальної роботи до аварій можна описати через зміни у часових рядах трифазних напруг і струмів, які вимірюються приладами класу PMU, цифровими захистами або реєстраторами аварійних процесів. Саме тому методи розпізнавання образів (кластеризація, класифікація, виявлення аномалій, глибинне навчання) стають ключовими інструментами інженерів з релейного захисту, автоматизації та експлуатації електроенергетичного обладнання.

Відповідно до завдань прописаних для цього типу задач в можливих варіантах задач до розрахункової роботи необхідно класифікувати режим електроенергетичної системи:

- нормальний режим,
- гойдання потужності (Power Swing),
- симетричні короткі замикання (three-phase fault).

та визначити, чи можна розмежувати гойдання потужност і КЗ за певними ознаками.

Постановка задачі та короткі теоретичні відомості про тематику дослідження

Загальні відомості про трифазні електроенергетичні сигнали

Сигнали напруги та струму в електроенергетичних системах високої напруги мають переважно синусоїдальну форму з частотою 50 Гц. У нормальних умовах система є симетричною, тобто амплітуди фаз однакові, а фазові зсуви становлять ±120°.

Трифазні напруги описуються рівняннями:

$$v_a(t) = V_m \sin(\omega t), v_b(t) = V_m \sin(\omega t - 120°), v_c(t) = V_m \sin(\omega t + 120°),$$

де V_m– амплітуда, $\omega = 2\pi f$, $f = 50$ Гц.

У реальних системах відхилення від ідеальної синусоїди спричинюють:

- зміни навантаження,
- гойдання потужності,
- короткі замикання,
- коливання частоти,
- несиметрії та гармоніки.

Тому методи розпізнавання образів дозволяють аналізувати сигнали, класифікувати режими та виявляти аварії.

Поняття фазора (комплексної амплітуди)

Фазор – це комплексне представлення основної гармоніки сигналу змінного струму. Якщо сигнал має вигляд:

$$x(t) = X_m \sin(\omega t + \varphi),$$

де φ – фаза сигналу, Xm – це амплітуда сигналу, тобто максимальне значення, якого досягає синусоїда, ω – кутова частота (рад/с), t – час (с)

то його фазор (тобто комплексна амплітуда) дорівнює:

$$\tilde{X} = X_{\text{RMS}} e^{j\varphi}, X_{\text{RMS}} = \frac{X_m}{\sqrt{2}}.$$

У цифровій формі фазор визначають через дискретне перетворення Фур'є (DFT) на частоті 50 Гц:

$$\tilde{X} = \frac{2}{N} \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi f n / f_s}.$$

де N – кількість відліків, що беруться для розрахунку (для 1 період $N=f_s/50$); f_s – частота дискретизації (Гц); $xa[n]$ –дискретні вибірки миттєвого значення у фазі a, отримані АЦП; n=0,1,2,…,N−1 – номер вибірки; $e^{-j2\pi fs/fn}$ – це комплексна експонента для еталонного сигналу на частоті f = 50 Гц.

Фазори застосовують для:

- визначення симетричних складових;
- розрахунку позитивної та негативної послідовностей;
- побудови алгоритмів релейного захисту;
- ідентифікації аварійних режимів.

При порівнянні фазорів (50 Гц) – напруги та струми для трьох режимів помітно, що в нормальному три вектори напруг рівномірно рознесені на 120°, струми відстають на кут φ, при гойданні – амплітуди фазорів змінені (усереднені), але геометрія близька до збалансованої, а для 3-фазному КЗ, модуль фазорів напруг суттєво менший, струми – більші (вказує на низький імпеданс у місці аварії).

Середньоквадратичне значення (RMS)

RMS (Root Mean Square) є базовою мірою амплітуди змінного сигналу. RMS (середньоквадратичне значення) – це статистична міра амплітуди змінного

сигналу, яка показує еквівалентне постійне (DC) значення, що виділяло б ту саму середню потужність на резистивному навантаженні, що й сам змінний сигнал.

Аналітичне означення:

$$X_{\text{RMS}} = \sqrt{\frac{1}{T}\int_0^T x^2\,(t)\,dt}.$$

де T – період сигналу; $x^2(t)$ – моментна потужність (якби через резистор 1 Ом)

Дискретне означення:

$$X_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} x^2\,[n]}.$$

У трифазній системі часто використовують RMS для напруг і струмів:

$$V_{a,\text{RMS}},\ V_{b,\text{RMS}},\ V_{c,\text{RMS}}, I_{a,\text{RMS}},\ I_{b,\text{RMS}},\ I_{c,\text{RMS}}$$

RMS значення використовують для:

- контролю симетрії фаз;
- визначення глибини падіння напруги при КЗ;
- аналізу амплітудної модуляції під час гойдань;
- нормування ознак для класифікації.

Симетричні складові (метод Фортеск'ю)

Метод симетричних складових є стандартом у релейному захисті та аналізі аварій. Будь-яку трифазну систему можна представити як суму трьох симетричних систем:

1. Пряма послідовність (positive sequence) – ідеальна, збалансована система, що відповідає нормальному режиму:

$$V_a = V_1, V_b = a^2 V_1, V_c = aV_1,$$

де $a = e^{j120^\circ}$.

Вона несе інформацію про нормальний режим (фаза B відстає на 120°, C – на 240°). При трифазному КЗ струм прямої послідовності великий, а напруга падає до ≈ 0.

2. Нульова послідовність (zero sequence) – виникає при замиканні на землю або несиметрії нульової точки. Усі три фази в однаковій фазі і з однаковою амплітудою:

$$V_a = V_0, V_b = V_0, V_c = V_0.$$

Може існувати лише тоді, коли є шлях через землю або нейтраль, а саме при однофазному КЗ на землю, двофазному КЗ на землю, пробою ізоляції, з'єднання з виведеною нейтраллю та відповідає струму замикання на землю.

3. Зворотна послідовність (negative sequence) – індикатор несиметрії, виникає при однофазних і двофазних КЗ, або перекосах навантаження.

Це збалансована система, але з оберненим порядком фаз:

$$V_a = V_2, V_b = aV_2, V_c = a^2 V_2.$$

Зазвичай дорівнює нулю в нормальному режимі. Виникає при однофазних та двофазних КЗ, перекосах навантаження, неправильній роботі обладнання (асиметрія трансформаторів, порушення контактів). Вона є дуже важлива для релейного захисту реле зворотної послідовності реагують на механічний нагрів двигунів від несиметрії.

Характеристика режимів

У ***нормальному режимі*** трифазна електрична система працює за умовами:

- збалансованого навантаження,
- синхронної роботи генераторів,
- стабільної частоти 50 Гц,
- сталих значень активної та реактивної потужності.

Напруга та струм у кожній фазі мають майже ідеальну синусоїдальну форму:

$$v_a(t) = V_m \sin(\omega t), v_b(t) = V_m \sin(\omega t - 120°), v_c(t) = V_m \sin(\omega t + 120°)$$

Струм відстає від напруги відповідно до характеру навантаження (при індуктивному – на кут φ).

Основні ознаки нормального режиму:

- компенсована трифазна система (негативна послідовність ≈ 0);
- стабільні RMS значення;
- стабільна частота;
- низькі коливання потужності;
- відсутність аперіодичних складових у струмі.

Це базовий стан, який використовується для навчання та порівняння з перехідними процесами.

Гойдання потужності (Power Swing) – це повільні коливання активної та реактивної потужності між вузлами електричної мережі. Вони виникають у ситуаціях:

- після короткого замикання, яке було успішно полодолане;
- при втраті синхронізму між генераторами;
- під час зміни конфігурації мережі та великих перетоків потужності;
- в «слабких» мережах під дією великих навантажень.

Фізично гойдання пов’язане зі зміною кута δ між ЕРС синхронних машин, що викликає модуляцію амплітуди та частоти напруги, а також хвилеподібні зміни потужності.

Прояв у сигналах:

- амплітудна модуляція напруг (періодичне «дихання» RMS);
- частотна модуляція (миттєва частота змінюється відносно 50 Гц);
- зміна положення фазорів у часі;
- чіткі коливальні процеси у кривих P(t), Q(t).

Найважливіше те, що power swing може бути помилково сприйнятим захистом як коротке замикання, тому сучасні функції захисту використовують алгоритми розпізнавання образів для коректного розмежування цих режимів.

Симетричне (трифазне) коротке замикання є одним із найважчих аварійних режимів, адже всі три фази одночасно замикаються на низький імпеданс.

Це спричиняє:

- різке падіння напруги у місці аварії (до 10–40 % від номіналу),
- різке збільшення струмів (у 2–10 разів),
- появу аперіодичної DC-складової у струмі (ефект намагнічування),
- зміну потоків потужності у мережі.

Фізичні ознаки 3-фазного КЗ:

- миттєве зниження напруг V_a, V_b, V_c;
- різке зростання I_a, I_b, I_c;
- поява компонентів, що не є синусоїдами (див. DC offset);
- збільшення дисперсії активної потужності P;
- суттєве порушення рівноваги між генераторами;
- зсув фазорів у бік навантаження;
- RMS напруг різко падає на ділянці КЗ;
- RMS струмів різко зростає в цей же період;
- $P_{\_mean}$ різко зростає або падає, $P_{\_std}$ дуже високий;
- Тривалість аварії обмежена інтервалом [$t_{\_fault}$; $t_{\_clear}$].

На відміну від гойдань, тривалість КЗ дуже коротка, і після його зникнення система переходить у новий перехідний стан.

У системах релейного захисту та автоматики критично важливо розрізняти де нормальний режим, який не потребує дій; де гойдання потужності, під час якого захист не повинен помилково відімкнути лінію; де трифазне коротке замикання, яке потребує негайного відключення аварійної ділянки.

Помилка у класифікації може призвести до:

- каскадних відключень,
- втрати стійкості,
- пошкодження генераторів і трансформаторів,
- великих економічних збитків.

Методи розпізнавання образів дозволяють автоматизувати цей процес, використовуючи:

- часові ряди сигналів,
- фазорні характеристики,
- симетричні складові,
- статистичні ознаки,
- алгоритми машинного навчання.

Усі етапи виконуються на основі датасету, що містить три типи режимів: normal, powerswing, fault_3ph.

Ключові ознаки кожного із режимів наведено в таблиці 2.

Таблиця 2 – Порівняння ознак, що визначають наявність певного класу режимів роботи трифазної електричної системи

Ознака	**normal**	**powerswing**	**fault_3ph**
Зміна RMS напруги	дуже мала	плавна	різке падіння
Зміна RMS струмів	мала	помірна	різке зростання ×2...5
env_std	низький	високий	різко високий під час КЗ
DC-компонента	немає	немає	є, затухає
Негативна послідовність	≈0	≈0	≈0 (бо симетричне КЗ)
Позитивна послідовність	стабільна	коливається	падає при КЗ
Частота	стабільна 50 Гц	«плаває»	стабільна/збита під час КЗ

Загальний алгоритм виконання роботи по практичним етапам

Роботу можна розділити на такі наступні етапи:

1. Ознайомлення з даними
    - завантаження сигналів з signals.csv;
    - перегляд структури датасету, кількості вибірок та міток.
2. Очищення даних
    - Виведення базової статистики;
    - Видалити рядки без обов'язкових полів
    - Видалити колонки з >40% пропусків
    - Імпутація: числові → медіана; категоріальні → мода / тег
3. Візуальний аналіз режимів
    - обчислення ковзних RMS;
    - побудова графіків напруг та струмів;
    - спостереження коливань напруги при гойданні;
    - ідентифікація стрибка струму та падіння напруги під час КЗ.
4. Попередня обробка сигналів
    - нормалізація;
    - згладжування;
    - визначення інтервалів аномалій.
5. Виділення ознак (feature engineering)
    - розрахунок RMS;
    - симетричних складових;
    - активної та реактивної потужності;
    - оцінка імпедансу;
    - метрики нестабільності (дисперсія огинаючої, P_std).
6. Формування вибірки для моделі
    - робота з features.csv;
    - очищення та нормування ознак;
    - виділення train/test наборів.
7. Застосування методів розпізнавання образів
    - побудова класифікаторів (SVM, Logistic Regression**,** Random Forest, k-NN, нейронні мережі);
    - кластеризація (k-Means, DBSCAN, PCA);

- побудова меж прийняття рішень.

8. Оцінювання якості моделей
   - матриця невідповідностей (confusion matrix);
   - точність (accuracy), повнота (recall), F1-міра;
   - візуалізація кластерів або проєкцій PCA.
9. Інтерпретація результатів
   - фізичний зміст отриманих кластерів або класів;
   - порівняння режимів за ознаками V_1, V_2, RMS, P(t).

Опис етапів проведення розпізнавання режимів

Виконуємо підготовку даних по кожному із можливих режимів. Для Normal генеруються для по синусоїдам 50 Гц з фазовими зсувами ±120°, малий вимірювальний шум. Для Power swing характерним є при генерації миттєва частота $f(t) = 50 + \Delta f \sin(2\pi f_{sw} t)$, інтегрується в фазу; амплітудна модуляція для Va/Vb/Vc; струми формуються як реакція навантаження $Z = R + jX$з фазовим зсувом (індуктивний характер). Для 3-фазного КЗ у вікні $[t_{fault}, t_{clear}]$ напруга масштабується до 0.1–0.4 pu; струм зростає у k=2…5 разів + додається експоненційний DC-зсув $I_{dc}(t) = I_0 e^{-(t-t_{fault})/tau}$. Формується файл у форматі csv (рекомендовано застосовувати такий тип файлів) signals.csv, який містить часову мітку t, напруги трьох фаз: Va, Vb, Vc, струми трьох фаз: Ia, Ib, Ic, мітку класу label.

Приклад файлу:

record_id, t, Va, Vb, Vc, Ia, Ib, Ic, label
rec_0001, 0.000, ..., ..., ..., ..., ..., ..., normal
rec_0001, 0.001, ..., ..., ..., ..., ..., ..., normal
...
rec_0245, 0.999, ..., ..., ..., ..., ..., ..., powerswing

На даному етапі було проведено початкове очищення даних. Розпочинаємо із огляду даних в Jupiter Notebook, а саме виведення кількох вибіркових рядків; перевірка, скільки вікон у кожному класі; побудова базових графіків (line plot) для однієї з фаз.

Необхідно зчитати файли з директорії за допомогою pandas.read_csv():

Python скипт

```
import pandas as pd
df = pd.read_csv("dataset_ps_fault/ signals.csv ")
```

Перевірити

- розмірність: *df.shape*
- перші рядки: *df.head()*
- типи даних: *df.info()*
- наявність обов'язкових колонок: label, record_id

На третьому етапі необхідно провести очищення даних, що включає в себе видалення непридатних даних, імпутацію пропусків, візуальний аналіз розподілів.

Визначаємо кількість пропусків:

Python скипт

```
df.isna().sum().sort_values(ascending=False)
```

Перевіряємо наявність дублікованих рядків:

Python скипт

```
df.duplicated().sum()
```

Далі проводимо видалення непридатних даних:

- Видалення рядків з критичними пропусками. Не можна будувати модель, якщо відсутні label або record_id.

Python скипт

```
df = df.dropna(subset=["label", "record_id"])
```

- Видалити колонки з надмірною кількістю пропусків. Видалення колонок з >40% пропусків: поріг можна змінити (наприклад, 30% або 50%), залежно від задач дослідження

Python скипт

```
na_ratio = df.isna().mean()
df = df.drop(columns=na_ratio[na_ratio > 0.40].index)
```

Здійснюємо імпутацію пропусків в певних рядках. Зокрема, в числових ознаках пропущені дані заміняємо медіанним значення, робастно до викидів (при відсутності викидів в даних можна використатати і середньоквадратичне значення, проте заміна медіанним значенням є більш універсальним варіантом).

Python скипт

```
num_cols = df.select_dtypes(include=['number']).columns
for col in num_cols:
df[col].fillna(df[col].median(), inplace=True)
```

Пропуски в категоріальних ознаках замінюємо модою (або спеціальний тег, якщо всі пропуски).

Python скипт

```
cat_cols = df.select_dtypes(exclude=['number']).columns
for col in cat_cols:
df[col].fillna(df[col].mode().iloc[0], inplace=True)
```

Третім етапом проводиться візуальний аналіз режимів. Будуємо

- часові форми напруг і струмів;
- ковзний RMS:

$$X_{\mathrm{RMS}}[n] = \sqrt{\frac{1}{W}\sum_{k=n-W}^{n} x^2\,[k]}$$

де x[k] – Вхідний сигнал, що відповідає дискретним значенням напруги або струму, зняті з кроком дискретизації; n – Поточний індекс часу, що вказує на момент часу, для якого рахується RMS; W – кількість відліків, які враховуються при обчисленні ковзного RMS.

- активну потужність:

$$P(t) = v_a(t)i_a(t) + v_b(t)i_b(t) + v_c(t)i_c(t)$$

Результати очікуваного спостереження наведені в таблиці 3.

Таблиця 3 – Особливості графіку щодо визначення режиму

Режим	Ознаки у графіку
Normal	синусоїди стабільної амплітуди
Power swing	«дихання» амплітуди, коливання P(t)
Fault_3ph	різке падіння напруги та стрибок струму

Наступним етапом визначаємо та розраховуємо ознаки, які є ключем до класифікації.

1) RMS-фічі

$$V_{a,\mathrm{RMS}}, V_{b,\mathrm{RMS}}, V_{c,\mathrm{RMS}}, I_{a,\mathrm{RMS}}, I_{b,\mathrm{RMS}}, I_{c,\mathrm{RMS}}$$

2) Фазори (50 Гц)

$$\tilde{V}_a = \frac{2}{N}\sum_{n=0}^{N-1} Va[n]\, e^{-j2\pi f n/f_s},$$

де N – кількість відліків, що беруться для розрахунку (для 1 період $N=f_s/50$); f_s – частота дискретизації (Гц); $Va[n]$ –дискретні вибірки миттєвого значення напруги у фазі *a*, отримані АЦП; n=0,1,2,…,N−1 – номер вибірки; $e^{-j2\pi fs/fn}$ – це комплексна експонента для еталонного сигналу на частоті f = 50 Гц.

3) Симетричні складові

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & a^2 \\ 1 & a^2 & a \end{bmatrix}\begin{bmatrix} \tilde{V}_a \\ \tilde{V}_b \\ \tilde{V}_c \end{bmatrix}$$

Інтерпретація, якщо V_1– основа нормального режиму, V_2– індикатор несиметрії (≈0 у гойданні та трифазному КЗ), то під час трифазного КЗ напруга прямої послідовності (positive-sequence voltage) падає ($V_1 \downarrow$). Компонента прямої послідовності – це те, що відповідає за нормальну симетричну роботу мережі. Коли в точці КЗ відбувається повне коротке замикання нормального режиму там більше не існує, тому і напруга прямої послідовності зникає.

4) Активна потужність

$$P = Va \cdot Ia + \bar{Vb} \cdot Ib + Vc \cdot Ic,$$

де Va, Vb, Vc – це комплексні фазні напруги у фазах a, b і c; Ia, Ib, Ic – це комплексні фазні струми у відповідних фазах

5) Можуть бути розраховані статистичні ознаки:

- стандартне відхилення RMS,

- коефіцієнт варіації,
- мінливість огинаючої (env_std).

Четвертим етапом проведемо візуальний аналіз розподілів та попередню обробку сигналів. Візуалізація дозволяє побачити зміщені або двомодальні розподіли, викиди, дивні масштаби.

Рекомендованими гістограми ключових ознак приймаємо Vrms_*, Irms_*, V1_mag, env_std, P_mean, Z_est.

Python скипт

df[[«Vrms_a»,»Vrms_b»,»Vrms_c»]].hist(bins=30, figsize=(9,4))

Інтерпретація результатів:

Нормальні дані → одновершинні розподіли.

Багато сплесків → шум / аномальні вікна.

Два «горби» → змішування різних режимів або різних одиниць.

Також в ході виконання роботи було проведено вибір ознак (**feature engineering** або **feature selection)** на п’ятому етапі. Було вибрано наступні знаки:

- Vrms_a/b/c, Irms_a/b/c – фазні RMS.
- P_mean, P_std – середня активна потужність і розкид.
- V1_mag, V2_mag, I1_mag, I2_mag – позитивна/негативна послідовності (за фазирними оцінками 50 Гц).
- V2_over_V1, I2_over_I1 – відносні індикатори несиметрії (для 3-фазного КЗ очікувано малі).
- env_std – дисперсія огинаючої сумарної напруги (збільшується при гойданнях).
- Z_est – оцінка модуля імпедансу мережі.

Для кінцевого форулювання переліку ознак для машинного навчання необхідно здійснити перевірку кореляцій та виявлення надлишкових ознак. Висока кореляція між ознаками ($|r| > 0.95$) означає, що вони дублюють інформацію.

Python скрипт побудови кореляційної матриці:

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), cmap="coolwarm", vmin=-1, vmax=1)

Типові сильні кореляції, що були отримані за результатами кореляційного аналізу:

- Vrms_a, Vrms_b, Vrms_c
- I1_mag та P_mean
- V1_mag та Z_est (частково)

Рекомендації:

- залишати 1 із сильних корельованих фіч (переважно – фізично найосмисленішу),
- зменшити мультиколінеарність перед навчанням моделей.

Також на етапі feature engineering проводимо контроль одиниць виміру на фізичну адекватність даних. Це є критично важливо для енергосистем. Зокрема перевіряємо RMS напругу. У мережах 230 В реалістичний діапазон:

$180 \leq V\_"RMS" \leq 260$

Python скипт

```
for col in ["Vrms_a", "Vrms_b", "Vrms_c"]:
if col in df.columns:
print(col, df[col].quantile([0.01, 0.5, 0.99]))
```

Звертаємо увагу на можливу наявність неправильних одиниць:

середнє ≈ 0.23 → це ймовірно кВ

середнє ≈ 1.0 → це може бути pu

межі 20–40 → можливий масштабний множник (×10)

та перевірку симетричних складових. У нормальному режимі має виконуватись: $|V\_2| \approx 0, |V\_0| \approx 0$.

Дуже великі V2_mag або I2_mag → ознака дефектних вимірів.

Якщо на попередньому етапі були помітні дивні масштаби, то для коректної роботи ML-алгоритмів (особливо SVM, LogisticRegression, нейронних мереж) потрібно масштабувати ознаки. Це робиться тому, що ознаки мають різний масштаб: Vrms ~ 200–300, P_std ~ 10^4, Z_med ~ 5–20, V2_mag ~ 0.1–5. Без масштабування SVM і k-NN працювали б некоректно – найбільші за масштабом ознаки перекривали б інші. Навіть Random Forest, який менш чутливий, все одно виграє від масштабу в плані стабільності та швидкості.

Найкращий метод це StandardScaler (Z-score).

Python скипт

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_numeric)
```

На наступних етапах проведено побудову моделей машинного навчання. В роботі було виконано побудову моделей класифікації та кластеризації.

Класифікація за ознаками

1. Вибрано підмножину ознак:
   - Vrms_a, Irms_a, P_mean, env_std, V1_mag, V2_mag, Z_est.
2. Розбито вибірку на тренувальну та тестову (train/test (75/25)).

Python скрипт

```
from sklearn.model_selection import train_test_split
X = features.drop(columns=['label'])
y = features['label']
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.25, random_state=7, stratify=y
)
```

3. Навчено моделі класифікації логістична регресія (Logistic Regression), SVM (метод опорних векторів), Random Forest (випадкового лісу) та k-NN (k-найближчих сусідів).

Класифікатор **SVM (RBF kernel)** добре працює на нелінійно роздільних даних, здатен відділяти PS та Fault за криволінійною межею рішень. SVM з RBF-ядром фактично будує нелінійну поверхню рішень, яка розділяє три класи (one-vs-rest схема).

Для сформованої в дослідженні задачі цей метод підходить, тому що:

- Power Swing утворює криволінійну траєкторію в просторі ознак (зміни імпедансу + зміна дисперсії потужності).
- 3-phase fault розташований більш компактно, низьке Z_mag та високий Irms.
- Normal – ще компактніше, з низькою нестабільністю.

Python скрипт

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
svm_model = Pipeline([
  ("scaler", StandardScaler()),
   ("clf", SVC(kernel='rbf', probability=True))
  ])
  svm_model.fit(X_train, y_train)
```

Перевагами використання SVM у задачі класифікації режимів є те, що метод чудово розрізняє Power Swing vs Fault, навіть якщо їхні ознаки перетинаються; здатний працювати з невеликою кількістю ознак; генерує “гладкі” межі, які добре узагальнюють дані.

Слабкими сторонами є те, що метод потребує масштабування (саме тому це було проведено на попередньому етапі); час навчання росте з кількістю даних (це у випадку аналізу значного за тривалістю часового ряду); важко інтерпретується у фізичному сенсі (межа рішень абстрактна).

У даній роботі також навчалась **логістична регресія (Logistic Regression).** В результаті дослідження було помітно, що модель швидко навчається на компактному наборі ознак (Z_med, P_std, Irms, V1_mag, V2_mag); дає інтерпретовані коефіцієнти, що дозволяють зрозуміти фізичні впливи параметрів; видає коректно калібровані ймовірності, корисні для подальших рішень (наприклад, підключення порогів, режим "trip/block"); показує хорошу точність у нормальному режимі, але має гіршу роботу на Power Swing, коли ознаки не повністю відділяють класи лінійно.

Перевагами застосування логістичної регресії є:

1. Висока інтерпретованість моделі. LR дає можливість легко оцінити вплив кожної ознаки:

- позитивний коефіцієнт → ознака підвищує ймовірність класу,
- негативний → знижує,
- нульовий → мало впливає.

У задачі PS/Fault це робить LR цінною, оскільки можна інтерпретувати важливість:

- Z_med (імпеданс падає при КЗ, що спричинює сильний негативний вплив на «normal» режим),

- P_std (нестабільність зростає при гойданні),
- V2_mag (зростає при несиметріях і PS),
- Irms (високий при Fault).

2. Надає гладкі, лінійні межі рішень. LR створює *лінійно розділяючі поверхні*, що є перевагою, коли:

- дані добре розділяються лінійно у просторі ознак,
- або ознаки вже добре підібрані (як у нашій задачі).

3. Низька складність і висока швидкість

Метод надзвичайно швидко тренується навіть на великих вибірках.

4. Вихідні ймовірності – калібровані та стабільні

LR зазвичай краще калібрує ймовірності, ніж SVM або Random Forest. Це дуже важливо для релеїв/захисних алгоритмів, де потрібна впевненість моделі у рішенні.

Недоліки логістичної регресії:

1. Погано працює на нелінійно роздільних даних. У випадку, якщо ознаки не забезпечують лінійну віддільність між PS і Fault, то LR може плутатися.

2. Чутливість до масштабування. Потрібно обов'язково масштабувати ознаки через StandardScaler.

3. Погано враховує складні взаємодії ознак. На відміну від Random Forest або Gradient Boosting, LR не «бачить» нелінійностей.

4. Слабка на даних із значним шумом або сильно корельованими ознаками. Потрібно застосовувати регуляризацію (L2 або L1).

Python скрипт

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(max_iter=500, multi_class='multinomial')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

Random Forest навчається як ансамбль рішень на основі дерева. Кожне дерево бачить випадкову підмножину ознак і вибірок. Фінальне рішення за голосуванням. Метод випадкового лісу добре підходить для енергетичних задач оскільки є стійким до шуму, дає feature importance та природно враховує взаємодію ознак (наприклад, в даній задачі комбінацію Z_med + P_std).

Python скрипт

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=300, random_state=7)
rf.fit(X_train, y_train)
```

Random Forest в даній задачі показує дуже високу Accuracy (іноді найвищу серед усіх трьох), високий Recall для fault_3ph, тому що ознаки для КЗ дуже специфічні, стабільну роботу навіть на трохи "брудних" даних.

Тобто сильними сторонами RF для задачі класифікації режимів є інтерпретованість (важливість ознак), стійкість, мінімальні вимоги до масштабування/гіпертюнінгу.

Однак слід виділити, що межі рішень можуть бути "рвані", можуть перенавчатися на малих вибірках і випадковий ліс не працює з out-of-distribution значеннями так гладко, як SVM.

Метод **k-NN** місить в собі побудову простого baseline (базової лінії), добре працює, коли кластери добре відокремлені геометрично та дає досліднику інтуїтивне розуміння геометрії даних

Python скрипт

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)

Висновки, щодо застосування методів класифікації:

SVM дає плавні нелінійні межі – добре для визначення PS vs Fault режимів.

RF дає хорошу точність і надає важливість ознак (Z_med, V2).

k-NN – простий baseline: працює лише якщо PCA уже показала компактні кластери.

Варіант B: Кластеризація

Застосування **PCA (Principal Component Analysis)** для зменшення розмірності. Перед застосуванням PCA усі ознаки (Vrms, Irms, V1_mag, V2_mag, P_mean, P_std, Z_med) були нормовані за допомогою StandardScaler. Це критично важливо, тому що PCA чутливий до масштабів ознак:

- ознаки з великою дисперсією можуть домінувати над іншими,
- нормування гарантує справедливий вклад кожної фічі в компоненти.

При реалізації кластеризації методом PCA було обчислено дві головні компоненти:

Python скрипт

pca = PCA(n_components=2, random_state=7)

Xp = pca.fit_transform(X_scaled)

Отримані компоненти (PC1 та PC2):

- лінійні комбінації початкових ознак,
- максимально добре відображають структуру варіації даних,
- дозволяють побачити приховану геометрію класів.

Потім було виведена пояснена дисперсія (explained variance ratio).

Python скрипт

*pca.explained_variance_ratio_*

Цей вектор вказує на частку інформації, яку зберігає кожна з компонент, наскільки «інформативною» є PCA-проєкція.

Також було зроблено 2D-проєкція та візуалізація класів. Було побудовано scatter-графік з падінгом класів у різні кольори:

- синій – normal
- помаранчевий – powerswing
- червоний – fault_3ph

Цей графік дозволяє:

✓ оцінити, чи утворюють класи чіткі кластери

- Fault зазвичай концентрується в одній зоні, оскільки Z_med сильно падає при КЗ.
- Power Swing утворює витягнуту хмару через зміну амплітуд і імпедансу.
- Normal – найбільш компактний.

✓ виявити потенційні перекриття класів

Це дозволяє досліднику зрозуміти, як добре ознаки відокремлюють режими навіть до тренування класифікаторів.

Основні висновки які можна сформулювати

- PC1 найчастіше відповідає за зміни імпедансу та RMS (тобто відображає Fault vs Power Swing).
- PC2 часто відображає енергетичні/нестабільні складові (P_std, V2_mag), що допомагає фіксувати динамічність Power Swing.

Якщо PS і Fault перекриваються → ознаки потрібно покращити.

- провести кластеризацію (k-means або DBSCAN)

Метод k-Means шукає сферичні кластери. Центроїди приблизно відповідають трьом «режимам»:

- Normal
- Power Swing
- Fault
- простий baseline

Метод добре працює, коли кластери добре відокремлені геометрично та дає дослідникам інтуїтивне розуміння геометрії даних. Якщо k-Means відтворює три кластери, схожі на справжні мітки, то це означає, що ознаки добре описують фізику режимів.

Для k-Means також проводиться попередня нормалізація:

Python скрипт

X_scaled = StandardScaler().fit_transform(X)

Потім відбувалось виконання трьох «режимів»:

km = KMeans(n_clusters=3)

km_labels = km.fit_predict(X_scaled)

Метод DBSCAN застосовується для щільності алгоритму, виявляє аномальні точки та граничні режими. Він здатен виділити Fault як «щільний кластер», а Power Swing – як «витягнуту структуру»

Виконання за Python скриптом:

db = DBSCAN(eps=0.8, min_samples=10)

db_labels = db.fit_predict(X_scaled)

Як результат DBSCAN дозволив знайти спорадичні або проміжні режими, навіть без міток. Оскільки Fault точки виділяються у кластер автоматично – ознаки добре відображають фізичні особливості аварій. Відображення класів:

Fault → компактний щільний кластер;

Normal → також щільний, але окремий;

Power Swing → може бути розтягнутим ланцюжком точок;

DBSCAN позначає «викиди» як -1 (аналог аномалій).

В роботі було проведено і оцінку якості моделей. В даній роботі оцінка проводилась безпосередньо після застосування методів. Оцінка якості грунтується на матриці невідповідностей (confusion matrix):

$$\text{CM} = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

та розраховує: accuracy, precision, recall, F1-score. В більшості випадків був застосований єдиний вивід метрик через print_classification_metrics(...), що включає accuracy в % і дробом, F1-macro, OVR-AUC, та classification_report.

Використані python скриптів для кожної з моделі:

Метод **k-NN**

```
print_classification_metrics(y_test,y_pred,y_proba=y_proba,labels=labels,title='k-NN (k=7)',average='macro')
print(accuracy_score(y_test, y_pred))
```

Random Forest

```
print_classification_metrics(y_test,y_pred,y_proba=y_proba,labels=labels,title='Random Forest (n=300)',average='macro')
print(accuracy_score(y_test, y_pred))
```

SVM

```
print_classification_metrics (
 y_test, y_pred, y_proba=y_proba, labels=labels, title='SVM (RBF)',
average='macro')
```

LogisticRegression (для класифікації)

```
best_C = 3
log_reg = LogisticRegression(
   C=best_C, class_weight="balanced",
   solver='lbfgs', multi_class='multinomial',
   max_iter=2000
).fit(X_poly_train, y_train)
```

Також в рботі було проведено аналіз моделей на основі імовірнісних метрик: Brier та RMSE(prob).

$MSE = mean((one\_hot(y\_true) - predict\_proba(X))^2)$

$RMSE = sqrt(MSE)$

Ці метрики дозволяють оцінити каліброваність моделі, що є актуальною для захисних пристроїв, де важлива впевненість моделі.

Результат аналізу показує, що було правильно підібрані фізичні ознаки (Z_med, V2_mag, RMS, P_std) – вони автоматично забезпечують роздільність режимів. Різні моделі дивляться на ці ознаки по-своєму, але якщо ознаки якісні, всі вони показують високі метрики. Power Swing і Fault можна розрізнити, якщо правильно враховувати імпеданс і нестабільність потужності.

За результатами дослідження можна зробити натупні висновки по роботі:

- SVM дає плавні нелінійні межі – добре для PS vs Fault.

- RF дає хорошу точність і виділяє важливі ознаки (Z_med, V2).
- k-NN – це модель простої базової лінійної залежності і працює лише якщо PCA уже показала компактні кластери.

Контрольні запитання до розрахунково-графічної роботи

1. Які ознаки були обрані для класифікації режимів та чому саме вони є інформативними?

2. Які ознаки виявились найважливішими для моделі (за feature importance або аналізом коефіцієнтів)?

3. Які види проблем у вимірювальних даних ви виявили (пропуски, викиди, шум, неправильні одиниці виміру) та як саме усували кожну з них?

4. Які ознаки (фічі) є найбільш інформативними при класифікації режимів електроенергетичної системи? Наведіть приклади.

5. Чому нормалізація/масштабування ознак є необхідною перед застосуванням методів машинного навчання?

6. Які моделі ви використали (SVM, Logistic Regression, Random Forest, k-NN тощо) та чим обґрунтували вибір кожної?

7. Чому PCA може покращити роздільність класів або допомогти візуалізувати структуру даних?

8. У яких випадках метод PCA є ефективним, і яку функцію він виконує у задачах аналізу енергетичних режимів?

9. Як ви інтерпретували результати PCA-візуалізації? Чи були випадки, коли класи накладалися один на одного, і як це вплинуло на роботу моделі?

10. Які проблеми з даними найчастіше виникають у вимірювальних вибірках електроенергетичних систем, та які методи очищення рекомендуються?

11. Які результати показали методи класифікації та кластеризації у вашому дослідженні? Які режими моделі визначали найкраще/найгірше?

12. Чим відрізняється кластеризація K-Means від DBSCAN і в яких випадках кожен метод є доцільним?

13. Які відмінності у роботі K-Means та DBSCAN були помітні на ваших даних? Чи правильно DBSCAN виокремив граничні та аномальні точки?

14. Які метрики якості ви застосували (accuracy, F1, confusion matrix, ROC-AUC) та що саме вони показали про ефективність моделей?

15. Яким чином у процесі виконання РГР слід інтерпретувати результати класифікації або кластеризації з урахуванням фізичних властивостей режимів?

16. Які ознаки у Вашій роботі виявились найважливішими для моделі (за feature importance або аналізом коефіцієнтів)?

17. Чому важливо інтерпретувати результати не лише за метриками, а й з точки зору фізики процесів?

18. Які обмеження ви бачите у використаних моделей або підходів?

19. Як би Ви покращили модель, якби мали більше даних або реальні PMU-вимірювання?

20. Чому важливо перевіряти правильність одиниць виміру (В, кВ, pu) перед побудовою моделі, і які наслідки можуть бути у разі помилки?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методи та технології обчислювального інтелекту: Навчальний посібник [Електронний ресурс] : навч. посіб. для студ. спеціальності 122 «Комп'ютерні науки» / І. В. Федорін; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 15,92 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2022. – 314 с.

2. Мацуга О. М., Архангельська Ю. М., Єрещенко Н. М. Навчальний посібник до курсу «Інформаційні технології розпізнавання образів». – Дніпро: ДНУ ім. О. Гончара, 2016. – 135 с.

3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник. – Харків: ХНУ ім. В. Н. Каразіна, 2020. – 157 с.

4. Duda R. O., Hart P. E., Stork D. G. Pattern Classification. 2nd ed. – New York : Wiley-Interscience, 2001. – 688 p.

5. Bishop C. M. Pattern Recognition and Machine Learning. – New York : Springer, 2006. – 738 p.

6. Theodoridis S., Koutroumbas K.Pattern Recognition. 4th ed. – Amsterdam : Academic Press, 2008. – 984 p.

7. Ripley B. D.Pattern Recognition and Neural Networks. – Cambridge : Cambridge University Press, 1996. – 403 p.

8. Kaufman L., Rousseeuw P. J.Finding Groups in Data: An Introduction to Cluster Analysis. – New York : Wiley, 1990. – 342 p.

9. Everitt B., Landau S., Leese M., Stahl D.Cluster Analysis. 5th ed. – Chichester : Wiley, 2011. – 352 p.

10. Rokach L., Maimon O.Data Mining with Decision Trees: Theory and Applications. – Singapore : World Scientific, 2008. – 244 p.

11. Wang L., Khanna R. Applications of Machine Learning in Power Systems. Amsterdam: Elsevier, 2020.

12. Zamora Manzano M., Olguín Ortiz R. et al. Power System Fault Classification Using Machine Learning Techniques // IEEE Access. 2020. Vol. 8. P. 184552–184564.

13. Методичні вказівки «Основи роботи з бібліотекою Pandas». Харків: НТУ «ХПІ», 2022.

14. Shalev-Shwartz S., Ben-David S. Understanding Machine Learning: From Theory to Algorithms. – Cambridge : Cambridge University Press, 2014. – 414 p.

15. Jain A., Duin R. P. W., Mao J. Statistical Pattern Recognition: A Review // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2000. – Vol. 22, No. 1. – P. 4–37.

16. O'Shea K., Nash R. An Introduction to Convolutional Neural Networks. ArXiv preprint.

ДОДАТОК А

Детальний практикум-лістінг коду “Розпізнавання режимів електроенергетичної системи”

Мета: побудувати повний пайплайн від сирих сигналів (Ia, Ib, Ic, Va, Vb, Vc) до ознак, моделей розпізнавання та оцінювання якості для трьох режимів: normal (нормальний режим), powerswing (гойдання потужності), fault_3ph (симетричні короткі замикання).

Реалізація – Python скрипт в юпітер ноутбуці.

```
# === Імпорти та налаштування ===
import os, math, json
import numpy as np, pandas as pd
import matplotlib.pyplot as plt, seaborn as sns
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, label_binarize
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, confusion_matrix,
classification_report

plt.rcParams['figure.figsize'] = (8,4)
plt.rcParams['axes.grid'] = True
sns.set(style='whitegrid', context='notebook')
FEAT_PATH = pd.read_csv("features.csv")
SIG_PATH = pd.read_csv("signals.csv")
print('Очікуємо файли:', SIG_PATH, FEAT_PATH)
Очікуємо файли:       record_id     t          Va          Vb          Vc        Ia  \
0       rec_0001  0.000   -1.826021 -283.792520  284.581209 -7.104207
1       rec_0001  0.001  102.022356 -317.694607  219.818271 -5.946219
2       rec_0001  0.002  193.520128 -327.264001  130.880719 -4.206172
3       rec_0001  0.003  267.386410 -299.723980   36.308900 -2.054397
4       rec_0001  0.004  311.426692 -245.784859  -69.693552  0.298477
...          ...    ...         ...         ...         ...       ...
319995  rec_0320  0.995 -326.263901  162.697737  165.175278 -3.074420
```

Уніфікований вивід метрик (Accuracy у відсотках + дріб)

```
from typing import Optional, Sequence, Union
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    roc_auc_score,
    confusion_matrix,
    classification_report,
)
from IPython.display import display  # щоб працював display у ноутбуці
def _format_pct(x: float, digits: int = 2) -> str:
```

```
    return f"{x * 100:.{digits}f}%"

def print_classification_metrics(
    y_true: Union[Sequence[int], np.ndarray],
    y_pred: Union[Sequence[int], np.ndarray],
    y_proba: Optional[np.ndarray] = None,
    labels: Optional[Sequence[str]] = None,
    title: Optional[str] = None,
    average: str = "macro",
    show_report: bool = True,
    show_cm: bool = True,
    cm_as_heatmap: bool = True,
):
    """
    Друкує базові метрики класифікації, матрицю плутанини та (за бажанням) теплову карту.
    Параметри:
        y_true, y_pred: істинні та передбачені класи (інт або стрінги–сумісні з sklearn)
        y_proba: ймовірності (1D для binary або 2D [n_samples, n_classes] для multi-class)
        labels: імена класів (у тому ж порядку, що й індекси/коди класів)
        title: заголовок блоку
        average: середнення для F1/ROC AUC у multi-class ("macro", "weighted", тощо)
        show_report: показувати classification_report
        show_cm: показувати матрицю плутанини
        cm_as_heatmap: будувати теплову карту (вимагає seaborn/matplotlib)
    """
    y_true = np.asarray(y_true)
    y_pred = np.asarray(y_pred)

    # Базові метрики
    acc = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred, average=average)
    # ROC AUC (за наявності y_proba)
    auc_str = "–"
    if y_proba is not None:
        try:
            y_proba = np.asarray(y_proba)
            # Якщо binary і y_proba має форму (n_samples,), ок
            # Якщо binary і y_proba має форму (n_samples, 2), беремо стовпець позитивного класу (за замовчуванням клас "1")
            if y_proba.ndim == 2 and y_proba.shape[1] == 2 and set(np.unique(y_true)) <= {0, 1}:
                y_proba_bin = y_proba[:, 1]
                auc_val = roc_auc_score(y_true, y_proba_bin)
            elif y_proba.ndim == 1:
                auc_val = roc_auc_score(y_true, y_proba)
            else:
                # multi-class: очікується (n_samples, n_classes)
                auc_val = roc_auc_score(y_true, y_proba, multi_class="ovr", average=average)
            auc_str = f"{auc_val:.4f}"
        except Exception as e:
            auc_str = f"n/a ({e.__class__.__name__})"
```

```
    print(f"Accuracy: {_format_pct(acc)}  ({acc:.4f})")
    print(f"F1-{average}: {f1:.4f}")
    print(f"ROC AUC: {auc_str}")
    if show_cm:
        cm = confusion_matrix(y_true, y_pred)
        # Узгодженість labels з розмірами CM: якщо labels None – використовуємо унікальні значення
        if labels is None:
            # Сортуємо за зростанням, але якщо класи стрінги – стандартне лексикографічне сортування
            unique_classes = np.unique(np.concatenate([y_true, y_pred]))
            labels_used = list(unique_classes)
        else:
            labels_used = list(labels)

        # Якщо надано labels, але їхня довжина не збігається з розміром CM – без назв
        if len(labels_used) != cm.shape[0]:
            labels_used = list(range(cm.shape[0]))
        cm_df = pd.DataFrame(cm, index=labels_used, columns=labels_used)
        print("\nConfusion Matrix:")
        display(cm_df)
        if cm_as_heatmap:
            import matplotlib.pyplot as plt
            import seaborn as sns
            plt.figure(figsize=(4.8, 4.0))
            sns.heatmap(cm_df, annot=True, fmt="d", cmap="Blues")
            plt.title("Confusion Matrix")
            plt.xlabel("Прогноз")
            plt.ylabel("Істина")
            plt.tight_layout()
            plt.show()

    # Класифікаційний звіт
    if show_report:
        print("\nClassification report:")
        if labels is not None:
            # Якщо довжина target_names не відповідає кількості унікальних класів, sklearn кине помилку.
            # Тому підстрахуємося: будемо передавати labels тільки якщо розмір відповідає.
            unique_classes = np.unique(np.concatenate([y_true, y_pred]))
            if len(labels) == len(unique_classes):
                print(classification_report(y_true, y_pred, target_names=list(labels)))
            else:
                print(classification_report(y_true, y_pred))
        else:
            print(classification_report(y_true, y_pred))
```

1) Ознайомлення з даними

```
from pathlib import Path
SIG_PATH = Path('signals.csv')  # або ваш фактичний шлях
np.random.seed(7)
```

```
if SIG_PATH.exists():
    signals = pd.read_csv(SIG_PATH)
    print('signals.csv завантажено:', signals.shape)
else:
    print('Датасет не знайдено – генеруємо приклад...')
    n_per_cls, T, fs = 50, 512, 1000
    t = np.arange(T)/fs

    def gen_normal(n):
        rows=[]
        for _ in range(n):
            f=50
            Va=325*np.sin(2*np.pi*f*t); Vb=325*np.sin(2*np.pi*f*t-2*np.pi/3);
Vc=325*np.sin(2*np.pi*f*t+2*np.pi/3)
            Ia=10*np.sin(2*np.pi*f*t-10*np.pi/180); Ib=10*np.sin(2*np.pi*f*t-2*np.pi/3-
10*np.pi/180); Ic=10*np.sin(2*np.pi*f*t+2*np.pi/3-10*np.pi/180)
            rows.append((Va,Vb,Vc,Ia,Ib,Ic,'normal'))
        return rows

    def gen_powerswing(n):
        rows=[]
        for _ in range(n):
            f=50; swing=0.2*np.sin(2*np.pi*1*t)
            Va=325*(1+swing)*np.sin(2*np.pi*f*t); Vb=325*(1+swing)*np.sin(2*np.pi*f*t-2*np.pi/3);
Vc=325*(1+swing)*np.sin(2*np.pi*f*t+2*np.pi/3)
            Ia=10*np.sin(2*np.pi*f*t-12*np.pi/180); Ib=10*np.sin(2*np.pi*f*t-2*np.pi/3-
12*np.pi/180); Ic=10*np.sin(2*np.pi*f*t+2*np.pi/3-12*np.pi/180)
            rows.append((Va,Vb,Vc,Ia,Ib,Ic,'powerswing'))
        return rows

    def gen_fault(n):
        rows=[]
        for _ in range(n):
            f=50
            Va=325*np.sin(2*np.pi*f*t); Vb=325*np.sin(2*np.pi*f*t-2*np.pi/3);
Vc=325*np.sin(2*np.pi*f*t+2*np.pi/3)
            k=T//2; Va[k:]*=0.2; Vb[k:]*=0.2; Vc[k:]*=0.2
            Ia=25*np.sin(2*np.pi*f*t-8*np.pi/180); Ib=25*np.sin(2*np.pi*f*t-2*np.pi/3-8*np.pi/180);
Ic=25*np.sin(2*np.pi*f*t+2*np.pi/3-8*np.pi/180)
            rows.append((Va,Vb,Vc,Ia,Ib,Ic,'fault_3ph'))
        return rows

    rows = gen_normal(n_per_cls) + gen_powerswing(n_per_cls+4) + gen_fault(n_per_cls)
    recs=[]
    for rid,(Va,Vb,Vc,Ia,Ib,Ic,label) in enumerate(rows):
        df=pd.DataFrame({'Va':Va,'Vb':Vb,'Vc':Vc,'Ia':Ia,'Ib':Ib,'Ic':Ic})
        df['rid']=rid; df['label']=label; recs.append(df)
    signals=pd.concat(recs, ignore_index=True)
    print('Синтетичний signals створено:', signals.shape, signals['label'].value_counts().to_dict())
    # (за бажанням) зберегти на диск:
    signals.to_csv(SIG_PATH, index=False)
    print(f'Збережено у {SIG_PATH}')
```

```
display(signals.head())
```

2) Очищення даних

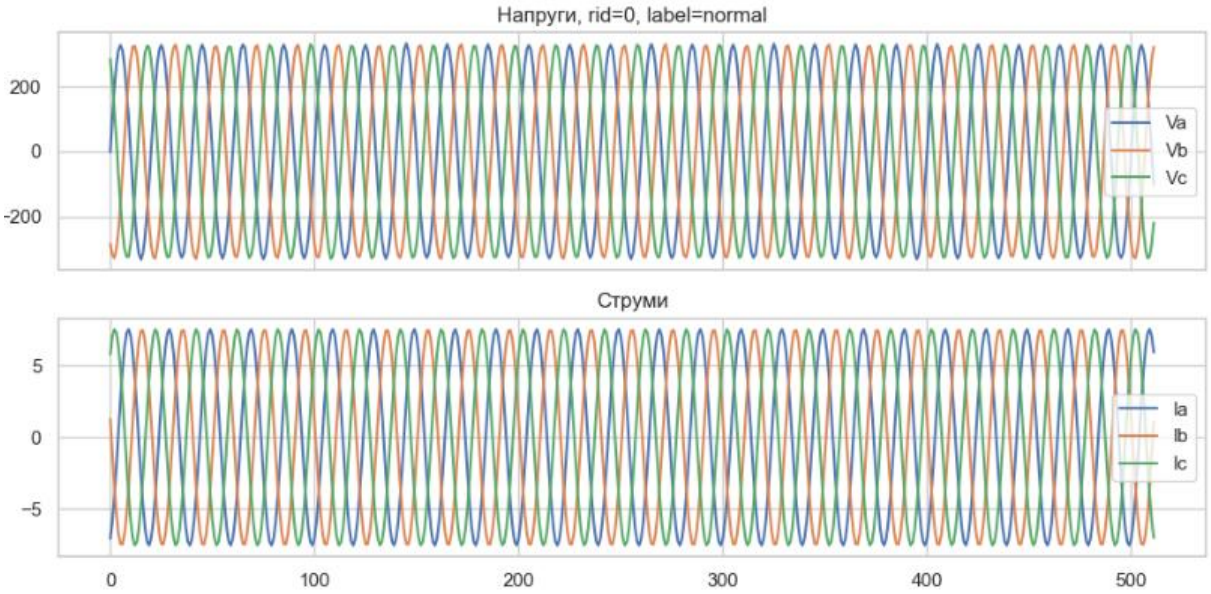
```
print("\nПерші 5 рядків:")
print(signals.head())
print("\nТипи даних та кількість ненульових значень:")
print(signals.info())
print("\nКількість пропусків (топ-20):")
na_counts = signals.isna().sum().sort_values(ascending=False)
print(na_counts.head(20))
print("\nКількість повністю дублікованих рядків:", signals.duplicated().sum())

# Базова статистика
num_cols_all = signals.select_dtypes(include=[np.number]).columns
if len(num_cols_all) > 0:
    print("\nБазова статистика (числові колонки):")
    print(signals.[num_cols_all].describe().T)
# Категоріальні – огляд унікальних
cat_cols_all = signals.select_dtypes(exclude=[np.number]).columns
if len(cat_cols_all) > 0:
    print("\nКатегоріальні колонки (к-ть унікальних):")
    for c in cat_cols_all[:10]:
        print(f"  {c}: { signals[c].nunique()}")

# Баланс класів (якщо є)
if "label" in signals.columns:
    print("\nБаланс класів (label):")
    print(signals["label"].value_counts(dropna=False))
    print("\nЧастки класів:")
    print(signals["label"].value_counts(normalize=True).round(3))

# 2.1. Видалити рядки без обов'язкових полів
must_have = ["label", "record_id"]
for col in must_have:
    if col not in df_clean.columns:
        raise ValueError(f"[ERROR] У features.csv має бути колонка: {col}")
before = len(df_clean)
df_clean = df_clean.dropna(subset=must_have)
print(f"\n[Clean] Видалено {before - len(df_clean)} рядків без label/record_id")

# 2.2. Видалити колонки з >40% пропусків
col_na_ratio = df_clean.isna().mean()
cols_to_drop = col_na_ratio[col_na_ratio > 0.40].index.tolist()
if cols_to_drop:
    print(f"[Clean] Видаляємо колонки з пропусками >40%: {cols_to_drop}")
    df_clean = df_clean.drop(columns=cols_to_drop)

# 2.3. Імпутація: числові → медіана; категоріальні → мода / тег
num_cols_clean = df_clean.select_dtypes(include=[np.number]).columns
cat_cols_clean = df_clean.select_dtypes(exclude=[np.number]).columns
for col in num_cols_clean:
```

```
    if df_clean[col].isna().any():
        df_clean[col] = df_clean[col].fillna(df_clean[col].median())
for col in cat_cols_clean:
    if col in ("label", "record_id"):
        continue
    if df_clean[col].isna().any():
        mode_vals = df_clean[col].mode(dropna=True)
        df_clean[col] = df_clean[col].fillna(mode_vals.iloc[0] if len(mode_vals) else
"__MISSING__")

print("[Clean] Після імпутації пропусків (топ-10):")
print(df_clean.isna().sum().sort_values(ascending=False).head(10))
```

3) Візуальний аналіз режимів

```
import matplotlib.pyplot as plt
# 1) Переконаймося, що 'rid' існує. Якщо ні – намагаємось відновити
if 'rid' not in signals.columns:
    # Спробуємо визначити T. Якщо знаємо, що T=512 – використаємо його.
    candidate_Ts = [512, 1024, 256]
    T_found = None
    for T in candidate_Ts:
        if len(df_clean) % T == 0:
            T_found = T
            break
    if T_found is not None:
        n_rec = len(df_clean) // T_found
        signals = df_clean.copy()
        signals['rid'] = np.repeat(np.arange(n_rec), T_found)
        # Якщо є 'label' і вона стала "рваною", можна зробити її консистентною по rid
        if 'label' in signals.columns:
            # Переприсвоюємо label як моду по кожному rid (на випадок шуму)
            labels_by_rid = signals.groupby('rid')['label'].agg(lambda s: s.mode().iloc[0] if not
s.mode().empty else s.iloc[0])
            signals['label'] = signals['rid'].map(labels_by_rid)
        print(f"[info] Відновлено 'rid' з T={T_found}.")
    else:
        # Фолбек: якщо T невідомий, просто вважаємо, що весь фрейм – один запис
        signals = signals.copy()
        signals['rid'] = 0
        print("[warn] Не вдалося визначити T. Призначено rid=0 для всіх рядків.")

# 2) Обрати прикладовий rid і побудувати графіки
rid_example = signals['rid'].iloc[0]
sub = signals.query('rid == @rid_example')
fig, axes = plt.subplots(2, 1, figsize=(10, 5), sharex=True)
axes[0].plot(sub['Va'], label='Va')
axes[0].plot(sub['Vb'], label='Vb')
axes[0].plot(sub['Vc'], label='Vc')
axes[0].set_title(f'Напруги, rid={rid_example}, label={sub["label"].iloc[0] if "label" in
sub.columns else "n/a"}')
axes[0].legend()
axes[1].plot(sub['Ia'], label='Ia')
```

```
axes[1].plot(sub['Ib'], label='Ib')
axes[1].plot(sub['Ic'], label='Ic')
axes[1].set_title('Струми')
axes[1].legend()

plt.tight_layout()
plt.show()
```



4) Попередня обробка сигналів

```
def moving_rms(x, w=20):
    x=np.asarray(x)
    if len(x)<w: return np.nan*np.ones_like(x)
    c=np.convolve(x**2, np.ones(w)/w, mode='same')
    return np.sqrt(c)

sub = signals.query('rid == @rid_example').copy()
for ch in ['Va','Vb','Vc','Ia','Ib','Ic']:
    sub[f'{ch}_rms'] = moving_rms(sub[ch].values, w=20)
plt.plot(sub['Va'], alpha=0.5, label='Va'); plt.plot(sub['Va_rms'], lw=2, label='Va_rms')
plt.title('Ковзне RMS для Va (приклад)'); plt.legend(); plt.show()
```

5) Виділення ознак (feature engineering)

```
from pathlib import Path
# 1) Шлях для збереження фіч
#FEAT_PATH = Path('features.csv')
# Матриця перетворення abc -> 0-1-2 (послідовності симетрії)
alpha = np.exp(2j * np.pi / 3)
A = (1/3) * np.array(
    [[1, 1, 1],
     [1, alpha, alpha**2],
     [1, alpha**2, alpha]],
    dtype=complex
)
def abc_to_012(va: np.ndarray, vb: np.ndarray, vc: np.ndarray):
```

```
    """
    va, vb, vc: (T,) real arrays
    return: V0, V1, V2 with shape (T,) complex each
    """
    v = np.vstack([va, vb, vc])       # (3, T)
    return A @ v                      # (3, T)

# Перевірка наявності колонок
required_cols = {'rid', 'label', 'Va', 'Vb', 'Vc', 'Ia', 'Ib', 'Ic'}
missing = required_cols - set(signals.columns)
if missing:
    raise KeyError(f"Відсутні колонки у signals: {missing}. "
                   f"Доступні: {list(signals.columns)}")

# 4) Побудова фіч
feat_rows = []
for rid, grp in signals.groupby('rid'):
    label = grp['label'].iloc[0]

    # Переконаємось, що це float-масиви правильної форми
    Va = np.asarray(grp['Va'].values, dtype=float)
    Vb = np.asarray(grp['Vb'].values, dtype=float)
    Vc = np.asarray(grp['Vc'].values, dtype=float)
    Ia = np.asarray(grp['Ia'].values, dtype=float)
    Ib = np.asarray(grp['Ib'].values, dtype=float)
    Ic = np.asarray(grp['Ic'].values, dtype=float)

    # RMS за трьома фазами (ефективне значення середньоквадратичного по часу, усередненe по фазах)
    Vrms = np.sqrt(np.mean(Va**2 + Vb**2 + Vc**2)) / np.sqrt(3)
    Irms = np.sqrt(np.mean(Ia**2 + Ib**2 + Ic**2)) / np.sqrt(3)

    # Послідовності симетрії: V0, V1, V2 (комплексні часові ряди)
    V0, V1, V2 = abc_to_012(Va, Vb, Vc)
    V1_mag = float(np.mean(np.abs(V1)))
    V2_mag = float(np.mean(np.abs(V2)))

    # Миттєва активна потужність (спрощено: сума фазних добутків)
    P = Va * Ia + Vb * Ib + Vc * Ic
    P_mean = float(np.mean(P))
    P_std  = float(np.std(P))

    # Еквівалентні амплітуди та «опір» Z по відношенню V/I (з фільтром малих струмів)
    I_mag = np.sqrt(Ia**2 + Ib**2 + Ic**2) / np.sqrt(3)
    V_mag = np.sqrt(Va**2 + Vb**2 + Vc**2) / np.sqrt(3)
    Z = np.where(I_mag > 1e-6, V_mag / I_mag, np.nan)  # <-- тут був HTML-escape
    Z_med = float(np.nanmedian(Z))

    feat_rows.append({
        'rid':   rid,
        'label': label,
        'Vrms':  Vrms,
```

```
        'Irms':  Irms,
        'V1_mag': V1_mag,
        'V2_mag': V2_mag,
        'P_mean': P_mean,
        'P_std':  P_std,
        'Z_med':  Z_med,
    })

features = pd.DataFrame(feat_rows)
#features.to_csv(FEAT_PATH, index=False)
##print('features.csv збережено →', FEAT_PATH.resolve())
display(features.head())
```

 # ---- ГІСТОГРАМИ КЛЮЧОВИХ ОЗНАК ----

```
import matplotlib.pyplot as plt

candidate_cols = [
    "Vrms_a", "Vrms_b", "Vrms_c",   # фазні RMS напруги
    "Irms_a", "Irms_b", "Irms_c",   # фазні RMS струми
    "V1_mag", "I1_mag",             # модулі позитивної послідовності
    "env_std", "Z_est", "P_mean", "P_std",
]
hist_cols = [c for c in candidate_cols if c in features_df.columns]
if len(hist_cols) > 0:
    n = len(hist_cols)
    ncols = 4
    nrows = int(np.ceil(n / ncols))
    plt.figure(figsize=(3.6 * ncols, 2.8 * nrows))
    for i, col in enumerate(hist_cols, 1):
        ax = plt.subplot(nrows, ncols, i)
        vals = features_df[col].dropna().values
        ax.hist(vals, bins=30, color="#1f77b4", alpha=0.75, edgecolor="white")
        ax.axvline(np.median(vals), color="crimson", linestyle="--", linewidth=1.2, label="median")
        ax.set_title(col)
        ax.grid(True, alpha=0.25)
        if i % ncols == 1:
            ax.set_ylabel("Частота")
        ax.legend(loc="upper right", fontsize=8)
    plt.suptitle("Гістограми ключових ознак", y=1.02)
    plt.tight_layout()
    plt.show()
else:
    print("\n[INFO] Немає ключових числових колонок для гістограм серед candidate_cols.")
```

 # КОНТРОЛЬ ОДИНИЦЬ ВИМІРУ (RMS НАПРУГИ ~ 230 В)

```
vrms_candidates = [c for c in df_clean.columns if c.lower() in ("vrms_a", "vrms_b", "vrms_c")]
if vrms_candidates:
    print("\n=== Контроль фазних RMS-напруг (очікуємо ~230 В) ===")
    lower, upper = 180.0, 260.0  # реалістичні межі для 230 В
    for col in vrms_candidates:
        s = df_clean[col].astype(float)
        print(f"{col}: mean={s.mean():.2f}, p1={s.quantile(0.01):.2f}, p99={s.quantile(0.99):.2f}, "
```

```
            f"min={s.min():.2f}, max={s.max():.2f}")
        ok_ratio = ((s >= lower) & (s <= upper)).mean()
        print(f" -> частка у [{lower}; {upper}] = {ok_ratio:.3f}")
        if ok_ratio < 0.90:
            print(f"[WARN] {col}: багато значень поза 180–260 В. "
                  f"Можлива інша шкала (кВ/pu) або помилка вимірів.")
        # Евристика на кВ
        if 0.15 <= s.mean() <= 0.40:
            print(f"[WARN] {col}: середнє ≈ {s.mean():.3f} (можливо, кВ; перевірте, чи потрібно
×1000).")
else:
    print("\n[INFO] У features.csv немає Vrms_a/Vrms_b/Vrms_c – пропускаємо контроль
напруг.")

# ----  HEATMAP КОРЕЛЯЦІЙ ----
import seaborn as sns
num_df = features_df.select_dtypes(include=[np.number]).copy()
if num_df.shape[1] >= 2:
    nunique = num_df.nunique()
    const_cols = nunique[nunique <= 1].index.tolist()
    if const_cols:
        print(f"[WARN] Константні колонки (будуть виключені перед heatmap): {const_cols}")
        num_df = num_df.drop(columns=const_cols)

    corr = num_df.corr(numeric_only=True, method="pearson")
    plt.figure(figsize=(min(14, 0.4 * corr.shape[1] + 4), min(12, 0.4 * corr.shape[0] + 4)))
    sns.heatmap(
        corr, vmin=-1, vmax=1, cmap="coolwarm", center=0,
        square=False, linewidths=0.5, linecolor="white",
        cbar_kws={"shrink": 0.8, "label": "Коефіцієнт кореляції (Пірсон)"}
    )
    plt.title("Матриця кореляцій числових ознак (features.csv)")
    plt.tight_layout()
    plt.show()
```

6) Формування вибірки для моделі

```
#features = pd.read_csv(FEAT_PATH)
print('Класи:', features['label'].value_counts().to_dict())
X = features.drop(columns=['rid','label', 'record_id']) if 'rid' in features.columns else
features.drop(columns=['label', 'record_id'])
y = features['label']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=7,stratify=y)
X_train.shape, X_test.shape
```

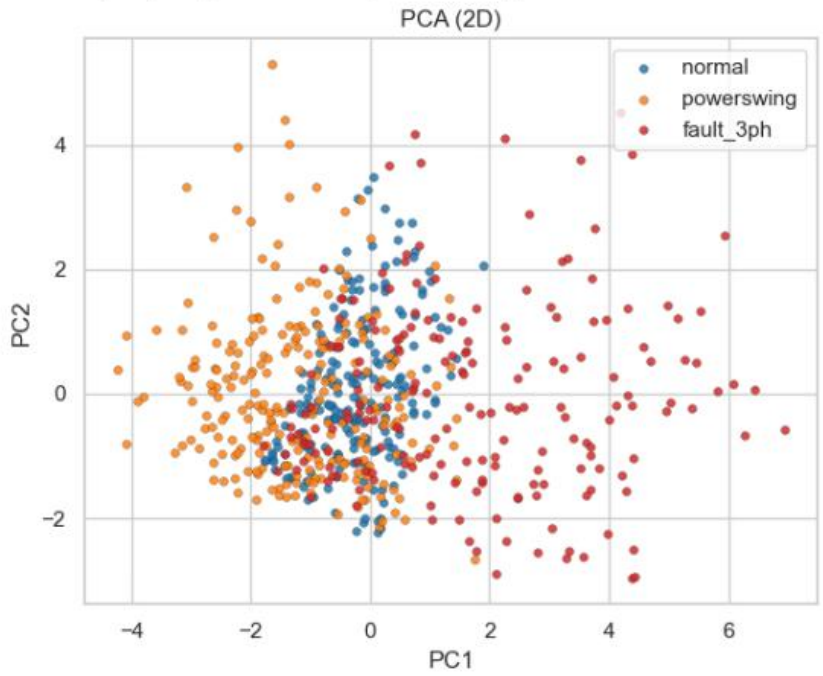
7) Навчання моделей

7) PCA – виділення ключових ознак гойдань

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Аналіз дата сорсу (якщо не був виконаний на попередніх етапах)
```

```
# Перевіримо наявність міток
if 'label' not in features.columns:
    raise KeyError("У DataFrame `features` відсутня колонка 'label'.")

# 1) Побудуємо y (мітки)
y = features['label'].astype(str)
# 2) Сформуємо X лише з числових колонок
#    - відкинемо 'label', 'rid' та усі ненумеричні
drop_cols = [c for c in ['label', 'rid'] if c in features.columns]
num_only = features.drop(columns=drop_cols, errors='ignore')

# Вибираємо тільки реальні числові (float/int), уникаємо complex і object:
X = num_only.select_dtypes(include=[np.number]).copy()
# Якщо раптом є комплексні (np.complexfloating), відкиньмо їх:
complex_cols = [c for c in X.columns if np.iscomplexobj(X[c].to_numpy())]
if complex_cols:
    X = X.drop(columns=complex_cols)
    print(f"[info] Відкинуто комплексні колонки: {complex_cols}")

# Діагностика: чи лишилися колонки
if X.shape[1] == 0:
    raise ValueError("У `X` не залишилося числових колонок для PCA. Перевір, що `features`
містить числові фічі.")

# Масштабування + PCA (2 компоненти)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2, random_state=7)
Xp = pca.fit_transform(X_scaled)
print("Пояснена дисперсія двох компонент:", np.round(pca.explained_variance_ratio_, 4))

# Візуалізація
color_map = {'normal':'#1f77b4', 'powerswing':'#ff7f0e', 'fault_3ph':'#d62728'}
fig, ax = plt.subplots(1, 1, figsize=(6, 5))
for cls, c in color_map.items():
    idx = (y.values == cls)
    if idx.sum() == 0:
        continue
    ax.scatter(Xp[idx, 0], Xp[idx, 1], s=18, c=c, label=cls,
               alpha=0.85, edgecolors='k', linewidths=0.2)
ax.set_title('PCA (2D)')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend()
plt.tight_layout()
plt.show()
```

```
Пояснена дисперсія двох компонент: [0.4895 0.2529]
```



#Варіант 2 для побудови діаграми

```
Xp = PCA(n_components=2, random_state=7).fit_transform(StandardScaler().fit_transform(X))
print('Пояснена дисперсія двох компонент:', PCA(n_components=2,
random_state=7).fit(StandardScaler().fit_transform(X)).explained_variance_ratio_)
color_map={'normal':'#1f77b4','powerswing':'#ff7f0e','fault_3ph':'#d62728'}
fig,ax=plt.subplots(1,1,figsize=(6,5))
for cls,c in color_map.items():
    idx=(y.values==cls)
    ax.scatter(Xp[idx,0],Xp[idx,1],s=18,c=c,label=cls,alpha=0.85,edgecolor='k',linewidths=0.2)
ax.set_title('PCA (2D)'); ax.legend(); plt.show()
```

8 Класифікація

 # Logistic Regression

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, classification_report
from sklearn.pipeline import Pipeline
```

 # 0) Очистимо X_train / X_test: залишимо лише числові колонки, узгоджено з train

```
numeric_cols = X_train.select_dtypes(include=[np.number]).columns.tolist()
# Приберемо потенційні комплексні та inf
def clean_X(df, cols):
    X = df[cols].copy()
    # Якщо раптом колонки комплексні
    complex_cols = [c for c in X.columns if np.iscomplexobj(X[c].to_numpy())]
    if complex_cols:
        X = X.drop(columns=complex_cols)
    # Замінити inf на NaN (далі імпутер)
    X = X.replace([np.inf, -np.inf], np.nan)
```

```
    return X
X_train_num = clean_X(X_train, numeric_cols)
X_test_num  = clean_X(X_test,  numeric_cols)

# 1) Поліноміальні ознаки: вчимося на train, на test лише transform
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly_train = poly_features.fit_transform(X_train_num)
X_poly_test  = poly_features.transform(X_test_num)

# 2) Імпутер + скейлер (рекомендовано для ЛогРега)
imp = SimpleImputer(strategy='median')
scaler = StandardScaler(with_mean=False)  # з poly ознаками інколи sparse; with_mean=False безпечніше
X_poly_train = scaler.fit_transform(imp.fit_transform(X_poly_train))
X_poly_test  = scaler.transform(imp.transform(X_poly_test))

# 3) Навчання LogisticRegression (мультиклас)
#    class_weight='balanced' ок для дисбалансу; multi_class='multinomial' + lbfgs – стабільний варіант
for C in [0.01, 0.1, 1, 3, 10, 30]:
    log_reg = LogisticRegression(
        C=C, class_weight="balanced",
        solver='lbfgs', multi_class='multinomial',
        max_iter=2000, n_jobs=None  # n_jobs ігнорується для lbfgs
    )
    lf = log_reg.fit(X_poly_train, y_train)
    y_pred_p = lf.predict_proba(X_poly_test)

    # Мультикласова AUC (OVR макро-середня)
    auc_macro = roc_auc_score(y_test, y_pred_p, multi_class='ovr', average='macro')
    print(f"C={C:<5}  AUC-macro={auc_macro:.4f}")

# (за бажанням) підсумковий звіт для найкращого C
best_C = 3
log_reg = LogisticRegression(
    C=best_C, class_weight="balanced",
    solver='lbfgs', multi_class='multinomial',
    max_iter=2000
).fit(X_poly_train, y_train)

y_pred = log_reg.predict(X_poly_test)
y_proba = log_reg.predict_proba(X_poly_test)
print("\nClassification report:")
print(classification_report(y_test, y_pred, target_names=['normal','powerswing','fault_3ph']))
```

#Класифікація: SVM

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.svm import SVC

# 1) Діагностика: які нечислові є зараз у train?
```

```
non_numeric = X_train.select_dtypes(exclude=[np.number]).columns.tolist()
if non_numeric:
   print("[warn] Ненумеричні колонки у X_train:", non_numeric)

# 2) Формуємо список дозволених фіч з train (лише числові, некомплексні)
numeric_cols = X_train.select_dtypes(include=[np.number]).columns.tolist()
# відкинемо комплексні, якщо раптом
numeric_cols = [c for c in numeric_cols if not np.iscomplexobj(X_train[c].to_numpy())]

# 3) Застосуємо той самий набір колонок до train/test
X_train_clean = X_train[numeric_cols].replace([np.inf, -np.inf], np.nan)
X_test_clean  = X_test[numeric_cols].replace([np.inf, -np.inf], np.nan)

# 4) Пайплайн
pipe_svm = Pipeline([
   ('imputer', SimpleImputer(strategy='median')),
   ('scaler', StandardScaler()),
   ('clf', SVC(kernel='rbf', probability=True, random_state=7))
])
pipe_svm.fit(X_train_clean, y_train)
y_pred  = pipe_svm.predict(X_test_clean)
y_proba = pipe_svm.predict_proba(X_test_clean)

labels = ['normal','powerswing','fault_3ph']
print_classification_metrics(
   y_test, y_pred, y_proba=y_proba, labels=labels, title='SVM (RBF)', average='macro'
)
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.preprocessing import PolynomialFeatures

svm_clf = Pipeline((
      ("poly_features", PolynomialFeatures(degree=2)), #1. Створення поліномних фіч
      ("scaler", StandardScaler()),                    #2. Нормалізація
      ("svm_clf", LinearSVC(C=1, loss="hinge"))      #3. Навчання моделі
))

svm_clf.fit(X_train, y_train)
p = svm_clf.predict(X_test)
print(confusion_matrix(y_test, p))
print(classification_report(y_test, p))
print("Accuracy:", str(accuracy_score(y_test, p)))
```

Класифікація: RandomForest

```
pipe_rf =
Pipeline([('scaler',StandardScaler()),('clf',RandomForestClassifier(n_estimators=300,random_state=
7))])
pipe_rf.fit(X_train,y_train)
y_pred=pipe_rf.predict(X_test); y_proba=pipe_rf.predict_proba(X_test)
print_classification_metrics(y_test,y_pred,y_proba=y_proba,labels=labels,title='Random Forest
(n=300)',average='macro')
```

```
print(y_pred, y_proba)
print(accuracy_score(y_test, y_pred))
```

Класифікація: KNeighbors

```
from sklearn.metrics import *
pipe_knn = Pipeline([('scaler',StandardScaler()),('clf',KNeighborsClassifier(n_neighbors=7))])
pipe_knn.fit(X_train,y_train)
y_pred=pipe_knn.predict(X_test); y_proba=pipe_knn.predict_proba(X_test)
print_classification_metrics(y_test,y_pred,y_proba=y_proba,labels=labels,title='k-NN
(k=7)',average='macro')

print(y_pred, y_proba)
print(accuracy_score(y_test, y_pred))
```

Додаткові імовірнісні метрики: Brier та RMSE(prob)

```
for name, pipe in [('SVM',pipe_svm),('RF',pipe_rf),('kNN',pipe_knn)]:
    if hasattr(pipe,'predict_proba'):
        proba=pipe.predict_proba(X_test)
        y_oh = label_binarize(y_test, classes=pipe.named_steps['clf'].classes_)
        mse = np.mean((y_oh - proba)**2); rmse=np.sqrt(mse)
        print(f"{name}: Brier={mse:.6f}, RMSE(prob)={rmse:.6f}")
```

8) Кластеризація: k-Means, DBSCAN

```
X_scaled=StandardScaler().fit_transform(X)
Xp=PCA(n_components=2,random_state=7).fit_transform(X_scaled)
km=KMeans(n_clusters=3,random_state=7); km_labels=km.fit_predict(X_scaled)
db=DBSCAN(eps=0.8,min_samples=10); db_labels=db.fit_predict(X_scaled)
fig,axs=plt.subplots(1,2,figsize=(10,4))
axs[0].scatter(Xp[:,0],Xp[:,1],c=km_labels,cmap='tab10',s=18,edgecolor='k',linewidths=0.2);
axs[0].set_title('k-Means (PCA)')
axs[1].scatter(Xp[:,0],Xp[:,1],c=db_labels,cmap='tab10',s=18,edgecolor='k',linewidths=0.2);
axs[1].set_title('DBSCAN (PCA)')
plt.tight_layout(); plt.show()

try:
    import tensorflow as tf
    from tensorflow.keras import layers, models
    cnn=models.Sequential([
        layers.Input(shape=(64,64,1)),
        layers.Conv2D(16,(3,3),activation='relu'), layers.MaxPooling2D(),
        layers.Conv2D(32,(3,3),activation='relu'), layers.MaxPooling2D(),
        layers.Flatten(), layers.Dense(64,activation='relu'), layers.Dense(3,activation='softmax')
    ])
    cnn.summary()
except Exception as e:
    print('TensorFlow не знайдено або не ініціалізовано:', e)
```

9) (Опціонально) CNN для спектрограм / dq0

```
try:
    from scipy.signal import stft
```

```
    rid0=signals['rid'].iloc[0]
    sub=signals.query('rid == @rid0')
    f,tt,Zxx=stft(sub['Va'].values,fs=1000,nperseg=128,noverlap=64)
    S=np.abs(Zxx)
    plt.figure(figsize=(5,3)); plt.pcolormesh(tt,f,20*np.log10(S+1e-
6),shading='gouraud',cmap='magma')
    plt.title('Спектрограма Va (демо)'); plt.ylabel('Hz'); plt.xlabel('s'); plt.colorbar(label='dB');
plt.tight_layout(); plt.show()
except Exception as e:
    print('STFT пропущено:', e)

try:
    import tensorflow as tf
    from tensorflow.keras import layers, models
    cnn=models.Sequential([
        layers.Input(shape=(64,64,1)),
        layers.Conv2D(16,(3,3),activation='relu'), layers.MaxPooling2D(),
        layers.Conv2D(32,(3,3),activation='relu'), layers.MaxPooling2D(),
        layers.Flatten(), layers.Dense(64,activation='relu'), layers.Dense(3,activation='softmax')
    ])
    cnn.summary()
except Exception as e:
    print('TensorFlow не знайдено або не ініціалізовано:', e)
```