

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___» _____ 20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Діагностика множинної мієломи за допомогою методів

машинного навчання»

Виконав:

студент IV курсу, групи КІ-22

Літовченко Владислав Олександрович

Керівник:

Професор кафедри штучного інтелекту, д.т.н., професор

Литвиненко Володимир Іванович

Консультант з нормоконтролю:

фахівець I категорії кафедри штучного інтелекту,

к.т.н., доцент, Комариста Богдана Миколаївна

Рецензент:

с.н.с. лабораторії аналітичної хімії та моніторингу токсичних речовин

Інституту медицини праці ім. Ю.І. Кундіїва, д.т.н.,

Ольшевський Сергій Валентинович

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студент _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«15» січня 2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Літовченку Владиславу Олександровичу

1. Тема роботи «Діагностика множинної мієломи за допомогою методів машинного навчання», керівник роботи Литвиненко Володимир Іванович, доктор технічних наук, професор кафедри, затверджені наказом по НН ІПСА від «27» травня 2026 р. № 1944-с.
2. Термін подання студентом роботи «05» червня 2026 року.
3. Матеріали та результати, отримані під час проходження переддипломної практики, методичні вказівки, технічна література.
4. Зміст розрахунково-пояснювальної записки: вступ; аналіз предметної області; теоретичні основи методів інтелектуального аналізу даних; розробка та програмна реалізація системи; експериментальне дослідження; висновки.
5. Перелік ілюстративного матеріалу: електронна презентація.
6. Дата видачі завдання «16» лютого 2026 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	20.04.2026	Виконано
2	Підготовка першого розділу	27.04.2026	Виконано
3	Підготовка другого розділу	04.05.2026	Виконано
4	Розробка програмного продукту	11.05.2026	Виконано
5	Підготовка третього розділу	18.05.2026	Виконано
6	Підготовка четвертого розділу	25.05.2026	Виконано
7	Підготовка презентації доповіді	01.06.2026	Виконано
8	Оформлення дипломної роботи	05.06.2026	Виконано

Студент

Владислав ЛІТОВЧЕНКО

Керівник

Володимир ЛИТВИНЕНКО

РЕФЕРАТ

Дипломна робота: 103 с., 33 рис., 11 табл., 49 джерел.

МНОЖИННА МІЄЛОМА, БІНАРНА КЛАСИФІКАЦІЯ, ВІДБІР
ОЗНАК, МАШИННЕ НАВЧАННЯ, BORUTA, СТЕКІНГ, XGBOOST,
ІНТЕЛЕКТУАЛЬНА ДІАГНОСТИКА

Об'єкт дослідження — процес інтелектуальної діагностики множинної мієломи за результатами клініко-лабораторного обстеження.

Предмет дослідження — методи відбору ознак та алгоритми машинного навчання для бінарної класифікації наявності захворювання.

Мета роботи — розроблення інтелектуальної системи класифікації пацієнтів з підозрою на множинну мієлому, що поєднує сучасні методи відбору ознак з ансамблевими класифікаторами.

У роботі проаналізовано медичні аспекти захворювання та обмеження класичної діагностики. Обґрунтовано комбінований підхід, де три методи відбору ознак (ентропійний, Boruta, RFE) поєднано з п'ятьма моделями (XGBoost, CatBoost, стекінг, MLP та багатогілкова нейромережа).

Результати роботи — навчено моделі класифікації, виконано 15 експериментів, реалізовано модульну систему мовою Python. Найкраща комбінація Boruta + Stacking досягла OOF AUC = 0,846, F1 = 0,815 та чутливості 0,862; виявлено стійку клінічну сигнатуру з 11 ознак.

Напрями подальшого розвитку — розширення та диверсифікація вибірки, зовнішня валідація моделі, інтеграція цитогенетичних маркерів та впровадження системи у клінічну інфраструктуру.

ABSTRACT

Bachelor's thesis: 103 pages, 33 figures, 11 tables, 49 references.

MULTIPLE MYELOMA, BINARY CLASSIFICATION, FEATURE SELECTION, MACHINE LEARNING, BORUTA, STACKING, XGBOOST, INTELLIGENT DIAGNOSIS

Object of the study — the process of intelligent diagnosis of multiple myeloma based on clinical and laboratory examination results.

Subject of the study — feature selection methods and machine learning algorithms for binary classification of the disease presence.

Purpose of the work — to develop an intelligent classification system for patients with suspected multiple myeloma, combining modern feature selection techniques with ensemble classifiers.

The thesis analyzes the medical aspects of the disease and the limitations of classical diagnosis. A combined approach is justified, pairing three feature selection methods (information gain, Boruta, RFE) with five models (XGBoost, CatBoost, stacking, MLP, and a multi-branch neural network).

Results — classification models were trained, 15 experiments were conducted, and a modular Python system was implemented. The best combination, Boruta + Stacking, achieved OOF AUC = 0.846, F1 = 0.815, and sensitivity 0.862; a stable clinical signature of 11 features was identified.

Directions for further research — dataset expansion and diversification, external model validation, integration of cytogenetic markers, and system implementation into clinical infrastructure.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	8
ВСТУП	10
РОЗДІЛ 1 АНАЛІЗ ОБЛАСТІ ДІАГНОСТИКИ ОНКОЛОГІЧНИХ ЗАХВОРЮВАНЬ З ВИКОРИСТАННЯМ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ.....	15
1.1 Множинна мієлома як медична задача.....	15
1.2 Існуючі підходи до діагностики множинної мієломи.....	21
1.3 Огляд методів інтелектуального аналізу даних у медицині	26
1.4 Огляд попередніх досліджень	31
1.5 Постановка задачі	35
Висновки до розділу 1	36
РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ ДЛЯ КЛАСИФІКАЦІЇ ММ	38
2.1 Попередня обробка та нормалізація даних	38
2.2 Методи відбору ознак	43
2.3 Ансамблеві методи класифікації.....	48
2.4 Методи глибинного навчання	53
2.5 Метрики оцінювання якості	59
Висновки до розділу 2.....	64
РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ ДІАГНОСТИКИ ММ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ.....	65
3.1 Загальна архітектура системи	65
3.2 Опис вхідних даних.....	68
3.3 Методика відбору ознак.....	69
3.4 Методика навчання класифікаційних моделей та схема валідації	72
3.5 Результати та порівняльний аналіз	75
3.6 Клінічна інтерпретація відібраних ознак	83
Висновки до розділу 3	88
РОЗДІЛ 4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОЇ ДІАГНОСТИКИ ММ.....	89
4.1 Загальна структура програмного проекту.....	89

4.2 Модуль конфігурації та завантаження даних	91
4.3 Модуль відбору ознак	91
4.4 Модуль моделей.....	92
4.5 Модуль крос-валідації.....	93
4.6 Модуль статистичного аналізу.....	94
4.7 Оркестрація експериментів та візуалізація.....	94
4.8 Тестування системи.....	95
Висновки до розділу 4.....	96
ВИСНОВКИ.....	97
ПЕРЕЛІК ПОСИЛАНЬ.....	100
ДОДАТОК А ЛІСТИНГ КОДУ	105

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ВООЗ – Всесвітня організація охорони здоров'я
ДІ – довірчий інтервал
ІАД – інтелектуальний аналіз даних
КМ – кістковий мозок
КТ – комп'ютерна томографія
ММ – множинна мієлома
МН – машинне навчання
МРТ – магнітно-резонансна томографія
НМ – нейронна мережа
ПЕТ – позитронно-емісійна томографія
ФДГ – фтордезоксиглюкоза (^{18}F -ФДГ — радіофармпрепарат для ПЕТ)
ІІІ – штучний інтелект
AUC – Area Under the Curve (площа під ROC-кривою)
BN – Batch Normalization (пакетна нормалізація)
CatBoost – Categorical Boosting
CBC – Complete Blood Count (загальний аналіз крові)
CRAB – Calcium, Renal, Anemia, Bone (критерії органного ураження при ММ)
CSV – Comma-Separated Values
F1 – F1-score (F-міра)
FN – False Negative
FP – False Positive
Hb – Hemoglobin (гемоглобін)
IMWG – International Myeloma Working Group
ISS – International Staging System
LDH – Lactate Dehydrogenase (лактатдегідрогеназа)

ML – Machine Learning

MLP – Multi-Layer Perceptron (багатошаровий перцептрон)

OOF – Out-Of-Fold (позафолдові прогнози)

ReLU – Rectified Linear Unit

RF – Random Forest (випадковий ліс)

RFE – Recursive Feature Elimination

R-ISS – Revised International Staging System

ROC – Receiver Operating Characteristic

SHAP – SHapley Additive exPlanations

SLiM – Sixty percent / Light chains / MRI (критерії IMWG 2014)

SMOTE – Synthetic Minority Over-sampling Technique

SVM – Support Vector Machine

TN – True Negative

TP – True Positive

XGBoost – eXtreme Gradient Boosting

ВСТУП

Множинна мієлома¹ (ММ) належить до групи злоякісних новоутворень плазматичних клітин кісткового мозку. Хвороба супроводжується продукцією моноклонального білка, ураженням скелета, нирок та системи кровотворення. За даними Міжнародного агентства з вивчення раку, на ММ припадає близько 1 % усіх онкологічних і 10–13 % гематологічних злоякісних патологій [1]. У країнах Європи захворюваність становить 4,5–6 випадків на 100 000 населення на рік, а у віковій групі понад 75 років показник зростає до 46 на 100 000 [2]. Медіана віку на момент встановлення діагнозу — 70 років. Останні два десятиліття принесли значний прогрес у лікуванні: медіана загальної виживаності виросла з 3 до 7–10 років завдяки новим режимам терапії та трансплантації гемопоетичних стовбурових клітин [3].

Прогрес у лікуванні робить раннє виявлення хвороби ще важливішим. Чим швидше пацієнт потрапляє до спеціалізованого гематолога, тим вищі шанси на тривалу ремісію і тим менш агресивна терапія знадобиться. Проте діагностичний шлях лишається проблемним. Симптоми ММ на ранніх стадіях неспецифічні — біль у кістках, рецидивні інфекції, втомлюваність, анемія невідомого генезу. Пацієнт нерідко проходить обстеження у кількох лікарів упродовж місяців, перш ніж його направлять до гематолога. Лабораторне підтвердження діагнозу вимагає одночасного аналізу великої кількості показників: загального і біохімічного аналізів крові, імуноелектрофорезу

¹ Тут і нижче використано такий(і) інструмент(и) штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

білків сироватки та сечі, кількісного визначення вільних легких ланцюгів імуноглобулінів, морфологічного дослідження пунктату кісткового мозку, рентгенологічного обстеження скелета. У такій ситуації якість діагнозу залежить від досвіду лікаря та його здатності інтегрувати десятки параметрів водночас.

Одним зі шляхів автоматизації цього процесу є застосування методів інтелектуального аналізу даних (ІАД) та машинного навчання. Алгоритми класифікації, побудовані на ансамблях дерев рішень — випадковий ліс, eXtreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost), Light Gradient-Boosting Machine (LightGBM) — стабільно показують високу ефективність на табличних медичних даних [4, 5]. Стекінг та багатогілкові архітектури дозволяють об'єднати прогнози кількох базових моделей і отримати додатковий приріст точності [6]. Окремий напрям становлять моделі глибинного навчання, зокрема багатосарові перцептрони, які добре працюють зі складними нелінійними залежностями між клінічними показниками [7].

Не менш важливим є етап відбору ознак. У клінічних наборах їх може бути понад дві сотні, і не всі однаково інформативні. Включення нерелевантних або сильно корельованих ознак знижує точність моделі, ускладнює її інтерпретацію та робить вразливою до перенавчання. У роботі досліджуються три методологічно різні підходи до відбору: ентропійний на основі інформаційного приросту (фільтрового типу), Boruta з механізмом тіньових ознак (обгортковий) та рекурсивне усунення ознак (Recursive Feature Elimination, RFE) (вбудований) [8, 9]. Їх порівняння дає змогу визначити оптимальну стратегію формування простору ознак саме для задачі діагностики ММ.

Зважаючи на викладене, дослідження методів інтелектуального аналізу даних для класифікації пацієнтів з підозрою на множинну мієлому є актуальним як у науковому, так і у клінічному вимірах.

Мета роботи — підвищення точності класифікації пацієнтів за наявністю ознак множинної мієломи на основі клініко-лабораторних показників шляхом розробки програмної системи, що поєднує сучасні методи відбору ознак та ансамблеві алгоритми машинного навчання.

Для досягнення мети розв'язуються такі задачі:

1. проаналізувати медичні аспекти множинної мієломи та існуючі підходи до її діагностики;
2. дослідити теоретичні засади методів відбору ознак (Information Gain, Boruta, RFE), ансамблевих алгоритмів класифікації та моделей глибинного навчання;
3. виконати попередню обробку реального клініко-лабораторного набору даних і підготувати його до моделювання;
4. реалізувати засобами мови Python програмну систему, що включає блок відбору ознак та блок класифікації;
5. провести експериментальне дослідження для алгоритмів XGBoost, CatBoost, стекінгу, багатогілкового ансамблю та багатошарового перцептрона з оцінкою повного набору метрик (Accuracy, Precision, Recall, Specificity, F1-score, Area Under the Curve (AUC));
6. виконати порівняльний аналіз моделей з оцінкою статистичної значущості відмінностей;
7. дослідити вплив окремих клінічних показників на якість класифікації та обґрунтувати медичну значущість відібраних ознак;
8. сформулювати висновки щодо доцільності застосування запропонованого підходу.

Об'єкт дослідження — процес діагностування множинної мієломи на основі клініко-лабораторних показників пацієнта.

Предмет дослідження — методи відбору ознак та ансамблеві алгоритми машинного навчання для задачі бінарної класифікації пацієнтів з підозрою на множинну мієлому.

Методи дослідження. Застосовано загальнонаукові методи аналізу та синтезу при огляді літератури, методи математичної статистики при обробці результатів експерименту, методи інтелектуального аналізу даних для відбору ознак (Information Gain, Boruta, RFE), ансамблеві методи машинного навчання (XGBoost, CatBoost, стекінг, багатогілковий ансамбль) та методи глибинного навчання (багатошаровий перцептрон) для розв'язання задачі бінарної класифікації. Реалізація виконана мовою Python.

Наукова новизна одержаних результатів полягає у запропонованому підході до діагностики ММ, що поєднує три методологічно різнотипні методи відбору ознак з ансамблевими алгоритмами та моделями глибинного навчання. На відміну від робіт, що використовують класичні WEKA-алгоритми у поєднанні з фільтрами на взаємній інформації, дослідження застосовує сучасний стек методів машинного навчання та виконує порівняльний аналіз із оцінкою статистичної значущості відмінностей між моделями. Окремо проведено клінічну інтерпретацію відібраних ознак у контексті патофізіології множинної мієломи.

Практичне значення отриманих результатів полягає у створенні програмної системи класифікації, яка може використовуватися як допоміжний інструмент лікаря-гематолога на етапі первинної оцінки пацієнтів з підозрою на ММ. Система автоматизує аналіз великої кількості клініко-лабораторних показників, виявляє найбільш інформативні з них та оцінює ймовірність наявності захворювання. Її застосування сприяє ранньому виявленню патології та своєчасному скеруванню пацієнта на додаткові обстеження.

Структура роботи. Дипломна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел і додатків. У першому розділі проаналізовано предметну область: розглянуто множинну мієлому як медичну задачу, існуючі підходи до її діагностики та огляд методів інтелектуального аналізу даних у медицині, а також сформульовано постановку задачі. Другий розділ містить теоретичні основи методів відбору ознак, ансамблевих та нейромережних алгоритмів класифікації й метрик оцінювання якості моделей. У третьому розділі описано архітектуру розробленої системи, методику відбору ознак і навчання класифікаційних моделей, а також наведено результати експериментального дослідження п'ятнадцяти комбінацій «метод відбору × класифікатор» з оцінкою статистичної значущості відмінностей та клінічною інтерпретацією відібраних ознак. У четвертому розділі описано програмну реалізацію системи: модулі завантаження даних, відбору ознак, навчання моделей, крос-валідації, статистичного аналізу та тестування.

РОЗДІЛ 1 АНАЛІЗ ОБЛАСТІ ДІАГНОСТИКИ ОНКОЛОГІЧНИХ ЗАХВОРЮВАНЬ З ВИКОРИСТАННЯМ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

1.1 Множинна мієлома як медична задача

Загальна характеристика захворювання

Множинна мієлома — це В-клітинна злоякісна пухлина з диференціюванням на рівні плазматичних клітин кісткового мозку. Хвороба характеризується неконтрольованою проліферацією моноклональних плазмоцитів, які виробляють патологічний моноклональний імуноглобулін (М-протеїн) і витісняють здорові гемопоетичні лінії з кісткового мозку [12]. Згідно з класифікацією ВООЗ, ММ належить до групи плазмоклітинних дискразій разом з моноклональною гамопатією невизначеного значення (MGUS) і тліючою мієломою.

Клініко-біологічна сутність захворювання визначається трьома процесами, що відбуваються одночасно. По-перше, моноклональні плазмоцити розмножуються переважно в кістковому мозку, формуючи багатогогнищеве (множинне) ураження скелета. По-друге, продукований ними М-протеїн нагромаджується в плазмі крові та сечі, спричиняючи ниркові ускладнення і підвищення в'язкості крові. По-третє, унаслідок витіснення нормальних клітинних ліній порушується гемопоєз, що клінічно проявляється анемією, лейкопенією і тромбоцитопенією [3]. У запущених випадках додається ураження нирок через гіперкальціємію та нагромадження легких ланцюгів імуноглобулінів у каналцях, а також вторинний амілоїдоз з ураженням серця, нирок і периферичної нервової системи.

Згідно з критеріями Міжнародної робочої групи з мієломи (International Myeloma Working Group, IMWG), діагноз ММ встановлюється за наявності понад 10 % клональних плазмоцитів у кістковому мозку або біопсійно підтвердженої плазмоцитоми, у поєднанні щонайменше з однією з ознак

ураження органів-мішеней. Ці ознаки об'єднують аббревіатурою CRAB (Calcium, Renal, Anemia, Bone): гіперкальціємія ($>2,75$ ммоль/л), порушення функції нирок (креатинін >177 мкмоль/л або кліренс <40 мл/хв), анемія (гемоглобін <100 г/л або зниження на 20 г/л від норми) і кісткові ураження за результатами рентгенографії, комп'ютерної томографії (КТ) або магнітно-резонансної томографії (МРТ) [15]. У 2014 році IMWG розширила критерії, додавши до CRAB ще три біомаркери — так звані SLiM — критерії (Sixty percent — ≥ 60 % клональних плазмоцитів; Light chains — співвідношення вільних легких ланцюгів ≥ 100 ; MRI — понад одне вогнище за МРТ), які дозволяють встановлювати діагноз ще до появи кінцевого органного ураження.

Епідеміологія

На множинну мієлому припадає близько 1 % усіх злоякісних новоутворень і 10–13 % гематологічних онкозахворювань. У глобальному масштабі щорічно реєструється понад 175 тисяч нових випадків ММ і близько 117 тисяч смертей [16]. Захворюваність варіює залежно від географічного регіону, етнічної приналежності та віку.

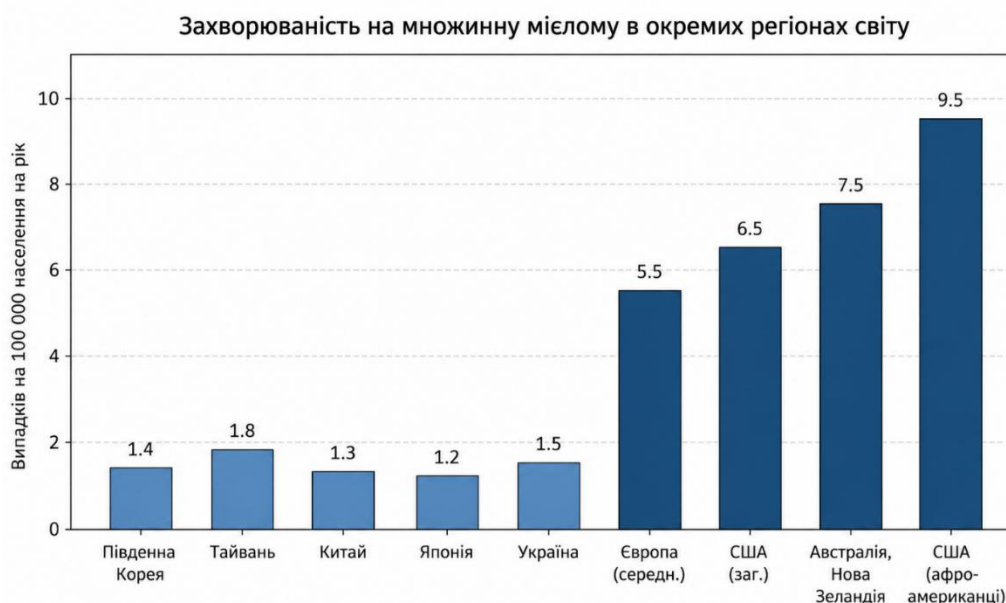


Рисунок 1.1 – Захворюваність на множинну мієлому в окремих регіонах світу, випадків на 100 000 населення на рік [1, 2, 16, 17]

З наведених даних видно різницю показників у понад 7 разів між країнами Східної Азії (1,2–1,8 на 100 000) і афроамериканською популяцією США (9–10 на 100 000). Цю різницю пов'язують з генетичними чинниками, особливо з поліморфізмом генів регуляції імунної відповіді. У країнах Європи захворюваність становить у середньому 4,5–6 випадків на 100 000 населення на рік. В Україні за даними Національного канцер-реєстру показник нижчий — приблизно 1,5 на 100 000, однак вважається, що значна частина випадків залишається невиявленою або діагностується на пізніх стадіях, коли формальна реєстрація відбувається вже з ускладненнями [17].

Особливістю ММ є виражена вікова залежність захворюваності. Медіана віку на момент встановлення діагнозу становить 70 років; у пацієнтів молодше 40 років хвороба зустрічається в менш ніж 2–3 % випадків, тоді як у віковій групі понад 75 років захворюваність зростає до 46 на 100 000 населення — тобто майже в 10 разів вище за середній показник у популяції.

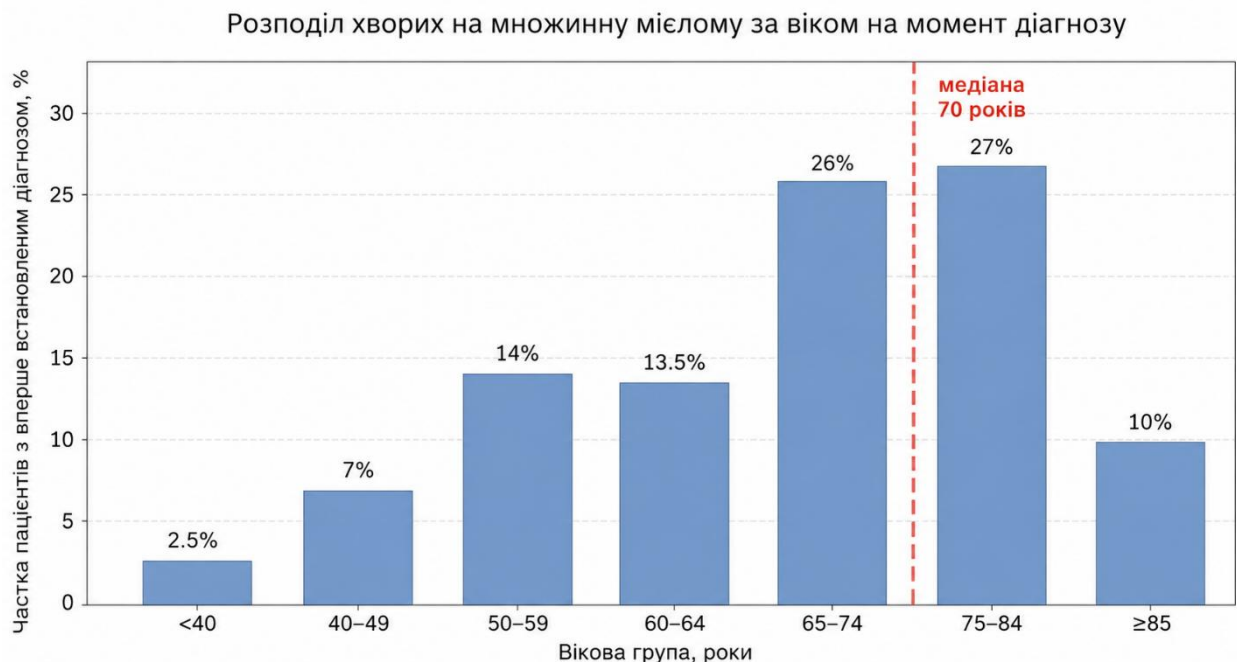


Рисунок 1.2 – Розподіл хворих на множинну мієлому за віком на момент встановлення діагнозу [2, 3, 12]

Чоловіки хворіють дещо частіше — близько 56–60 % випадків. У майбутньому очікується подальше зростання абсолютної кількості хворих, що пов'язано як з покращенням діагностики, так і з демографічним старінням населення [16].

Клінічна картина і причини діагностичних затримок

Симптоматика ММ є вкрай неспецифічною, особливо на ранніх стадіях. Найчастіші скарги пацієнтів — біль у кістках (передусім у поперековому відділі хребта і ребрах), загальна слабкість, втомлюваність, рецидивні інфекційні захворювання, незрозуміла втрата ваги. Приблизно у 30–40 % випадків першим клінічним проявом є патологічні переломи кісток, які виникають без адекватної травми [12].

Окремою клінічною проблемою є ниркова недостатність, з якою у ~20 % хворих ММ вперше потрапляє у стаціонар. У таких випадках нефролог змушений диференціювати моноклональну гамопатію від інших причин гострого ураження нирок, що додатково затягує постановку правильного діагнозу. Часта супутня патологія у літніх пацієнтів — артеріальна гіпертензія, серцева недостатність, остеопороз — призводить до того, що окремі симптоми ММ помилково приписують іншим хворобам.

Як наслідок, середній час від появи перших симптомів до встановлення діагнозу становить 5–7 місяців, а у частини пацієнтів — понад рік. За цей період хвороба прогресує і часто переходить у стадію з ураженням органів-мішеней, що погіршує прогноз і робить лікування складнішим. Раннє виявлення ММ є нерозв'язаною клінічною проблемою, і саме тут інструменти автоматичної підтримки рішень мають найбільший потенціал.

Стадування

У клінічній практиці використовуються дві основні системи стадування ММ: класична система Durie-Salmon, запропонована у 1975 році [14], і Міжнародна система стадування (International Staging System, ISS), представлена у 2005 році [11] та переглянута у 2015 році з включенням цитогенетичних даних та рівня лактатдегідрогенази (Lactate Dehydrogenase,

LDH) — Revised ISS (R-ISS) [10]. Ключові характеристики цих систем порівняно в таблиці 1.1.

Таблиця 1.1 – Системи стадування множинної мієломи

Стадія	<i>Durie-Salmon (1975)</i>	<i>ISS (2005)</i>	<i>R-ISS (2015)</i>
I	Hb >100 г/л, Са норма, ≤ 1 кісткова лезія, низький рівень М-протеїну	$\beta 2$ -мікроглобулін <3,5 мг/л і альбумін ≥ 35 г/л	ISS I + стандартний цитогенетичний ризик + нормальний LDH
II	Проміжний стан	Проміжний	Не ISS I і не ISS III
III	Hb <85 г/л, Са >2,98 ммоль/л, >2 кісткових лезій, високий М-протеїн	$\beta 2$ -мікроглобулін $\geq 5,5$ мг/л	ISS III + (високий цитогенетичний ризик або підвищений LDH)

ISS-стадування зосереджене на двох простих лабораторних показниках — рівні $\beta 2$ -мікроглобуліну і альбуміні сироватки крові, і саме воно широко використовується в сучасних клінічних дослідженнях завдяки своїй відтворюваності і незалежності від методу візуалізації. R-ISS додатково враховує цитогенетичні аберації високого ризику (del17p, t(4;14), t(14;16)) та рівень лактатдегідрогенази, що дозволяє точніше прогнозувати перебіг хвороби.

З погляду задачі автоматичної класифікації, важливо те, що обидві системи побудовані на тих самих лабораторних параметрах, які входять до звичайного клінічного обстеження пацієнта. Це створює природну основу для розробки систем підтримки прийняття рішень, які могли б на основі тих самих даних видавати попередню оцінку ризику.

Прогноз і виживаність

Перебіг ММ значною мірою залежить від стадії на момент діагнозу і обраного протоколу лікування. У 1970-х роках, до впровадження нових препаратів, 5-річна виживаність хворих на ММ становила лише 25 %. Послідовне впровадження високодозової хіміотерапії з трансплантацією аутологічних гемопоетичних стовбурових клітин (з 1990-х років), а потім інгібіторів протеасоми (бортезомиб, карфілзомиб) та імуномодуляторів (леналідомид, помалідомид) принципово змінило цю картину. У 2020 році 5-річна виживаність у США становила вже близько 62 % [13].

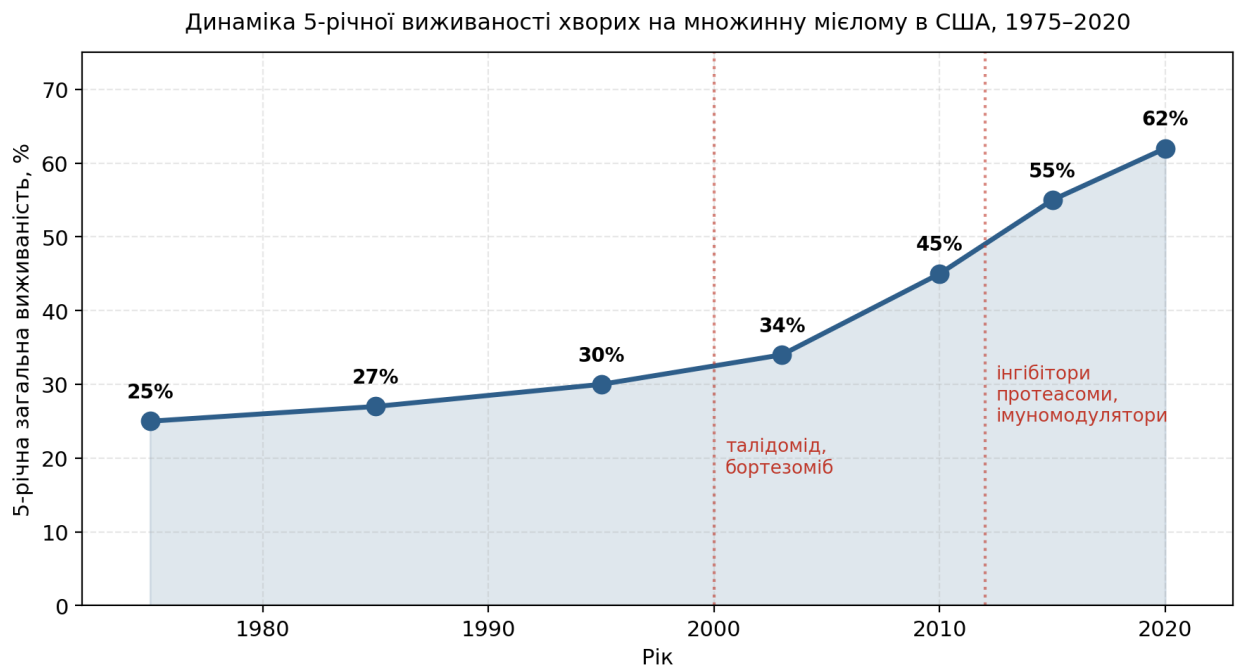


Рисунок 1.3 – Динаміка 5-річної загальної виживаності хворих на множинну мієлому в США, 1975–2020 рр. [3, 13]

При цьому виживаність сильно залежить від стадії на момент діагнозу. Для пацієнтів з ISS I 5-річна виживаність перевищує 75 %, для ISS II — близько 55 %, для ISS III — лише 35–40 % [11]. Це підкреслює критичну важливість ранньої діагностики: пацієнт, виявлений на стадії I, має майже вдвічі вищі шанси прожити п'ять років, ніж пацієнт із запущеною формою. Сучасні

підходи прагнуть зміщувати діагностику в бік раннього виявлення — і саме в цьому контексті алгоритми машинного навчання можуть стати корисним інструментом, особливо на рівні первинної медичної допомоги, де гематолог недоступний, а лікар-терапевт має оцінити велику кількість показників за обмежений час.

Підсумок підрозділу

Множинна мієлома є рідкісним, але клінічно складним онкологічним захворюванням. Його основні особливості з погляду задачі автоматичного діагностування такі: неспецифічність ранніх симптомів, необхідність аналізу великої кількості клініко-лабораторних показників, наявність формалізованих систем стадування (ISS, R-ISS), які спираються на ті самі параметри, що збираються у звичайному обстеженні пацієнта. Сукупність цих факторів робить ММ зручним об'єктом для побудови автоматичних класифікаційних моделей на табличних медичних даних — тих самих, які лікар використовує у щоденній практиці.

1.2 Існуючі підходи до діагностики множинної мієломи

Загальний підхід

Діагностика множинної мієломи спирається на комбіноване використання клінічних, лабораторних, морфологічних та інструментальних методів. Жоден окремий тест не дозволяє встановити діагноз достовірно — рішення ухвалюють на основі сукупності даних згідно з критеріями Міжнародної робочої групи з мієломи [15]. Це робить діагностичний процес багатоетапним і трудомістким, що, з одного боку, забезпечує його точність, а з іншого — створює можливості для оптимізації, в тому числі за допомогою методів інтелектуального аналізу даних.

Клінічне обстеження

На першому етапі лікар оцінює анамнез і скарги пацієнта. Найбільш характерними є біль у кістках (передусім у поперековому відділі хребта, ребрах, тазі), хронічна втомлюваність, рецидивні бактеріальні інфекції, незрозуміла втрата ваги, ознаки ниркової дисфункції. Об'єктивне обстеження виявляє можливу блідість шкірних покривів, болючість при пальпації кісток, неврологічну симптоматику при компресії спинного мозку. Проблема цього етапу полягає в тому, що симптоми ММ нагадують прояви ревматологічних, ендокринних, інфекційних та інших патологій, що часто призводить до тривалого діагностичного пошуку [12, 18].

Лабораторні методи

Лабораторна діагностика складає основу підтвердження діагнозу ММ і включає кілька груп досліджень.

Загальний аналіз крові виявляє нормоцитарну нормохромну анемію ($Hb < 100$ г/л у понад 70 % пацієнтів на момент діагнозу), помірну лейкопенію та тромбоцитопенію. У мазку крові може спостерігатися характерна агрегація еритроцитів у вигляді «монетних стовпчиків» (rouleaux formation), пов'язана з гіперпротеїнемією.

Біохімічний аналіз крові дозволяє оцінити функцію нирок (креатинін, кліренс) і виявити характерні зміни: гіперкальціємію ($> 2,75$ ммоль/л у близько 15 % пацієнтів), підвищення рівня лактатдегідрогенази (LDH) як маркера агресивності пухлини, зниження альбуміну. Окремі біохімічні показники — рівень $\beta 2$ -мікроглобуліну та альбуміну сироватки — формують основу системи стадування ISS [11].

Білкові маркери є патогномонічними для ММ. Методом електрофорезу білків сироватки (SPEP, Serum Protein Electrophoresis) виявляється моноклональний пік (М-протеїн), який відображає продукцію патологічного імуноглобуліну плазмочитами. Метод електрофорезу білків сечі (UPEP, Urine Protein Electrophoresis) дозволяє визначити наявність білка Бенс-Джонса — легких ланцюгів, що виділяються нирками. Імунофікаційний електрофорез

(IFE, Immunofixation electrophoresis) уточнює тип моноклонального білка (IgG, IgA, IgM, IgD, IgE або тільки легкі ланцюги — κ чи λ). Кількісне визначення вільних легких ланцюгів імуноглобулінів у сироватці (sFLC, serum Free Light Chains) є особливо інформативним при олігосекреторних і несекреторних формах ММ [18].

Цитогенетичне дослідження — FISH (Fluorescence in situ hybridization) — виконується на аспіраті кісткового мозку і виявляє транслокації та делеції, що мають прогностичне значення: $t(4;14)$, $t(14;16)$, $del(17p)$ визначають високий ризик [10].

Морфологічні методи

Аспіраційна біопсія кісткового мозку залишається обов'язковим етапом діагностики. Підвищення кількості клональних плазмоцитів понад 10 % від загальної клітинної популяції кісткового мозку є одним з ключових критеріїв ММ за IMWG [15]. Трепанобіопсія дає змогу оцінити архітектоніку кісткового мозку та виявити вогнищеві ураження.

Інструментальні методи візуалізації

Оцінка ураження кісток скелета є ще одним обов'язковим етапом діагностики. Класичним методом протягом десятиліть була рентгенографія скелета у вигляді «скелетного огляду» (skeletal survey), однак її чутливість обмежена — типові «вибиті» вогнища в кістках черепа, хребті та тазі стають видимими лише при значному руйнуванні кісткової маси (близько 30 %).

У сучасних рекомендаціях IMWG (2019) перевага надається таким методам [19]:

- низькодозова комп'ютерна томографія всього тіла (whole-body low-dose CT) — забезпечує високу чутливість при відносно невеликій променевій дозі;
- магнітно-резонансна томографія (МРТ) — є золотим стандартом для виявлення вогнищевих уражень кісткового мозку і компресії спинного мозку;

- позитронно-емісійна томографія в комбінації з КТ (ПЕТ-КТ з ^{18}F -ФДГ) — використовується для оцінки активності пухлини та виявлення позаскелетних плазмоцитом.

Діагностичний алгоритм

Послідовність застосування описаних методів при підозрі на ММ формалізована у клінічних настановах. Загальна схема представлена на рисунку 1.4.

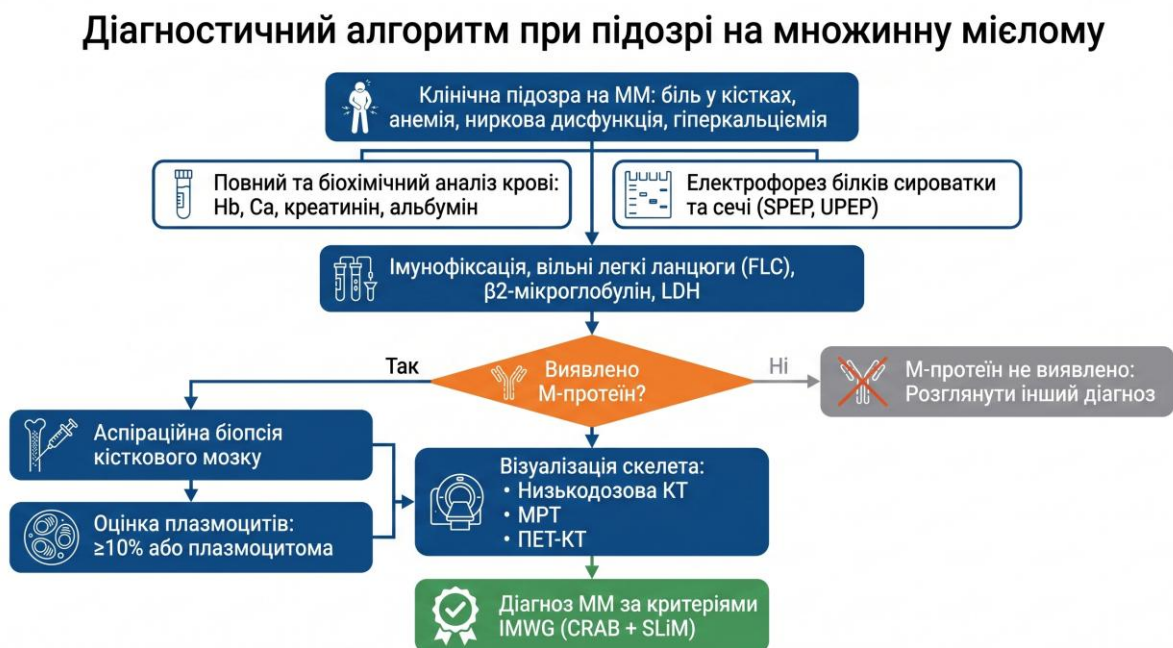


Рисунок 1.4 – Діагностичний алгоритм при підозрі на множинну мієлому²

Узагальнення основних діагностичних методів з оцінкою їх ролі в загальному процесі наведено в таблиці 1.2.

² Зображення створено за допомогою чат-боту з генеративним штучним інтелектом Chat GPT

Таблиця 1.2 – Лабораторні та інструментальні методи діагностики множинної мієломи

<i>Категорія</i>	<i>Метод</i>	<i>Що оцінює</i>	<i>Чутливість/ специфічність</i>	<i>Роль у задачі ML</i>
Гематологія	Загальний аналіз крові	Анемія, цитопенії	Висока чутливість, низька специфічність	Базові ознаки
Біохімія	Креатинін, Са, LDH, альбумін	Функція нирок, активність пухлини	Висока — для стадіювання	Базові ознаки
Білкові маркери	SPEP / UPEP	М-протеїн в крові/сечі	Висока специфічність	Підтверджуючий тест
Білкові маркери	Імунофіксація (IFE)	Тип моноклонального білка	Найвища специфічність	Підтверджуючий
Білкові маркери	sFLC	Вільні легкі ланцюги	Чутливий до олігосекреторних форм	Розширений
Цитогенетика	FISH	Аберації високого ризику	Прогностична	Прогноз, не діагноз
Морфологія	Біопсія КМ	Кількість плазмочитів	Стандарт підтвердження	Інвазивна, не для скринінгу
Візуалізація	Low-dose CT / MPT / ПЕТ-КТ	Кісткові ураження	Висока чутливість	Кінцева верифікація

Обмеження поточних підходів і місце машинного навчання

Описаний діагностичний алгоритм є точним, але має кілька обмежень з погляду ранньої діагностики. По-перше, він активується лише після клінічної підозри — тобто пацієнт уже потрапив до лікаря зі скаргами, типовими для ММ. Друга проблема — багатоетапність і тривалість процесу: повне обстеження може зайняти кілька тижнів. По-третє, для частини етапів (зокрема цитогенетики, ПЕТ-КТ) необхідне спеціалізоване обладнання, доступне переважно у великих гематологічних центрах.

З погляду задачі цієї роботи, ключове спостереження таке: значна частина показників, що використовуються при діагностиці ММ, входить до звичайного клініко-лабораторного обстеження — загальний і біохімічний аналізи крові, базові біохімічні маркери. Ці дані доступні навіть у поліклінічних умовах і можуть бути використані для попередньої оцінки ризику ММ ще до отримання результатів більш складних досліджень. Це створює природне поле для застосування методів машинного навчання: алгоритм, навчений на сукупності рутинних показників, може виступати як інструмент скринінгу, який сигналізує лікареві про необхідність направлення пацієнта на поглиблене обстеження.

1.3 Огляд методів інтелектуального аналізу даних у медицині

Поняття машинного навчання

Інтелектуальний аналіз даних (Data Mining, ІАД) і машинне навчання (Machine Learning, ML) — це сукупність методів, які дозволяють виявляти закономірності у великих масивах даних і будувати на їх основі прогностичні або описові моделі. На відміну від традиційного програмування, де алгоритм явно задається людиною, у машинному навчанні модель «навчається» сама — на основі історичних даних, у яких відомі правильні відповіді. У класичному визначенні Митчелла, програма вважається такою, що навчається, якщо її ефективність на певному класі задач зростає з накопиченням досвіду [16].

У медицині ML використовується для широкого спектру задач: класифікація пацієнтів за діагнозом, прогнозування ризику ускладнень, оцінка ефективності терапії, аналіз медичних зображень, виявлення нових біомаркерів [21, 22]. Перевагою таких підходів є здатність одночасно обробляти десятки і сотні параметрів, виявляючи неочевидні взаємозв'язки, які важко зафіксувати людським оком навіть досвідченому лікарю.

Класифікація методів машинного навчання

За способом подання даних і характером навчання методи ML традиційно розподіляють на три великі групи (рис. 1.5).



Рисунок 1.5 – Класифікація методів машинного навчання з виділенням алгоритмів, використаних у роботі

Контрольоване навчання (supervised learning) передбачає наявність розмічених даних — кожному прикладу зіставлено правильну відповідь. Залежно від типу цієї відповіді розрізняють задачі класифікації (мітка категоріальна — наприклад, «ММ / не-ММ») і регресії (мітка числова — наприклад, рівень моноклонального білка). До цієї групи належать логістична регресія, метод опорних векторів (SVM), дерева рішень та їх ансамблі (Random Forest, XGBoost, CatBoost, LightGBM), а також нейронні мережі.

Неконтрольоване навчання (unsupervised learning) працює з нерозміченими даними — модель самостійно шукає структуру в інформації. Основні підкласи — кластеризація (виявлення груп схожих об'єктів) та

зниження розмірності (PCA, t-SNE, UMAP). У медицині такі методи використовуються для виявлення підтипів захворювань без апіорної категоризації.

Навчання з підкріпленням (reinforcement learning) передбачає взаємодію агента з середовищем і отримання винагороди за правильні дії. У клінічних застосуваннях найбільше досліджується для оптимізації режимів лікування і керування інсуліновими помпами.

Задача діагностики ММ за клініко-лабораторними показниками належить до бінарної класифікації з контрольованим навчанням — модель отримує набір показників пацієнта на вході і повертає клас (наявність або відсутність ознак ММ) на виході.

Особливості медичних табличних даних

Медичні дані мають низку специфічних рис, які впливають на вибір моделей і способів їх валідації [22, 23].

Незбалансованість класів. У клінічних задачах позитивні випадки (з підтвердженою патологією) зазвичай становлять меншість. Для ММ типове співвідношення хворі/контроль становить приблизно 1:3 або 1:5 у скринінгових вибірках. У роботі використовується вибірка з розподілом 298:227 ($\approx 57:43$ %), що є відносно збалансованим випадком, проте і тут необхідне зважування класів або застосування методів синтетичного семплінгу (SMOTE) за потреби.

Пропуски значень. У реальних клінічних наборах окремі аналізи можуть бути не виконані для частини пацієнтів. Стратегії боротьби з пропусками включають видалення зразків, імпутацію медіаною або середнім, складніші методи — kNN-імпутацію, MICE-алгоритм або вбудовану обробку пропусків в окремих моделях (XGBoost, CatBoost).

Висока розмірність простору ознак. У табличних медичних наборах кількість показників нерідко перевищує сто. Не всі ознаки однаково інформативні, частина може бути сильно корельованою, частина — створювати шум. Тому етап відбору ознак (feature selection) є обов'язковим.

Виділяють три великі групи методів: фільтрові (на основі статистичних мір), обгорткові (на основі продуктивності модель-кандидатів), вбудовані (інтегровані у сам алгоритм навчання) [9].

Інтерпретованість. На відміну від багатьох інженерних задач, у медицині недостатньо, щоб модель давала правильну відповідь — лікар має розуміти, на чому ця відповідь базується. Тому моделі типу «чорна скринька» (наприклад, глибокі нейронні мережі) часто доповнюються інструментами пояснюваності — SHapley Additive exPlanations (SHAP) -значеннями, Local Interpretable Model-agnostic Explanations (LIME), аналізом feature importance [25]. Ансамблі дерев рішень мають перевагу над простішими нейромережними архітектурами завдяки вбудованій оцінці важливості ознак.

Типовий пайплайн обробки медичних даних

Послідовність обробки клінічних даних від збору до прийняття рішення лікарем зображена на рис. 1.6.

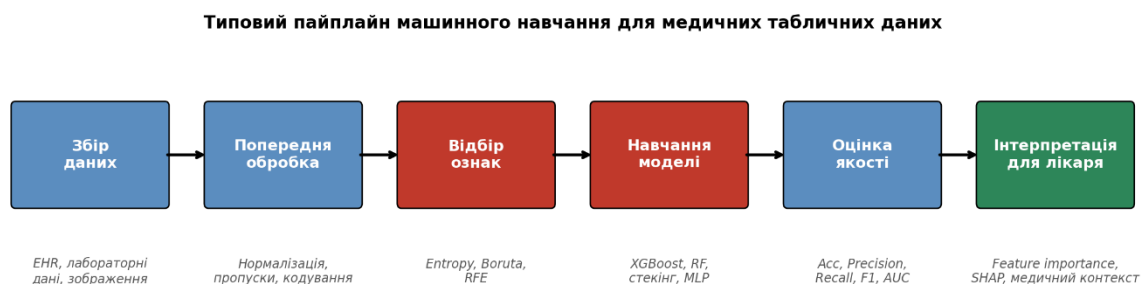


Рисунок 1.6 – Типовий пайплайн машинного навчання для медичних табличних даних

Червоним кольором виділено два ключові етапи, що становлять предмет цього дослідження: відбір ознак і навчання моделі. Зелений блок інтерпретації — обов'язкова заключна стадія для медичних застосувань.

Огляд застосувань ML в онкології

За останнє десятиліття ML-методи продемонстрували значні успіхи в онкології. Найбільш розвиненим напрямом є аналіз медичних зображень — згорткові нейронні мережі здатні класифікувати дерматологічні пухлини з точністю на рівні сертифікованих дерматологів [22], виявляти підозрілі вузлики на КТ легень [23], розпізнавати інвазивні форми раку молочної залози на мамографіях.

Інший важливий напрям — аналіз табличних клініко-лабораторних даних. У роботі Mosquera Orgueira et al. на даних 2105 пацієнтів з ММ продемонстровано, що моделі XGBoost і Random Forest на основі рутинних клінічних показників і експресії генів дозволяють прогнозувати загальну виживаність з точністю, яка перевершує класичні системи стадування ISS [24]. У Farswan et al. застосування ансамблевих моделей на даних 1003 пацієнтів дозволило побудувати персоналізовані індекси ризику прогресування MGUS у ММ. Загальна тенденція — поступовий перехід від простих логістичних моделей до сучасних ансамблів і нейромереж, причому саме ансамблеві методи дерев рішень показують найкращий компроміс між точністю та інтерпретованістю на табличних даних.

Підсумок підрозділу

Машинне навчання у медицині — це інструмент, який добре доповнює, але не замінює традиційну клінічну оцінку. Для табличних клініко-лабораторних даних, з якими працює ця робота, найбільш ефективним підходом є контрольоване навчання у задачі бінарної класифікації, з обов'язковим попереднім відбором ознак і подальшою клінічною інтерпретацією отриманих результатів. Саме така архітектура обрана для розробки системи інтелектуальної діагностики ММ — її теоретичні засади детально розглянуто у Розділі 2.

1.4 Огляд попередніх досліджень

Застосування методів машинного навчання до задач, пов'язаних з множинною мієломою, активно розвивається з кінця 2010-х років. Умовно ці роботи можна поділити на три напрями: діагностика і скринінг (відрізнення хворих від здорових), прогнозування перебігу (оцінка виживаності та ризику прогресування), а також підбір терапії. Для цієї роботи найбільший інтерес становить перший напрям, тому розгляд починається саме з нього.

Діагностика та скринінг

Найближчою за постановкою до цієї роботи є дослідження Cai та співавторів (2024), виконане на даних 465 пацієнтів з ММ і 150 здорових осіб [27]. Автори використали відбір ознак методом LASSO, після чого побудували три моделі — логістичну регресію, метод опорних векторів і випадковий ліс — на параметрах загального аналізу крові (CBC, Complete Blood Count) і даних клітинної популяції (CPD). Найкращий результат показав випадковий ліс: AUC 0,956 на навчальній, 0,892 на валідаційній і 0,875 на тестовій вибірках. Цінним є те, що автори застосували тест DeLong для оцінки статистичної значущості відмінностей між моделями — саме той підхід, який рекомендований у рецензії до нашої роботи. Висновок дослідження прямо співзвучний нашій меті: модель випадкового лісу може слугувати допоміжним інструментом швидкого скринінгу ММ.

За даними оглядового аналізу застосувань ШІ в гематології [29], модель на основі градієнтного бустингу (GBDT) досягла AUC 0,929 для виявлення ММ, причому навчання проводилося лише на шести рутинних показниках — гемоглобін, креатинін, кальцій сироватки, альбумін, загальний білок і співвідношення альбумін/глобулін. Це підтверджує ключову гіпотезу нашої роботи: для попередньої оцінки ризику ММ достатньо звичайних лабораторних аналізів, які доступні навіть на рівні поліклініки.

Peng та співавтори (2023) побудували модель прогнозування інфекційних ускладнень у вперше діагностованих пацієнтів з ММ на основі 564 випадків [28]. Відбір ознак методом LASSO виділив 15 ключових змінних, серед яких — вік, гемоглобін, тромбоцити, альбумін, креатинін, лактатдегідрогеназа, β 2-мікроглобулін. Найкращою моделлю став XGBoost з AUROC 0,866, а для інтерпретації результатів автори застосували SHAP-аналіз. Перелік відібраних ними ознак значною мірою перетинається з тими, що були відібрані ентропійним методом у нашому наборі даних (гемоглобін, креатинін, показники червоної крові), що є непрямым підтвердженням адекватності нашого відбору.

Прогнозування перебігу та виживаності

Mosquera Orgueira та співавтори (2021) на даних 2105 пацієнтів продемонстрували, що ансамблі дерев рішень (XGBoost, Random Forest) на основі клінічних показників та експресії генів прогнозують загальну виживаність точніше за класичну систему стадування ISS [24]. Bao та співавтори (2022) на мультицентрових даних літніх пацієнтів з ММ підтвердили перевагу ансамблевих моделей у задачі прогнозу виживаності [30]. Ren та співавтори (2023) розробили прогностичну сигнатуру на основі генів убіквітин-протеасомного шляху з використанням LASSO-регресії та машинного навчання [31]. Спільною рисою цих робіт є послідовне використання LASSO або інших методів відбору ознак перед навчанням моделі — що ще раз підкреслює критичну роль етапу feature selection.

Підбір терапії та комплексні системи

Окрему групу складають роботи з підбору хіміотерапії. У дослідженні, опублікованому в Evolutionary Intelligence (2023), порівнювалися чотири моделі для розпізнавання протоколу хіміотерапії ММ: випадковий ліс досяг точності 98,63 %, XGBoost — 95,89 %, Multi-Layer Perceptron (MLP) — 92,23 %, SVM — 87,21 % [33]. Автори застосували одразу три інструменти пояснюваності (permutation importance, Kernel SHAP, LIME). Hussain та

співавтори (2024) у прj Digital Medicine запропонували комплексну AI-систему спільного прогнозування подій і моделювання динаміки захворювання [32].

Узагальнене порівняння розглянутих робіт наведено в таблиці 1.3 та на рисунку 1.7.

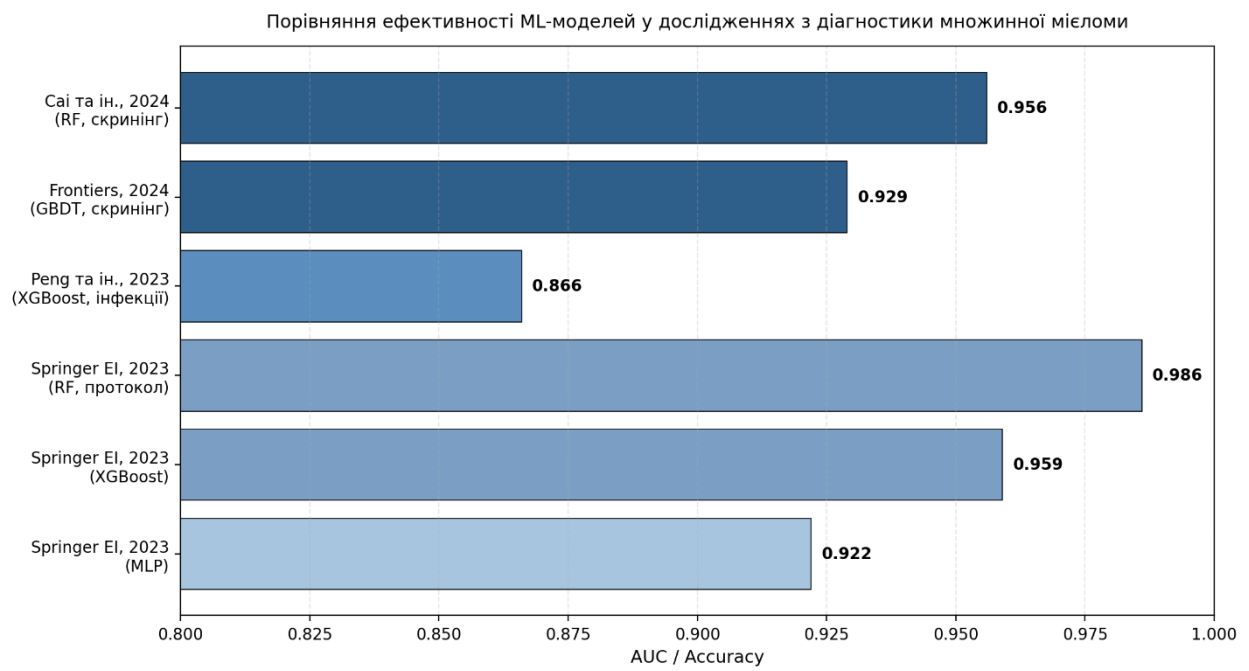


Рисунок 1.7 – Порівняння ефективності ML-моделей у дослідженнях діагностики множинної мієломи (за даними [27, 28, 29, 33])

Таблиця 1.3 – Огляд попередніх досліджень із застосування ML для задач, пов'язаних з ММ

Дослідження, рік	Задача	Вибірка	Відбір ознак	Найкраща модель	Результат
Cai та ін., 2024 [27]	Скринінг ММ	465 + 150 контр.	LASSO	Random Forest	AUC 0,875 (тест)
Frontiers, 2024 [29]	Скринінг ММ	—	—	GBDT	AUC 0,929
Peng та ін., 2023 [28]	Прогноз інфекцій	564	LASSO (15 ознак)	XGBoost	AUROC 0,866

Кінець таблиці 1.3

Дослідження, рік	Задача	Вибірка	Відбір ознак	Найкраща модель	Результат
Mosquera Orgueira, 2021 [24]	Виживаність	2105	вбудовані	XGBoost / RF	вище за ISS
Bao та ін., 2022 [30]	Виживаність	мультицентр	—	ансамблі	—
Springer EI, 2023 [33]	Протокол терапії	—	—	Random Forest	Асс 98,63 %
Наша робота	Скринінг ММ	525	Entropy + Boruta + RFE	XGBoost / CatBoost	(експеримент)

Висновок з огляду

Аналіз літератури дозволяє зробити кілька висновків. По-перше, ансамблеві методи дерев рішень (Random Forest, XGBoost, GBDT) є домінуючим вибором у задачах, пов'язаних з ММ, і стабільно показують найкращі результати на табличних даних. Ансамблевий підхід підтверджує свою ефективність і в суміжних задачах медичної діагностики, зокрема в інтелектуальній оцінці активності легеневого туберкульозу на основі ансамблю глибоких згорткових нейронних мереж [49]. По-друге, етап відбору ознак присутній майже у всіх роботах, причому переважно використовується метод LASSO; ентропійні методи, Boruta та RFE застосовуються рідше, а їх систематичне порівняння в задачі ММ практично відсутнє. По-третє, лише частина робіт проводить статистичну оцінку значущості відмінностей між моделями.

Саме тут полягає внесок цієї роботи. На відміну від розглянутих досліджень, у ній виконується систематичне порівняння трьох різнотипних методів відбору ознак (фільтрового — ентропійного, обгорткового — RFE, та вбудованого — Boruta) у поєднанні з п'ятьма сучасними класифікаторами, з обов'язковою оцінкою статистичної значущості та клінічною інтерпретацією

відібраних ознак. Такий комплексний порівняльний підхід для задачі скринінгу ММ у доступній літературі не представлений.

1.5 Постановка задачі

На основі проведеного аналізу предметної області сформулюємо формальну постановку задачі дослідження.

Змістовна постановка

Необхідно розробити програмну систему, яка на основі клініко-лабораторних показників пацієнта здатна автоматично віднести його до однієї з двох категорій: «наявні ознаки множинної мієломи» або «ознаки відсутні». Система має слугувати допоміжним інструментом скринінгу, що сигналізує лікареві про необхідність направлення пацієнта на поглиблене обстеження.

Формальна постановка

Нехай задано множину об'єктів (пацієнтів) $X = \{x_1, x_2, \dots, x_n\}$, де кожен пацієнт описується вектором ознак $x_i = (f_1, f_2, \dots, f_m)$, у якому f_j — значення j -го клініко-лабораторного показника. Кожному об'єкту відповідає мітка $y_i \in \{0, 1\}$, де $y_i = 1$ означає наявність ММ, $y_i = 0$ — її відсутність.

Потрібно побудувати класифікатор — відображення $a: X \rightarrow \{0, 1\}$, яке для довільного вектора ознак пацієнта повертає прогнозовану мітку класу. Якість класифікатора оцінюється за набором метрик (Accuracy, Precision, Recall, Specificity, F1-score, AUC), а критерієм оптимальності є максимізація узагальненої здатності правильно розрізняти класи на нових, не використаних під час навчання, даних.

Задача розв'язується у два етапи:

Відбір ознак. З вихідного простору $m = 213$ ознак необхідно сформувати скорочений інформативний підпростір $m' \ll m$, застосувавши три методи відбору (Information Gain, Boruta, RFE) та порівнявши їх результати.

Класифікація. На відібраному підпросторі ознак необхідно навчити та порівняти п'ять моделей (XGBoost, CatBoost, стекінг, багатогілковий ансамбль, MLP), визначивши оптимальну комбінацію «метод відбору + класифікатор».

Вхідні та вихідні дані

Вхідні дані: набір з 525 пацієнтів, кожен з яких описується 213 клініко-лабораторними показниками (загальний і біохімічний аналізи крові, електроліти, білкові фракції, кардіомаркери), нормалізованими у діапазоні [0, 1]. Розподіл класів: 298 пацієнтів з ММ ($Y=1$) і 227 без ММ ($Y=0$).

Вихідні дані: навчена класифікаційна модель, що повертає прогнозовану мітку класу, а також ранжований за важливістю перелік клініко-лабораторних показників.

Обмеження та припущення

Дослідження має такі обмеження: розмір вибірки (525 пацієнтів) є помірним для медичних задач; дані зібрані в одному медичному центрі (Люблін, Польща), що обмежує безпосереднє узагальнення на інші популяції; робота розглядає лише бінарну класифікацію без розрізнення стадій ММ. Ці обмеження враховуються при інтерпретації результатів і визначають напрями подальших досліджень.

Висновки до розділу 1

У першому розділі проаналізовано предметну область дослідження. Розглянуто множинну міселому як медичну задачу: її епідеміологію, клінічну картину, системи стадування та проблему пізньої діагностики. Показано, що значна частина діагностичних показників ММ входить до рутинного клініко-лабораторного обстеження, що створює основу для застосування методів машинного навчання. Проаналізовано існуючі підходи до діагностики ММ та методи інтелектуального аналізу даних у медицині. Виконано огляд

попередніх досліджень, який показав, що ансамблеві методи домінують у цій галузі, але систематичне порівняння різнотипних методів відбору ознак у задачі скринінгу ММ практично відсутнє. На основі аналізу сформульовано формальну постановку задачі, визначено вхідні та вихідні дані, обмеження та припущення. Теоретичні засади обраних методів детально розглядаються у наступному розділі.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ ДЛЯ КЛАСИФІКАЦІЇ ММ

2.1 Попередня обробка та нормалізація даних

Роль попередньої обробки

Якість прогностичної моделі залежить не лише від обраного алгоритму, а й значною мірою від якості вхідних даних. У літературі поширений принцип «garbage in — garbage out»: модель, навчена на неякісних даних, дає ненадійні результати незалежно від своєї складності. Реальні клініко-лабораторні дані рідко придатні для безпосереднього використання — вони містять пропуски, викиди, різномірні шкали вимірювання, а класи в них зазвичай незбалансовані. Тому етап попередньої обробки (data preprocessing) є обов’язковою частиною будь-якого пайплайну машинного навчання [22]. Загальна послідовність етапів обробки наведена на рисунку 2.1.

Етапи попередньої обробки клініко-лабораторних даних

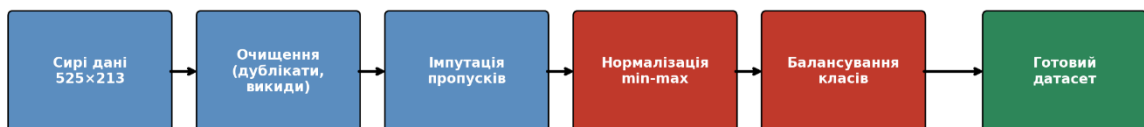


Рисунок 2.1 – Етапи попередньої обробки клініко-лабораторних даних

Очищення даних

Перший етап — очищення (data cleaning). Воно включає виявлення і видалення дублікатів записів, перевірку коректності типів даних, виявлення аномальних значень (викидів). Викиди у медичних даних потребують обережного поводження: на відміну від технічних задач, екстремальне

значення показника може бути не помилкою вимірювання, а ознакою важкого стану пацієнта. Наприклад, рівень креатиніну, що в кілька разів перевищує норму, є клінічно значущим для ММ і не повинен механічно видалятися. Тому для медичних даних застосовують не агресивне відсікання викидів, а їх позначення з подальшим аналізом.

Обробка пропущених значень

У реальних наборах даних окремі показники нерідко відсутні — аналіз не призначався, не був виконаний або результат загублений. Існує кілька стратегій боротьби з пропусками.

Найпростіша — видалення записів або ознак з пропусками. Вона прийнятна, лише якщо частка пропусків невелика, інакше призводить до значної втрати інформації. Альтернатива — імпутація (заповнення) пропущених значень. Найчастіше застосовують імпутацію медіаною для числових ознак, оскільки медіана стійка до викидів, на відміну від середнього арифметичного. Більш складні методи — kNN-імпутація (заповнення на основі найближчих за іншими ознаками сусідів) та багатовимірна імпутація рівняннями MICE. Окремі сучасні алгоритми, зокрема XGBoost і CatBoost, мають вбудований механізм обробки пропусків і не потребують попередньої імпутації [5].

Нормалізація даних

Клінічні показники вимірюються у різних одиницях і мають різні діапазони: рівень гемоглобіну вимірюється в десятках г/л, кількість лейкоцитів — одиницями $\times 10^9/\text{л}$, а рівень NT-pro BNP може сягати тисяч пг/мл. Якщо подати такі дані без перетворення, ознаки з більшими абсолютними значеннями домінуватимуть під час навчання, особливо для методів, чутливих до масштабу (SVM, kNN, нейронні мережі). Для усунення цього застосовують масштабування ознак.

Найпоширеніший метод — мінімаксна нормалізація (min-max normalization), що приводить значення кожної ознаки до діапазону $[0, 1]$:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (2.1)$$

де x — вихідне значення ознаки, x_{min} та x_{max} — її мінімальне та максимальне значення у вибірці, x' — нормалізоване значення.

Альтернативний метод — стандартизація (z-нормалізація), яка приводить ознаку до нульового середнього та одиничного стандартного відхилення:

$$z = \frac{x - \mu}{\sigma} \quad (2.2)$$

де μ — середнє значення ознаки, σ — її стандартне відхилення.

Вибір методу залежить від алгоритму та розподілу даних. Мінімаксна нормалізація зберігає форму вихідного розподілу і зручна, коли значення обмежені відомим діапазоном. Стандартизація краще підходить для даних, близьких до нормального розподілу, і менш чутлива до викидів. У даній роботі вхідний набір даних попередньо нормалізовано мінімаксним методом у діапазон $[0, 1]$, що відображено у самій назві файлу даних.

Кодування категоріальних ознак

Більшість алгоритмів працює лише з числовими даними, тому категоріальні ознаки (стать, тип патології) потребують кодування. Основні підходи — однокодове кодування (one-hot encoding), що створює окрему бінарну ознаку для кожної категорії, міткове кодування (label encoding) для порядкових змінних, та цільове кодування (target encoding). Алгоритм CatBoost відрізняється тим, що має вбудовану ефективну обробку категоріальних ознак, що й відображено в його назві (Categorical Boosting).

Балансування класів

Незбалансованість класів — поширена проблема медичних даних. Якщо позитивний клас значно менший за негативний, модель може досягати високої формальної точності, просто завжди передбачаючи більший клас, але при цьому пропускати хворих пацієнтів. Для боротьби з цим застосовують методи

семплінгу: випадкове збільшення меншого класу (oversampling), випадкове зменшення більшого (undersampling) та синтетичну генерацію прикладів.

Найвідоміший метод синтетичної генерації — SMOTE (Synthetic Minority Over-sampling Technique) [34]. Замість простого дублювання прикладів меншого класу він створює нові синтетичні зразки вздовж відрізків, що з'єднують наявні приклади меншого класу з їх найближчими сусідами:

$$x_{new} = x_i + \lambda \cdot (x_{zi} - x_i) \quad (2.3)$$

де x_i — приклад меншого класу, x_{zi} — один з його найближчих сусідів того ж класу, $\lambda \in [0, 1]$ — випадкове число. Принцип роботи SMOTE проілюстровано на рисунку 2.2.

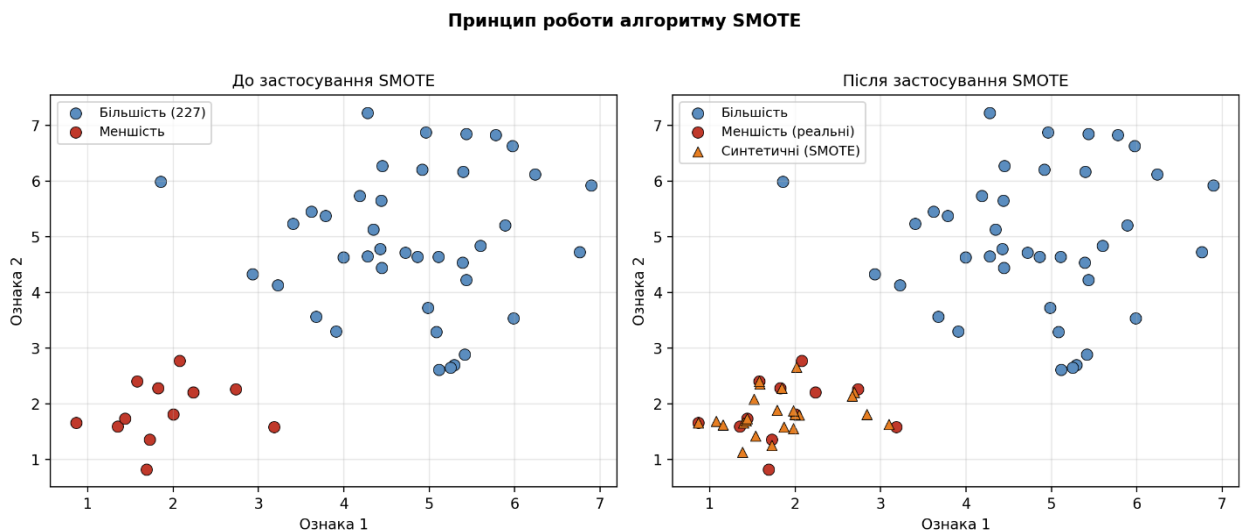


Рисунок 2.2 – Принцип роботи алгоритму SMOTE: генерація синтетичних прикладів меншого класу

У наборі даних цієї роботи розподіл класів становить 298 проти 227 (приблизно 57 % до 43 %), що є відносно збалансованим випадком. Для таких даних агресивне застосування SMOTE не є обов'язковим; альтернативою є зважування класів (class weighting), коли алгоритму під час навчання призначається більший штраф за помилку на меншому класі. Цей підхід реалізований, зокрема, через параметр `scale_pos_weight` в XGBoost.

Розбиття вибірки та крос-валідація

Для об'єктивної оцінки моделі дані поділяють на навчальну, валідаційну та тестову підмножини. Навчальна використовується для підбору параметрів моделі, валідаційна — для налаштування гіперпараметрів, тестова — для фінальної оцінки на даних, які модель ніколи не бачила. Типове співвідношення — 70/15/15 або 80/10/10.

Для невеликих вибірок одноразове розбиття дає нестабільну оцінку, що сильно залежить від конкретного поділу. Надійнішим методом є k -кратна крос-валідація (k -fold cross-validation): дані розбивають на k рівних частин, після чого модель навчають k разів, щоразу використовуючи одну частину для тестування, а решту — для навчання (рис. 2.3). Підсумкова оцінка усереднюється за всіма k ітераціями, що зменшує залежність від випадкового поділу. У медичних задачах застосовують стратифіковану крос-валідацію, яка зберігає пропорцію класів у кожній частині — це особливо важливо при незбалансованих даних.

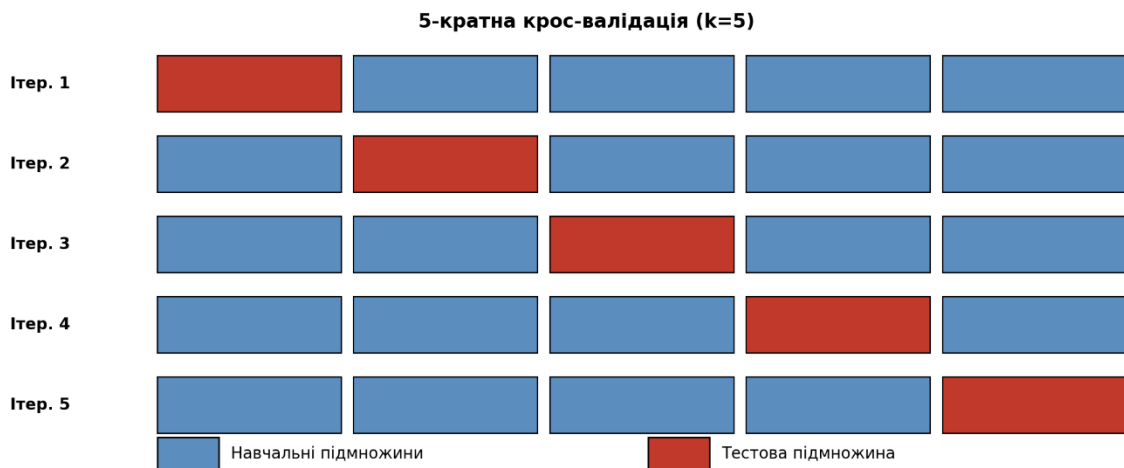


Рисунок 2.3 – Схема 5-кратної крос-валідації

У цій роботі доцільно використання стратифікованої k -кратної крос-валідації ($k=5$ або $k=10$), що забезпечує надійну оцінку якості моделей на

вибірці помірного розміру і дозволяє коректно порівнювати алгоритми між собою з оцінкою статистичної значущості відмінностей.

2.2 Методи відбору ознак

Загальна класифікація

Відбір ознак (feature selection) — це процес виокремлення з повного набору показників тих, що несуть найбільше інформації для розв’язання задачі, з одночасним відкиданням нерелевантних і надлишкових. Мета відбору двояка: підвищити якість і узагальнювальну здатність моделі та зробити її простішою для інтерпретації. У медичних застосуваннях друга мета не менш важлива за першу, адже лікар має розуміти, на яких показниках ґрунтується рішення системи [25].

За способом взаємодії з моделлю класифікації методи відбору поділяють на три категорії (рис. 2.4): фільтрові, обгорткові та вбудовані [36].



Рисунок 2.4 – Класифікація методів відбору ознак

Фільтрові методи оцінюють інформативність ознак на основі статистичних мір, незалежно від моделі класифікації. Вони швидкі та обчислювально дешеві, але не враховують взаємодії між ознаками. Обгорткові методи перебирають підмножини ознак, оцінюючи якість конкретної моделі на кожній з них — це точніше, але значно повільніше. Вбудовані методи виконують відбір безпосередньо під час навчання моделі. У роботі використано по одному представнику з кожної групи: Information Gain (фільтровий), RFE (обгортковий) та Boruta (на основі вбудованої важливості ансамблю).

Фільтровий метод: інформаційний приріст (Information Gain)

Метод інформаційного приросту ґрунтується на теоретико-інформаційному понятті ентропії, введеному Шенноном [35]. Ентропія Шеннона та похідні від неї метрики (зокрема інформаційний приріст) широко застосовуються для відбору інформативних ознак у задачах медичного інтелектуального аналізу даних — як у класифікації, так і для зменшення розмірності просторів генних експресій [48]. Ентропія є мірою невизначеності (хаотичності) у розподілі класів. Для вибірки S з двома класами вона обчислюється так:

$$H(S) = - \sum_{i=1}^c p_i \log_2 p_i \quad (2.4)$$

де p_i — частка прикладів класу i у вибірці, c — кількість класів. Ентропія максимальна, коли класи представлені порівну (повна невизначеність), і дорівнює нулю, коли всі приклади належать одному класу.

Інформаційний приріст ознаки A показує, наскільки зменшується ентропія вибірки після її розбиття за значеннями цієї ознаки:

$$IG(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v) \quad (2.5)$$

де S_v — підмножина прикладів, для яких ознака A має значення v , $|S_v|$ та $|S|$ — кількості елементів у відповідних множинах. Чим більший

інформаційний приріст, тим інформативніша ознака для розрізнення класів. Ознаки ранжуються за значенням IG, після чого відбираються ті, що мають найвищі показники. Саме цей підхід застосовано для формування набору Dataset-Cleaned-Entropy — відібрані десять ознак мають найвищий інформаційний приріст серед усіх 213.

Перевагою методу є швидкодія та незалежність від класифікатора, недоліком — оцінювання кожної ознаки окремо, без урахування їх спільного впливу.

Обгортковий метод: рекурсивне усунення ознак (RFE)

Метод рекурсивного усунення ознак (Recursive Feature Elimination, RFE) запропонований Гайоном та співавторами [9]. Це ітеративний обгортковий метод, який працює за принципом послідовного відкидання найменш важливих ознак (рис. 2.5).

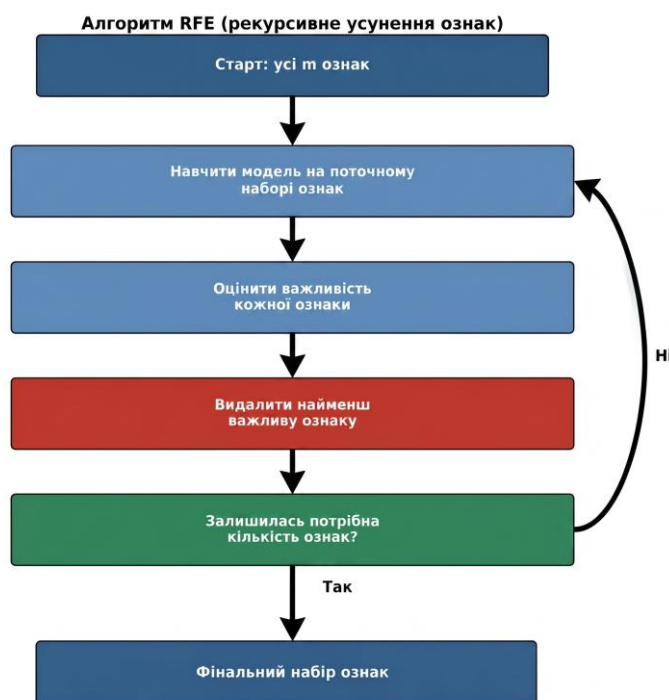


Рисунок 2.5 – Блок-схема алгоритму рекурсивного усунення ознак (RFE)

Алгоритм складається з таких кроків. Спочатку модель (наприклад, SVM або випадковий ліс) навчається на повному наборі ознак. Потім обчислюється важливість кожної ознаки — для лінійних моделей це абсолютні значення вагових коефіцієнтів, для деревних — вбудована оцінка важливості. Ознака з найменшою важливістю видаляється, після чого модель перенавчається на скороченому наборі. Процес повторюється, доки не залишиться задана кількість ознак. Перевагою RFE є врахування спільного впливу ознак через використання моделі; недоліком — висока обчислювальна вартість, оскільки модель доводиться навчати багато разів.

Метод на основі ансамблю: Boruta

Алгоритм Boruta, розроблений Курсою та Рудніцьким [8], реалізує концепцію відбору всіх релевантних ознак (all-relevant feature selection) на основі важливості, отриманої з випадкового лісу. На відміну від методів, що шукають мінімальний достатній набір, Boruta прагне знайти всі ознаки, які мають хоч якийсь зв'язок з цільовою змінною. Принцип роботи показано на рисунку 2.6.



Рисунок 2.6 – Принцип роботи алгоритму Boruta

Ключова ідея методу — введення «тіньових ознак» (shadow features). Для кожної реальної ознаки створюється її копія, значення якої випадково перемішуються між пацієнтами, що руйнує будь-який зв'язок з цільовою змінною. Розширений набір (реальні + тіньові ознаки) подається на вхід випадковому лісу, який обчислює важливість усіх ознак. Реальна ознака вважається значущою, якщо її важливість статистично перевищує максимальну важливість серед усіх тіньових ознак (MZSA — Maximum Z-score of Shadow Attribute). Процедура повторюється багато разів, і за результатами серії випробувань кожна ознака отримує статус «підтверджена», «відхилена» або «невизначена». Перевагою Boruta є надійність і стійкість до випадковостей завдяки статистичному тестуванню, недоліком — значна обчислювальна вартість через багаторазове навчання випадкового лісу.

Порівняння методів та обґрунтування вибору

Три обрані методи представляють принципово різні підходи до відбору, що робить їх порівняння методологічно цінним. Узагальнення наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння методів відбору ознак, використаних у роботі

<i>Характеристика</i>	<i>Information Gain</i>	<i>RFE</i>	<i>Boruta</i>
Категорія	Фільтровий	Обгортковий	На основі ансамблю
Залежність від моделі	Незалежний	Залежний	На основі RF
Урахування взаємодій ознак	Ні	Так	Так
Обчислювальна вартість	Низька	Висока	Висока
Тип результату	Ранжування	Підмножина заданого розміру	Підтверджені/відхилені
Стійкість до шуму	Середня	Середня	Висока

Використання трьох різнотипних методів дозволяє не покладатися на обмеження жодного окремого підходу. Фільтровий Information Gain дає швидку базову оцінку, обгортковий RFE враховує взаємодію ознак у контексті конкретної моделі, а Boruta забезпечує статистично обґрунтований відбір усіх релевантних ознак. Зіставлення наборів ознак, відібраних трьома методами, дає змогу виявити стабільно важливі показники — ті, що потрапляють у відбір незалежно від методу, заслуговують найбільшої довіри з клінічної точки зору.

2.3 Ансамблеві методи класифікації

Концепція ансамблювання

Ансамблеві методи (ensemble methods) ґрунтуються на ідеї, що об'єднання прогнозів кількох моделей дає точніший і стійкіший результат, ніж будь-яка окрема модель. Це пов'язано з компромісом між зміщенням і дисперсією (bias-variance tradeoff): одна складна модель схильна до перенавчання (висока дисперсія), а проста — до недонавчання (високе зміщення). Об'єднання багатьох моделей дозволяє зменшити дисперсію без істотного зростання зміщення [4]. Розрізняють дві основні стратегії ансамблювання — беггінг і бустинг (рис. 2.7).

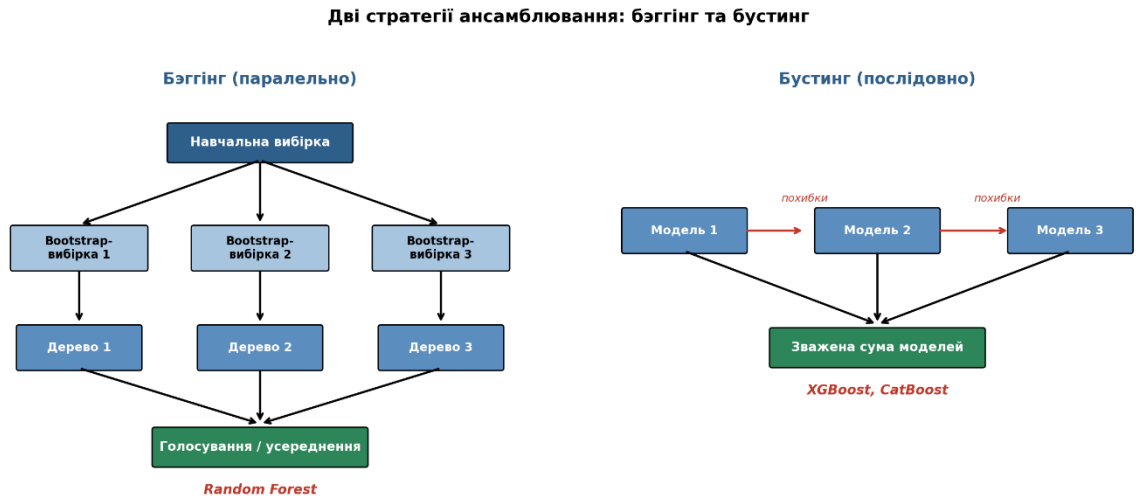


Рисунок 2.7 – Дві стратегії ансамблювання: беггінг та бустинг

Дерева рішень як базова модель

Більшість ансамблів для табличних даних будується на основі дерев рішень. Дерево рішень — це ієрархічна структура, у якій кожен внутрішній вузол відповідає перевірці деякої ознаки, гілки — результатам перевірки, а листя — підсумковому класу (рис. 2.8).

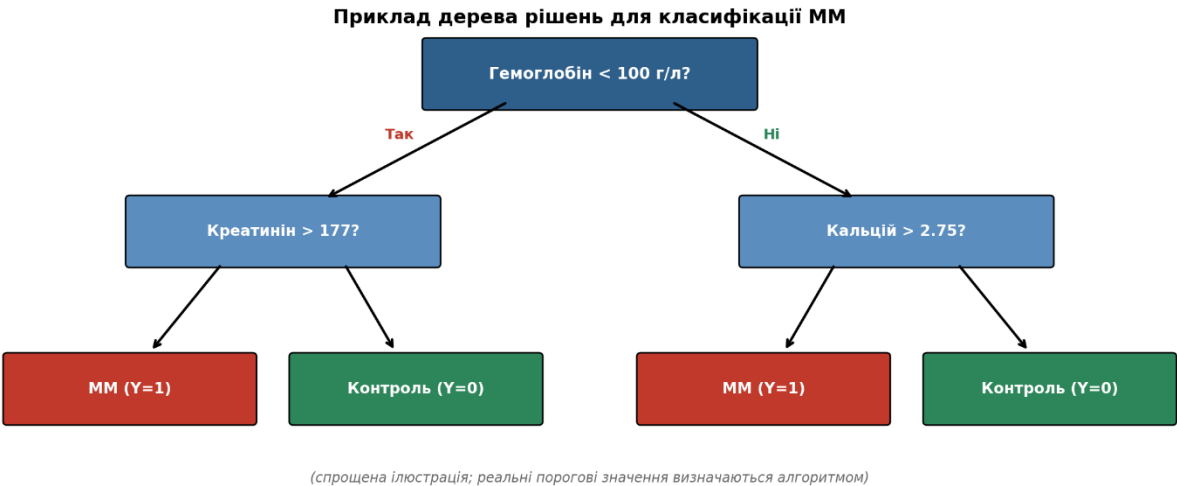


Рисунок 2.8 – Приклад дерева рішень для класифікації ММ (спрощено)

Для вибору ознаки в кожному вузлі використовують критерій, що максимізує однорідність дочірніх вузлів. Найпоширеніший — індекс Джині:

$$Gini(S) = 1 - \sum_{i=1}^c p_i^2 \quad (2.6)$$

Де p_i — частка прикладів класу i у вузлі. Алгоритм обирає те розбиття, яке найбільше зменшує індекс Джині (або, в інших реалізаціях, ентропію). Окреме дерево схильне до перенавчання, тому на практиці використовують їх ансамблі.

Випадковий ліс (Random Forest)

Випадковий ліс, запропонований Брейманом [37], реалізує стратегію беггінгу (bootstrap aggregating). Алгоритм будує велику кількість дерев, кожне з яких навчається на власній bootstrap-вибірці (випадковій підвибірці з поверненням), причому в кожному вузлі розглядається лише випадкова підмножина ознак. Фінальний прогноз отримується голосуванням більшості дерев. Подвійна випадковість — у виборі прикладів і ознак — забезпечує різноманітність дерев і знижує кореляцію між ними, що зменшує дисперсію ансамблю. Важлива перевага випадкового лісу — вбудована оцінка важливості ознак, яка лежить в основі методу Boruta, розглянутого в попередньому підрозділі.

Градiєнтний бустинг

Бустинг будує ансамбль послідовно: кожна наступна модель намагається виправити помилки попередніх. У градієнтному бустингу (gradient boosting), формалізованому Фрідманом [38], нова модель навчається на градієнтах функції втрат попереднього ансамблю. Формально ансамбль будується ітеративно:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \quad (2.7)$$

де $F_{m-1}(x)$ — поточний ансамбль, $h_m(x)$ — нове базове дерево, навчене апроксимувати негативний градієнт функції втрат, γ_m — коефіцієнт

(швидкість навчання). Послідовне додавання дерев поступово зменшує помилку, але створює ризик перенавчання, який стримується регуляризацією.

XGBoost

XGBoost (Extreme Gradient Boosting), запропонований Ченом і Гестріном [5], є оптимізованою реалізацією градієнтного бустингу, що стала стандартом для табличних даних. Його ключова відмінність — додавання регуляризації безпосередньо до цільової функції:

$$\mathcal{L} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k), \quad \Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2 \quad (2.8)$$

де $l(y_i, \hat{y}_i)$ — функція втрат на i -му прикладі, $\Omega(f_k)$ — штраф за складність k -го дерева, T — кількість листя, w — ваги листя, γ та λ — параметри регуляризації. Завдяки регуляризації XGBoost стійкіший до перенавчання за класичний градієнтний бустинг. Додаткові переваги — паралельне обчислення, вбудована обробка пропусків і можливість зважування класів через параметр `scale_pos_weight`, що корисно при незбалансованих медичних даних.

CatBoost

CatBoost (Categorical Boosting), розроблений компанією Яндекс [39], є ще однією реалізацією градієнтного бустингу з двома ключовими інноваціями. Перша — впорядкований бустинг (ordered boosting), що зменшує зміщення прогнозу через спеціальну схему перестановок при обчисленні градієнтів. Друга — ефективна вбудована обробка категоріальних ознак без потреби в ручному кодуванні. CatBoost зазвичай показує високу якість «з коробки», без тривалого налаштування гіперпараметрів, що робить його зручним для практичних задач.

Стекінг

Стекінг (stacking, stacked generalization), запропонований Волпертом [6], об'єднує моделі іншим способом, ніж бегінг чи бустинг. Замість простого

голосування він використовує мета-модель, яка навчається на прогнозах базових моделей (рис. 2.9).

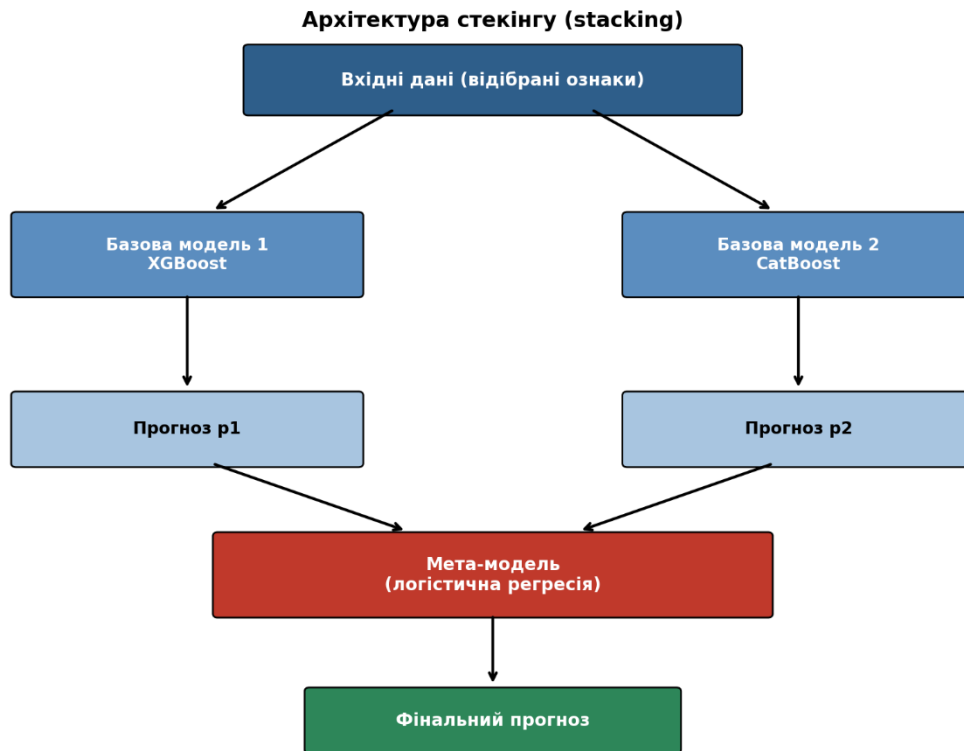


Рисунок 2.9 – Архітектура стекінгу

На першому рівні кілька різнотипних базових моделей (у цій роботі — XGBoost і CatBoost) навчаються на вхідних даних і видають свої прогнози. На другому рівні мета-модель (зазвичай проста — логістична регресія) навчається комбінувати ці прогнози в остаточне рішення. Перевага стекінгу в тому, що він поєднує сильні сторони різних базових моделей: якщо одна модель помиляється на певному типі прикладів, інша може це компенсувати, а мета-модель навчається, коли якій моделі довіряти.

Багатогілковий ансамбль (Multi-branch)

Багатогілкова архітектура є розвитком ідеї стекінгу, у якому кілька паралельних «гілок» обробки даних (що можуть включати як деревні моделі, так і нейромережні компоненти) об'єднуються на фінальному рівні. Кожна гілка може спеціалізуватися на різних аспектах даних або використовувати

різні представлення ознак, а підсумковий шар агрегує їх виходи. Така архітектура реалізована в одному з програмних модулів цієї роботи з використанням бібліотеки глибинного навчання.

Підсумок

Усі розглянуті методи належать до родини ансамблевих, але реалізують різні принципи: Random Forest — паралельний беггінг, XGBoost і CatBoost — послідовний бустинг, стекінг і Multi-branch — мета-навчання над базовими моделями. Така різноманітність обрана навмисно: вона дозволяє порівняти, який принцип ансамблювання найкраще підходить для задачі скринінгу ММ на конкретному наборі клініко-лабораторних даних. Теоретичні засади нейромережної моделі, що використовується як одна з гілок та як окремий класифікатор, розглянуто в наступному підрозділі.

2.4 Методи глибинного навчання

Концепція глибинного навчання

Глибинне навчання (deep learning) — це підгалузь машинного навчання, основою якої є штучні нейронні мережі з кількома прихованими шарами [4]. На відміну від класичних алгоритмів, які працюють з ознаками, побудованими людиною, нейронні мережі здатні самостійно формувати ієрархічне представлення даних: ранні шари виявляють прості закономірності, глибші — складніші абстракції, що ґрунтуються на висновках попередніх [7]. У табличних медичних задачах ансамблеві методи дерев рішень зазвичай конкурують або переважають нейронні мережі, проте останні залишаються цінним доповненням завдяки здатності моделювати складні нелінійні залежності й бути інтегрованими у багатогілкові архітектури.

Штучний нейрон

Базовою одиницею мережі є штучний нейрон — спрощена обчислювальна модель біологічного нейрона (рис. 2.10). Він обчислює зважену суму входів і застосовує до неї нелінійну функцію активації:

$$z = \sum_{i=1}^n w_i x_i + b, \quad y = f(z) \quad (2.9)$$

де x_i — значення входних ознак, w_i — вагові коефіцієнти, b — параметр зсуву, f — функція активації, y — вихід нейрона. Ваги w_i і зсув b є параметрами, що налаштовуються під час навчання.

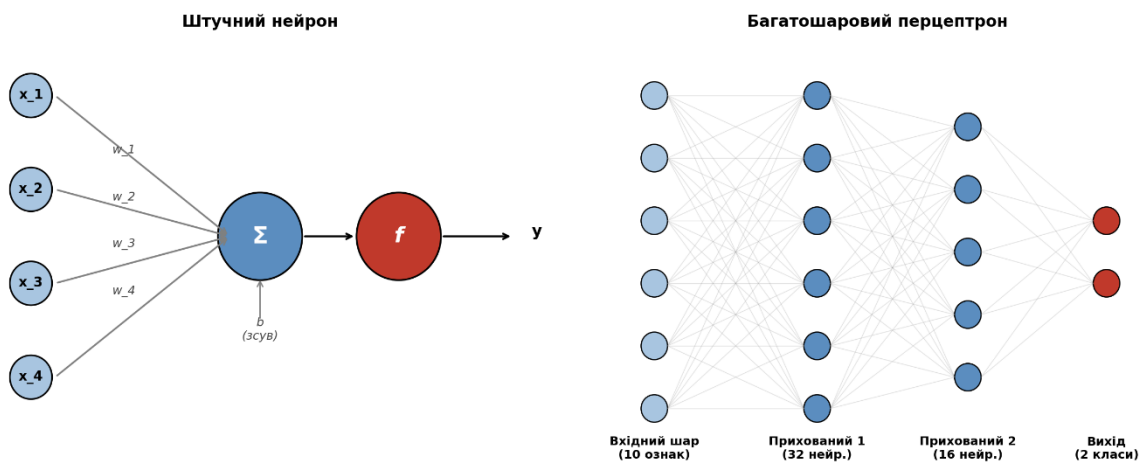


Рисунок 2.10 – Штучний нейрон (зліва) та архітектура багатошарового перцептрона (справа)

Багатошаровий перцептрон

Багатошаровий перцептрон (Multi-Layer Perceptron, MLP) — це найпростіша архітектура глибокої мережі, яка складається з кількох послідовних шарів нейронів: входного, одного чи кількох прихованих, та вихідного. Кожен нейрон шару пов'язаний з усіма нейронами наступного шару (повнозв'язні шари). Кількість нейронів у входному шарі дорівнює кількості ознак, у вихідному — кількості класів (для бінарної класифікації —

один нейрон з сигмоїдною активацією або два з softmax). Структуру прихованих шарів обирає дослідник: типово вона звужується від великої кількості нейронів до меншої.

Функції активації

Функція активації вносить нелінійність у мережу — без неї будь-яка глибока мережа була б еквівалентна одному лінійному перетворенню. На рисунку 2.11 показано чотири найпоширеніші функції.

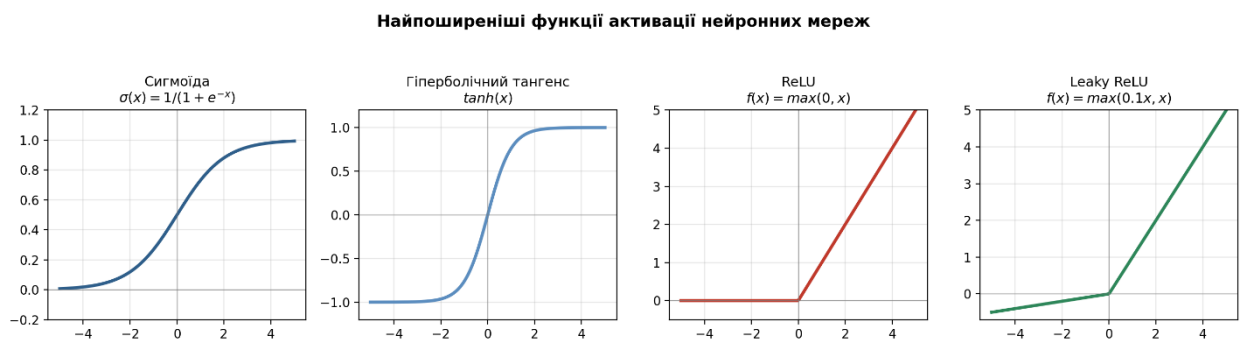


Рисунок 2.11 – Графіки найпоширеніших функцій активації

Сигмоїда (логістична функція):

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.10)$$

Перетворює будь-яке число у діапазон (0, 1), що зручно інтерпретувати як ймовірність. Історично була найпоширенішою активацією, але має суттєвий недолік — затухання градієнтів при великих за модулем значеннях, що ускладнює навчання глибоких мереж.

Гіперболічний тангенс $\tanh(x)$ — подібний до сигмоїди, але набуває значень у діапазоні (-1, 1) і центрований відносно нуля, що сприяє швидшій збіжності.

ReLU (Rectified Linear Unit), запропонована для глибокого навчання Глоро, Бордесом і Бенжіо [40]:

$$ReLU(x) = \max(0, x) \quad (2.11)$$

Стала стандартною активацією у сучасних архітектурах завдяки простоті обчислення та відсутності проблеми затування градієнта при додатних значеннях. Її варіант *Leaky ReLU* має невеликий ненульовий нахил для від'ємних значень, що уникає «мертвих» нейронів — ситуації, коли нейрон постійно видає нуль і перестає навчатися.

Для вихідного шару у бінарній класифікації використовують сигмоїду; для багатокласової — функцію softmax, яка перетворює вектор активацій у розподіл ймовірностей по класах.

Функція втрат

Для задачі бінарної класифікації стандартною є функція бінарної крос-ентропії:

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n [y_i \log p_i + (1 - y_i) \log(1 - p_i)] \quad (2.12)$$

де $y_i \in \{0, 1\}$ — справжня мітка, $p_i \in (0, 1)$ — передбачена ймовірність належності до позитивного класу. Функція досягає мінімуму, коли передбачення точно збігаються з мітками.

Процес навчання

Навчання мережі полягає у пошуку таких вагових коефіцієнтів, які мінімізують функцію втрат на навчальних даних. Алгоритм складається з двох фаз: прямого поширення (обчислення прогнозу від входу до виходу) і зворотного поширення помилки (backpropagation), яке обчислює градієнти функції втрат по всіх параметрах мережі за допомогою правила ланцюга [41]. Після обчислення градієнтів параметри оновлюються методом градієнтного спуску.

У найпростішій формі — стохастичному градієнтному спуску (SGD) — оновлення відбувається за правилом:

$$w \leftarrow w - \eta \nabla_w \mathcal{L} \quad (2.13)$$

де η — швидкість навчання (learning rate). На практиці майже завжди використовують адаптивні оптимізатори, найпопулярнішим з яких є Adam,

запропонований Кінгмою і Ба [42]. Adam адаптивно змінює швидкість навчання для кожного параметра на основі першого і другого моментів градієнта, що значно прискорює збіжність на більшості задач.

Регуляризація

Глибокі мережі схильні до перенавчання — запам'ятовування навчальних прикладів замість виявлення узагальнюючих закономірностей. Для боротьби з цим застосовують кілька технік:

- *Dropout*, запропонований Срівастава та співавторами [43] — на кожному кроці навчання випадково «вимикається» певна частка нейронів (типово 0,2–0,5), що змушує мережу не покладатися на окремі нейрони;
- *L2-регуляризація* (weight decay) — додавання штрафу за великі вагові коефіцієнти у функцію втрат;
- *Раннє припинення* (early stopping) — припинення навчання, коли якість на валідаційній вибірці перестає покращуватися.

Багатогілкова архітектура

Як уже згадувалося в підрозділі 2.3, багатогілкова (multi-branch) архітектура — це нейромережний варіант стекінгу. Вхідні дані одночасно подаються на кілька паралельних гілок, кожна з яких є окремою послідовністю шарів зі своєю структурою (рис. 2.12). Виходи гілок об'єднуються (конкатенуються) і подаються на фінальний шар, який видає підсумкове рішення.

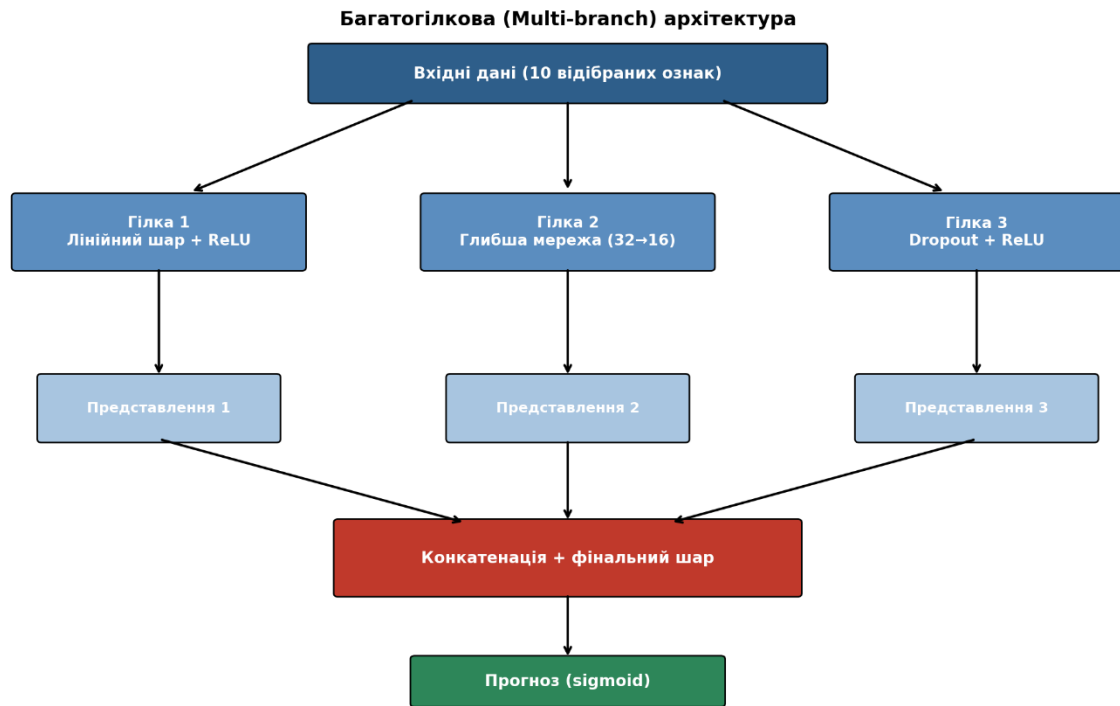


Рисунок 2.12 – Багатогілкова (Multi-branch) архітектура нейронної мережі

Перевага такого підходу — спеціалізація гілок: одна може використовувати ширші повнозв'язні шари, інша — глибші з регуляризацією через dropout, третя — компактну архітектуру з акцентом на швидкість. Об'єднання різних представлень даних дає кінцевій моделі гнучкість, недосяжну для одного лінійного стеку шарів. У даній роботі багатогілкова мережа реалізована з використанням бібліотеки torch.

Підсумок

Глибинне навчання у формі багат шарового перцептрона додає до набору моделей роботи нелінійну альтернативу деревним ансамблям. Воно дозволяє виявляти складні взаємодії між клініко-лабораторними показниками, які не завжди вловлюються деревами рішень. Поєднання класичних ансамблів (Random Forest, XGBoost, CatBoost), стекінгу і нейромережних архітектур (MLP, Multi-branch) у межах однієї роботи створює основу для змістовного порівняльного аналізу, який буде проведено в Розділі 4.

2.5 Метрики оцінювання якості

Чому метрики критичні для медичних задач

Оцінка якості класифікаційної моделі не зводиться до однієї цифри. Особливо це справедливо для медицини, де помилки двох типів мають різну ціну: пропуск хворого (модель не виявила ММ у пацієнта, який її має) клінічно небезпечніший за хибну тривогу (підозра у здорового). Тому в роботі застосовано повний набір метрик, який дозволяє оцінити модель з різних боків і виявити її сильні та слабкі сторони.

Матриця помилок

Основою всіх класифікаційних метрик є матриця помилок (confusion matrix), у якій порівнюються прогнози моделі зі справжніми мітками (рис. 2.13).

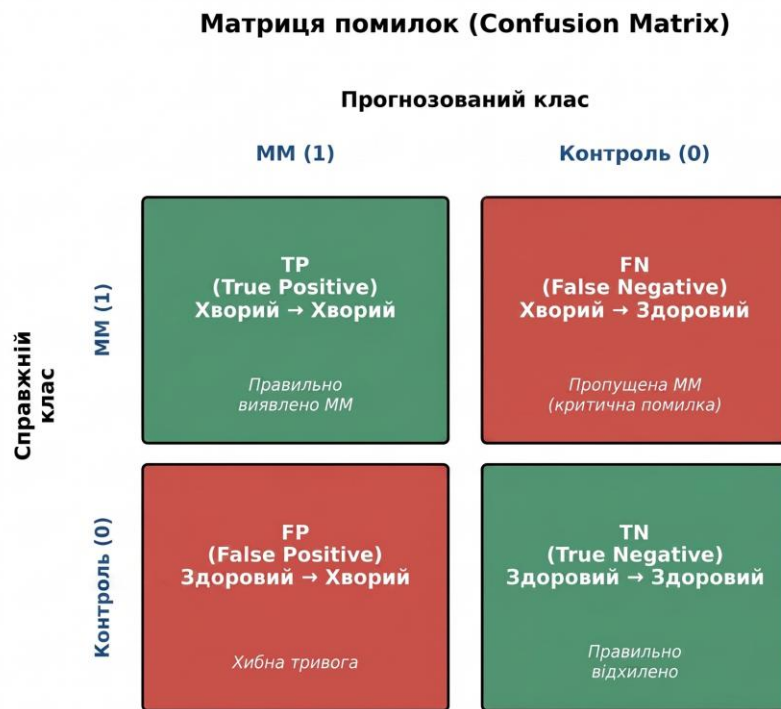


Рисунок 2.13 – Матриця помилок для задачі бінарної класифікації ММ

Чотири клітинки матриці відповідають чотирьом можливим результатам:

- *TP (True Positive)* — модель правильно класифікувала пацієнта з ММ як хворого;
- *TN (True Negative)* — модель правильно віднесла здорового до контрольної групи;
- *FP (False Positive)* — модель помилково класифікувала здорового як хворого (хибна тривога);
- *FN (False Negative)* — модель пропустила реального хворого, віднісши його до здорових (найгірший тип помилки в задачі скринінгу).

Базові метрики

На основі чотирьох величин матриці помилок обчислюється низка метрик [47].

Accuracy (точність) — частка правильно класифікованих об'єктів від загальної кількості:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.14)$$

Це найпростіша і найзагальніша метрика, але при незбалансованих класах вона оманлива: модель, що завжди прогнозує більший клас, може мати високу ассигасу, не виявляючи жодного хворого.

Precision (точність прогнозу позитивного класу) — частка дійсно хворих серед тих, кого модель віднесла до хворих:

$$Precision = \frac{TP}{TP + FP} \quad (2.15)$$

Висока precision означає, що модель рідко б'є хибну тривогу.

Recall (повнота, чутливість, sensitivity) — частка виявлених хворих серед усіх дійсно хворих:

$$Recall = \frac{TP}{TP + FN} \quad (2.16)$$

Для скринінгової задачі recall — ключова метрика, оскільки відображає здатність системи не пропускати хворих.

Specificity (специфічність) — частка правильно віднесених до контролю серед усіх здорових:

$$Specificity = \frac{TN}{TN + FP} \quad (2.17)$$

Висока специфічність означає, що модель не «лякає» здорових пацієнтів.

F1-score — гармонійне середнє precision і recall, що дає єдину збалансовану оцінку:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.18)$$

F1 особливо корисний при незбалансованих класах, бо штрафует одночасно і за пропуски, і за хибні тривоги.

ROC-крива та AUC

Усі попередні метрики обчислюються за фіксованого порога ймовірності — зазвичай 0,5: усе, що вище — позитивний клас, нижче — негативний. Однак цей поріг можна змінювати, і кожен поріг дає свою комбінацію Recall і False Positive Rate. Якщо побудувати графік залежності True Positive Rate (Recall) від False Positive Rate ($1 - \text{Specificity}$) для всіх можливих порогів, отримаємо ROC-криву (Receiver Operating Characteristic), схематичне зображення якої наведено на рисунку 2.14 [44].

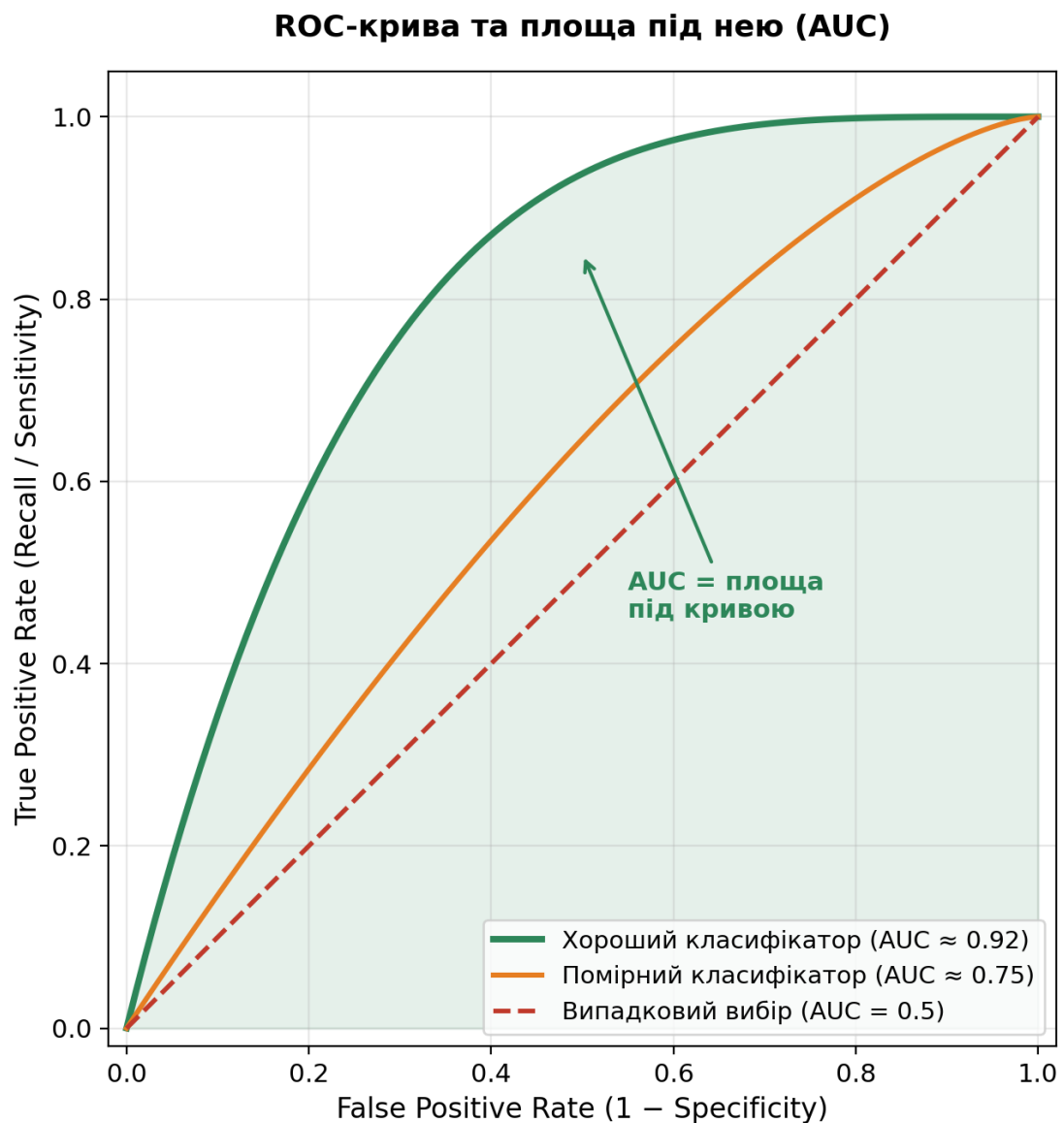


Рисунок 2.14 – ROC-крива та інтерпретація площі під нею (AUC)

Чим ближче ROC-крива підіймається до лівого верхнього кута, тим краща модель. Інтегральною мірою якості є площа під кривою — AUC (Area Under Curve):

$$AUC = \int_0^1 TPR(FPR) d(FPR) \quad (2.19)$$

AUC = 1 відповідає ідеальному класифікатору, AUC = 0,5 — випадковому. У медичних дослідженнях AUC є золотим стандартом метрик: вона не залежить від обраного порогу і коректно працює при незбалансованих класах. У сучасних роботах з машинного навчання у гематології AUC > 0,9 вважається ознакою високої якості моделі [27, 29].

Оцінка статистичної значущості

Для коректного порівняння моделей одних точкових оцінок метрик недостатньо: різниця між двома моделями може виявитися статистично незначущою — простим випадковим коливанням. Тому в роботі застосовуються методи статистичного оцінювання, на необхідність яких прямо вказала рецензія попередньої версії дослідження.

Тест DeLong [45] — непараметричний метод порівняння AUC двох пов'язаних класифікаторів, тобто тих, що оцінюються на одних і тих самих даних. Він базується на теорії U-статистик і дає р-значення для гіпотези «AUC двох моделей рівні». Якщо $p < 0,05$, різницю вважають статистично значущою.

Парний критерій Вілкоксона (Wilcoxon signed-rank test) — використовується для порівняння метрик моделей, обчислених на фолдах крос-валідації. Він не вимагає припущень про нормальність розподілу, що зручно для невеликих вибірок.

Бутстреп-довірчі інтервали. Для кожної метрики можна побудувати 95 %-вий довірчий інтервал методом бутстрапу [46]: тестова вибірка багаторазово (зазвичай 1000 разів) семплюється з поверненням, метрика обчислюється на кожній підвибірці, а 2,5 % та 97,5 % перцентилі утвореного розподілу дають межі інтервалу. Інтервал показує діапазон, у якому з 95 %-вою ймовірністю лежить справжнє значення метрики.

Інтегрований підхід до оцінювання

У межах роботи кожна модель оцінюється за повним набором метрик: Accuracy, Precision, Recall, Specificity, F1-score та AUC. Усі метрики представлені точковими значеннями та 95 %-вими довірчими інтервалами. Для парного порівняння моделей використовується тест DeLong (для AUC) та критерій Вілкоксона (для метрик з крос-валідації). Такий підхід забезпечує не лише обчислення «найбільшого числа», а й статистично обґрунтований висновок про те, чи є різниця між моделями значущою.

Висновки до розділу 2

У другому розділі викладено теоретичні основи методів, використаних у роботі. Розглянуто весь пайплайн машинного навчання — від попередньої обробки даних до фінального оцінювання якості моделей. Описано три методологічно різнотипні методи відбору ознак (Information Gain як фільтровий, RFE як обгортковий та Boruta на основі ансамблю Random Forest) з відповідними математичними формулюваннями. Детально розглянуто п'ять класифікаційних моделей, що належать до родини ансамблевих: випадковий ліс (беггінг), XGBoost і CatBoost (градієнтний бустинг), стекінг (мета-навчання) та багатогілкову нейромережну архітектуру; додатково розкрито теорію багатошарового перцептрона як окремої моделі. Окремий підрозділ присвячено метрикам якості: матриці помилок, базовим метрикам (Accuracy, Precision, Recall, Specificity, F1), ROC-AUC, а також статистичним методам оцінки значущості відмінностей між моделями (тест DeLong, критерій Вілкоксона, бутстрап-довірчі інтервали). Такий комплексний апарат створює основу для коректної реалізації програмної системи

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ ДІАГНОСТИКИ ММ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ

3.1 Загальна архітектура системи

Загальна структура

У цьому розділі викладено методику побудови системи інтелектуальної діагностики ММ, представлено результати її експериментальної валідації та клінічну інтерпретацію відібраних ознак. Програмна реалізація відповідних модулів описана у Розділі 4. Розроблена система побудована як модульний дослідницький конвеєр, що послідовно виконує завантаження даних, відбір ознак, навчання моделей, оцінювання та статистичний аналіз. Модульна побудова обрана свідомо: вона спрощує тестування окремих етапів, забезпечує відтворюваність і дозволяє повторно використовувати компоненти. Загальну архітектуру системи показано на рисунку 3.1.

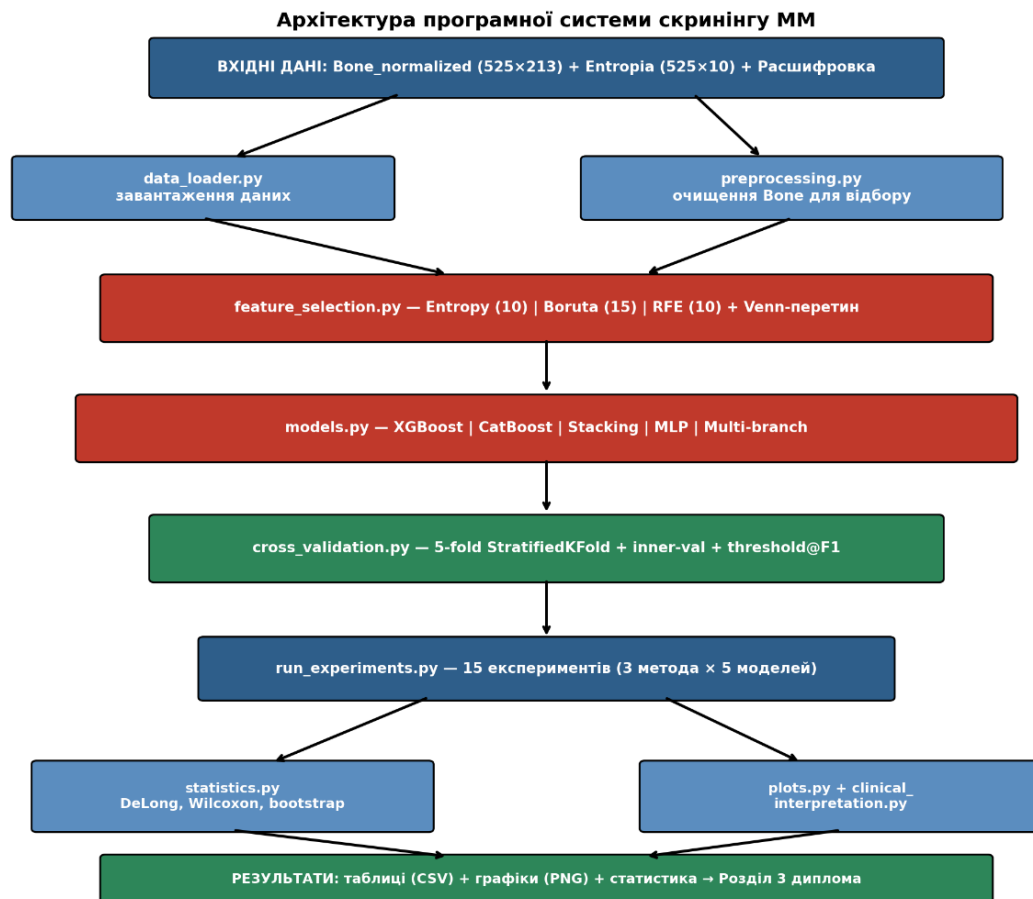


Рисунок 3.1 – Архітектура програмної системи скринінгу множинної мієломи

Конвеєр системи об’єднує вісім функціональних блоків, що відповідають типовим етапам сучасного пайплайну машинного навчання:

завантаження даних — зчитування вихідних наборів та таблиці розшифровки ознак;

попередня обробка — підготовка повного набору з 213 ознак для алгоритмів відбору;

відбір ознак — застосування трьох методів (Information Gain, Boruta, RFE) та обчислення їх перетину;

класифікаційні моделі — побудова п’яти моделей із гіперпараметрами, узгодженими з референсними R-скриптами;

крос-валідація — стратифікована 5-кратна схема з внутрішнім розщепленням та підбором порога;

оркестрація експериментів — послідовне виконання повного циклу з 15 експериментів;

статистичний аналіз — тести значущості (DeLong, Вілкоксона) та бутстрап-інтервали;

візуалізація та інтерпретація — побудова графіків і формування таблиці клінічної інтерпретації відібраних ознак.

Технологічний стек

Систему побудовано на екосистемі наукового Python; деталі технологічного стеку, конкретні бібліотеки та структура коду наведено в Розділі 4.

Забезпечення відтворюваності

Критична вимога до наукового програмного забезпечення — відтворюваність результатів. У системі вона забезпечена фіксацією початкового значення генератора випадкових чисел (`random_seed = 123`) у всіх стохастичних компонентах: розбитті на фолди, ініціалізації моделей, бутстрап-семплюванні. Це значення успадковане з референсних R-скриптів керівника, що дозволяє зіставляти результати між двома реалізаціями. Усі параметри експерименту винесені в окремий конфігураційний модуль `config.py`, що централізує керування і виключає «магічні числа» у коді. Завдяки цьому будь-який запуск системи на тих самих даних дає ідентичні результати.

3.2 Опис вхідних даних

Характеристика набору даних

Дослідження виконано на реальному клініко-лабораторному наборі даних, зібраному в гематологічному центрі міста Люблін (Польща). Набір містить записи 525 пацієнтів, кожен з яких описується клініко-лабораторними показниками та бінарною цільовою змінною. Цільова змінна Y кодує наявність множинної мієломи: $Y = 1$ — пацієнт з підтвердженою ММ, $Y = 0$ — контрольна група без ММ.

Розподіл класів наведено в таблиці 3.1.

Таблиця 3.1 – Розподіл пацієнтів за класами

<i>Клас</i>	<i>Кількість</i>	<i>Частка</i>
$Y = 1$ (множинна мієлома)	298	56,8 %
$Y = 0$ (контроль)	227	43,2 %
Разом	525	100 %

Як видно з таблиці, набір є помірно незбалансованим — позитивний клас переважає у співвідношенні приблизно 1,3:1. Такий дисбаланс не є критичним і не вимагає синтетичної генерації прикладів (SMOTE); натомість він компенсується зважуванням класів безпосередньо в моделях, через параметр `scale_pos_weight` (XGBoost) та `class_weights` (CatBoost), що відповідає підходу референсних R-скриптів.

Два рівні представлення даних

У роботі використано два пов'язані набори даних:

`Bone_normalized_data.xlsx` — повний набір, що містить усі 213 клініко-лабораторних показників (з нумерацією $x_1 \dots x_{283}$, де пропуски відповідають ознакам, видаленим на етапі попереднього очищення). Цей набір слугує вихідним простором ознак для алгоритмів Boruta та RFE.

Dataset-Cleaned-Entropy.xlsx — скорочений набір з 10 ознак (x5, x15, x17, x18, x19, x20, x25, x70, x121, x158), відібраних методом ентропійного інформаційного приросту. Цей відбір було виконано науковим керівником, і саме цей набір є основним робочим набором для класифікації згідно з його вказівкою.

Усі числові показники в обох наборах попередньо нормалізовано методом мінімаксної нормалізації у діапазон $[0, 1]$ (формула 2.1). Пропущених значень у наборах немає, що спрощує етап попередньої обробки.

Інтерпретація ознак

Для змістовного аналізу результатів використано файл розшифровки Расшифровка.xlsx, який зіставляє кодові позначення ознак (x5, x15 тощо) з їх медичними назвами польською та російською мовами. Оскільки дані зібрані в польській клініці, за основу взято польські назви. Найважливіші відібрані ознаки та їх медичну інтерпретацію детально розглянуто в підрозділі 3.6.

3.3 Методика відбору ознак

У роботі застосовано три методологічно різні методи відбору ознак, теоретичні засади яких описано в підрозділі 2.2. Кожен метод використовувався незалежно для виявлення інформативних клініко-лабораторних показників, після чого набори відібраних ознак порівнювалися між собою — це дозволяє виявити стабільну клінічну сигнатуру, відібрану незалежно різними алгоритмами.

Ентропійний метод (Information Gain)

Ентропійний відбір було виконано науковим керівником на повному наборі з 213 ознак, а його результат надано як готовий файл Dataset-Cleaned-Entropy.xlsx. Метод відібрав 10 ознак з найвищим інформаційним приростом:

x5, x15, x17, x18, x19, x20, x25, x70, x121, x158. У системі цей набір використовується безпосередньо як один з трьох порівнюваних варіантів.

Метод Boruta

Метод Boruta застосовано як обгортку над класифікатором випадкового лісу з параметрами `n_estimators=100`, `max_depth=5` та зважуванням класів за принципом `balanced` для врахування дисбалансу; кількість ітерацій алгоритму обмежено значенням 100. Метод запускався на повному наборі з 213 ознак. За результатами статистичного тестування проти тіньових ознак Boruta підтвердив 15 ознак як значущі: x3, x5, x9, x14, x15, x16, x17, x18, x19, x20, x25, x94, x121, x153, x158. Характерно, що Boruta відібрав більше ознак, ніж ентропійний метод, оскільки за своєю природою він прагне знайти всі релевантні ознаки, а не фіксовану їх кількість.

Метод RFE

Метод RFE застосовано на основі логістичної регресії (`solver liblinear`, до 1000 ітерацій, з вагами класів `balanced`). Для прямого порівняння з ентропійним методом цільову кількість ознак встановлено на рівні 10. Запуск на повному наборі дав такі 10 ознак: x3, x15, x17, x27, x70, x72, x75, x121, x155, x158.

Перетин методів

Зіставлення трьох наборів відібраних ознак є одним з ключових результатів роботи. Перетин візуалізовано діаграмою Венна на рисунку 3.2.

Перетин відібраних ознак між методами

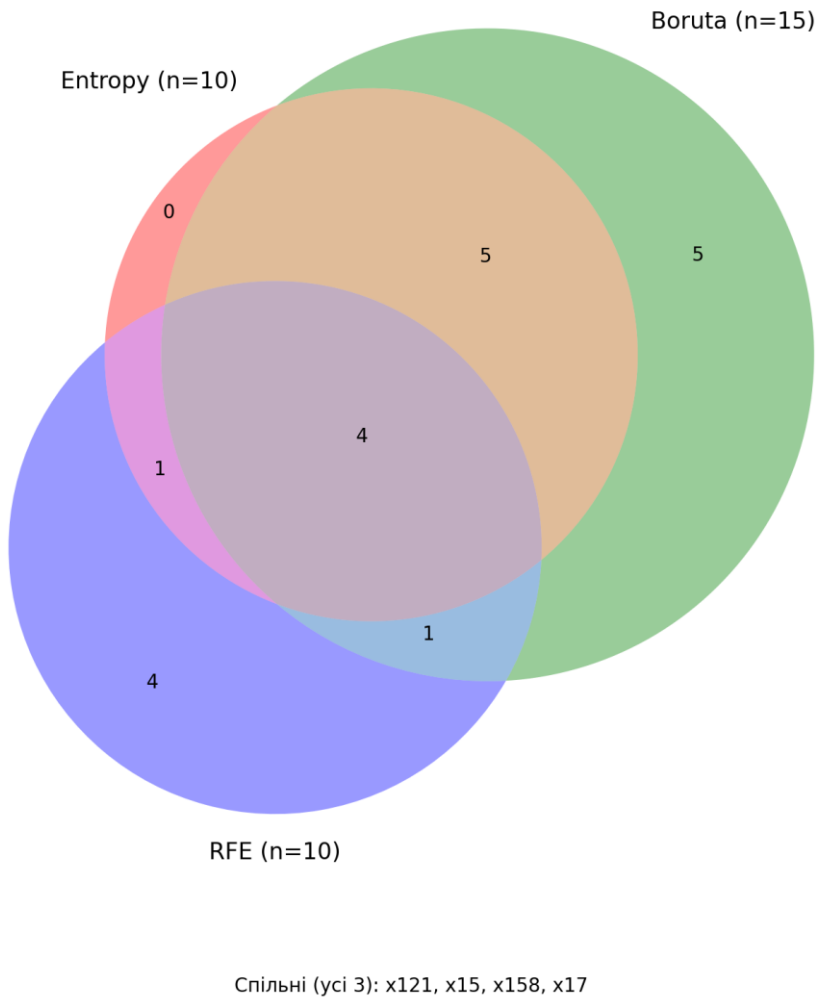


Рисунок 3.2 – Перетин наборів ознак, відібраних трьома методами

Аналіз перетину дає такі результати (таблиця 3.2):

Таблиця 3.2 – Перетин ознак між методами відбору

Підмножина	Кількість	Ознаки
Усіма трьома методами	4	x15, x17, x121, x158
Entropy $\cap$ Boruta	9	x5, x15, x17, x18, x19, x20, x25, x121, x158
Entropy $\cap$ RFE	5	x15, x17, x70, x121, x158
Boruta $\cap$ RFE	5	x3, x15, x17, x121, x158
Тільки Boruta	5	x9, x14, x16, x94, x153
Тільки RFE	4	x27, x72, x75, x155

Чотири ознаки — x15, x17, x121, x158 — були незалежно відібрані всіма трьома методами. Це найсильніший аргумент на користь їх діагностичної значущості: різні за принципом дії алгоритми (фільтровий, обгортковий і на основі ансамблю) збіглися у виборі саме цих показників. Медичну інтерпретацію цих ознак розглянуто в підрозділі 3.6.

Слід окремо обґрунтувати методологічний аспект. Відбір ознак методами Boruta та RFE виконано на всій вибірці, а не всередині кожного фолду крос-валідації. Такий вибір зроблено свідомо: основна мета відбору — сформувати єдину стабільну клінічну сигнатуру, тобто сталий набір біомаркерів, придатний для медичної інтерпретації, а не лише отримати оцінку узагальнювальної здатності. Для абсолютно неупередженої оцінки генералізації процедуру відбору слід було б перенести всередину циклу крос-валідації; це визначено як напрям подальшого вдосконалення методології.

3.4 Методика навчання класифікаційних моделей та схема валідації

Конфігурація моделей

Для класифікації застосовано п'ять моделей різних архітектурних класів: ансамблеві XGBoost і CatBoost, мета-модель стекінгу над ними, а також неймережні моделі — багат шаровий перцептрон (MLP) та багатогілкову архітектуру (Multi-branch). Гіперпараметри XGBoost, CatBoost і стекінгу обрано згідно з референсними R-скриптами наукового керівника, що забезпечує методологічну спадкоємність. Основні параметри наведено в таблиці 3.3.

Таблиця 3.3 – Гіперпараметри моделей класифікації

<i>Модель</i>	<i>Ключові параметри</i>
XGBoost	max_depth=4, eta=0,03, min_child_weight=3, subsample=0,85, colsample_bytree=0,75, lambda=1,0, scale_pos_weight= N_0/N_1 , nrounds=1000, early_stopping=40
CatBoost	iterations=1500, learning_rate=0,03, depth=5, l2_leaf_reg=5, od_wait=50, class_weights збалансовані
Stacking	базові: XGBoost + CatBoost; мета-модель: логістична регресія на out-of-fold прогнозах
MLP	2 приховані шари (32 $\rightarrow$ 16), ReLU, dropout=0,2, BCEWithLogitsLoss з pos_weight, оптимізатор AdamW, Reduce Learning Rate on Plateau (ReduceLROnPlateau)
Multi-branch	3 гілки (BranchEncoder: Linear+ Batch Normalization (BN)+ReLU+Dropout), gated fusion з softmax, голова Linear(64) $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Dropout $\rightarrow$ Linear(32) $\rightarrow$ ReLU $\rightarrow$ Dropout $\rightarrow$ Linear(1)

Нейромережні моделі (MLP та Multi-branch) побудовано як повнозв'язні архітектури з функціями активації ReLU. Для балансування класів у функції втрат застосовано ваговий коефіцієнт, що є нейромережним аналогом параметра `scale_pos_weight` ансамблевих моделей. У Multi-branch ознаки розподіляються між трьома гілками за біологічним принципом: перша гілка обробляє еритроцитарні індекси (x15, x17, x18, x19, x20), друга — інші клітинні показники (x5, x25), третя — біохімічні маркери (x70, x121, x158). Кожна гілка формує власне представлення, після чого механізм gated fusion зважає внески гілок через softmax-ваги.

Схема валідації

Для надійної оцінки якості застосовано стратифіковану 5-кратну крос-валідацію (StratifiedKFold з `random_state=123`), яка зберігає пропорцію класів у кожному фолді. У середині кожного навчального фолду виконується додаткове розщеплення: 15 % даних виділяється у внутрішню валідаційну підмножину для ранньої зупинки навчання (early stopping) та підбору порога класифікації.

Поріг класифікації не фіксується на стандартному значенні 0,5, а підбирається індивідуально для кожної моделі за максимумом F1-міри на валідаційній підмножині (крок перебору 0,01). Цей підхід успадкований з R-скриптів керівника і важливий для медичних задач, де баланс між чутливістю і специфічністю слід налаштовувати свідомо. Підсумкові метрики усереднюються за п'ятьма фолдами; для них обчислюються стандартне відхилення та 95 %-ві бутстрап-довірчі інтервали (1000 повторів).

Експериментальна матриця

Поєднання трьох методів відбору з п'ятьма моделями дає 15 експериментів, які виконуються послідовно модулем `run_experiments.py`. Кожен експеримент проходить повний цикл крос-валідації з обчисленням усіх метрик і збереженням прогнозів для подальшого статистичного аналізу. Структуру експериментів подано в таблиці 3.4.

Таблиця 3.4 – Матриця експериментів

<i>Метод відбору Модель</i>	<i>XGBoost</i>	<i>CatBoost</i>	<i>Stacking</i>	<i>Multi-branch</i>	<i>MLP</i>
Entropy (10 ознак)	+	+	+	+	+
Boruta (15 ознак)	+	+	+	+	+
RFE (10 ознак)	+	+	+	+	+

Результати всіх 15 експериментів та їх статистичний аналіз розглянуто в наступному підрозділі.

3.5 Результати та порівняльний аналіз

Зведені результати експериментів

Усі 15 експериментів виконано за описаною схемою крос-валідації і впорядковано за спаданням AUC — основної метрики порівняння. У роботі застосовано два різновиди AUC: Out-Of-Fold (OOF) AUC (обчислений на об'єднаному векторі прогнозів усіх фолдів, наводиться як основна метрика для оцінки узагальнювальної здатності) та fold-mean AUC (середнє арифметичне AUC окремих фолдів, наводиться для порівняння стійкості моделей між фолдами). Якщо нижче в тексті явно не вказано протилежне, мається на увазі fold-mean AUC; OOF AUC та його бутстрап-довірчі інтервали наведено окремо в розділі статистичного аналізу. Підсумкові метрики, усереднені за п'ятьма фолдами, наведено в таблиці 3.5.

Таблиця 3.5 – Результати 15 експериментів (середнє за 5 фолдами)

<i>Відбір + Модель</i>	<i>AUC</i>	<i>Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>Specificity</i>	<i>F1</i>
Boruta + Stacking	0,855	0,779	0,775	0,862	0,669	0,815
Boruta + CatBoost	0,851	0,775	0,772	0,869	0,653	0,814
Boruta + XGBoost	0,849	0,775	0,784	0,835	0,696	0,808
Entropy + CatBoost	0,843	0,752	0,784	0,795	0,696	0,785
Entropy + Stacking	0,842	0,756	0,765	0,842	0,643	0,798
Entropy + XGBoost	0,827	0,718	0,731	0,812	0,595	0,763
RFE + Stacking	0,819	0,737	0,741	0,842	0,599	0,785
RFE + CatBoost	0,812	0,745	0,755	0,829	0,634	0,787
RFE + XGBoost	0,808	0,731	0,753	0,812	0,626	0,774

Кінець таблиці 3.5

<i>Відбір + Модель</i>	<i>AUC</i>	<i>Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>Specificity</i>	<i>F1</i>
Boruta + Multi-branch	0,803	0,726	0,784	0,738	0,709	0,747
Entropy + Multi-branch	0,801	0,716	0,746	0,788	0,620	0,757
RFE + MLP	0,745	0,695	0,675	0,906	0,420	0,771
RFE + Multi-branch	0,743	0,684	0,717	0,782	0,556	0,737
Boruta + MLP	0,708	0,638	0,626	0,902	0,290	0,737
Entropy + MLP	0,702	0,617	0,645	0,783	0,402	0,688

Аналіз за методами відбору

Результати демонструють чітку перевагу ознак, відібраних методом Boruta: три найкращі комбінації (Stacking, CatBoost, XGBoost на Boruta-ознаках) показали fold-mean AUC у діапазоні 0,849–0,855 (середнє за п'ятьма фолдами), що відповідає OOF AUC 0,841–0,846 на об'єднаному векторі прогнозів. Це можна пояснити тим, що Boruta відібрав більше ознак (15 проти 10), зберігши при цьому всі ключові — він не обмежувався фіксованою кількістю, а залишав усі статистично значущі показники. Ентропійний відбір зайняв проміжну позицію, RFE — дещо нижчу, що логічно: RFE на основі лінійної логістичної регресії гірше вловлює нелінійні взаємодії, важливі для ансамблевих моделей.

Аналіз за моделями

Серед моделей найкращі результати стабільно показали ансамблі градієнтного бустингу та стекінг. Stacking, що поєднує XGBoost і CatBoost, очікувано виявився найсильнішим на двох із трьох методів відбору.

Багатошаровий перцептрон (MLP) показав найгірші результати (AUC 0,70–0,75). Це закономірно: на вибірках малого обсягу (менш ніж 1000 записів, у нашому випадку лише 525) нейронні мережі не мають достатньо даних для ефективного навчання й стабільно поступаються класичним ансамблям дерев, які значно результативніші на табличних даних такого розміру. Цікавим є патерн MLP: висока чутливість (Recall до 0,906) за рахунок низької специфічності (до 0,290) — модель схильна відносити більшість пацієнтів до позитивного класу, що для скринінгу неприйнятно через велику кількість хибних тривог.

ROC-аналіз

ROC-криві всіх 15 експериментів зображено на рисунку 3.3. Криві найкращих моделей помітно вибираються у лівий верхній кут, що відповідає високій якості розрізнення класів.

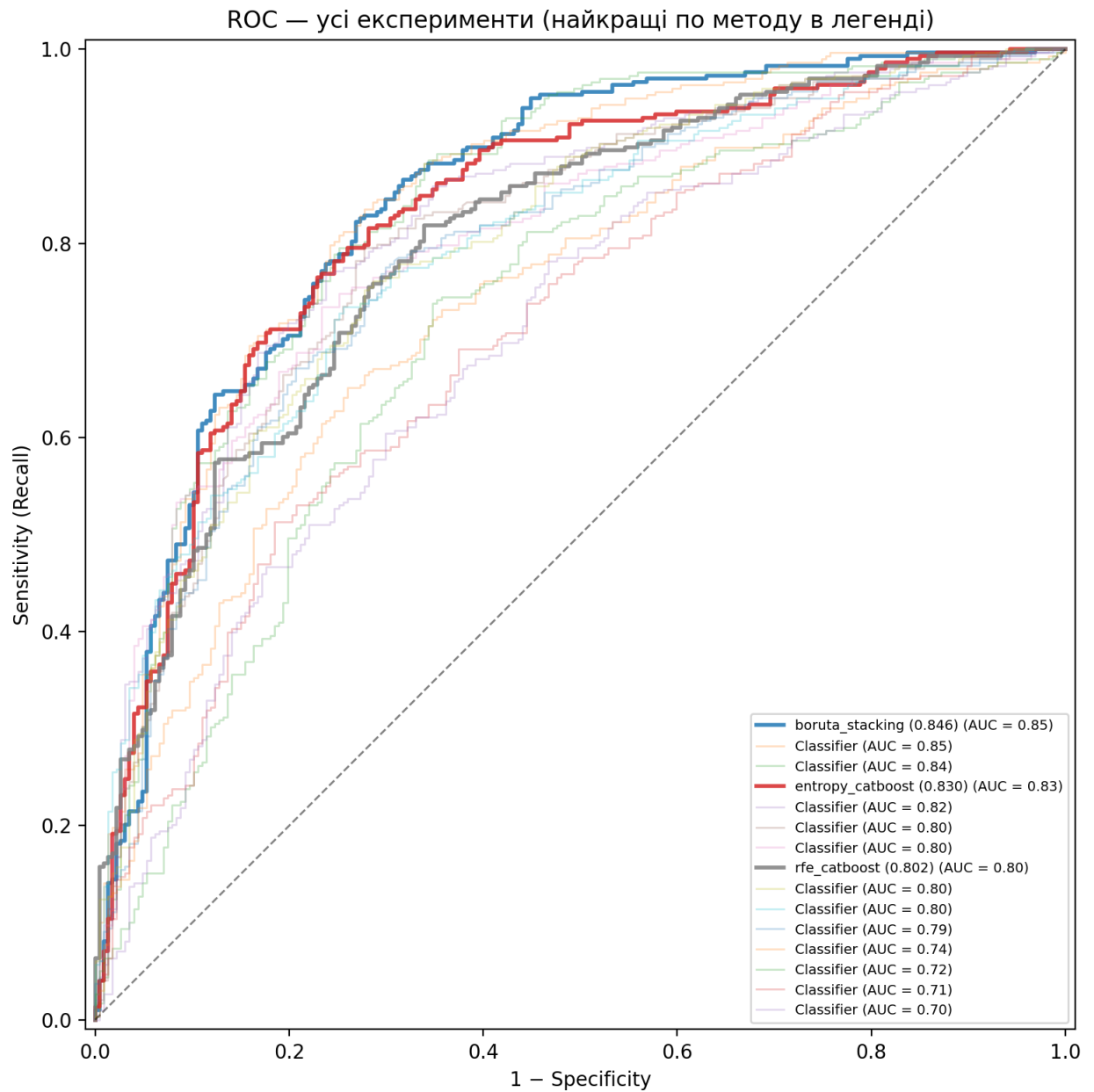


Рисунок 3.3 – ROC-криві всіх експериментів (найкращі моделі виділено)

Порівняння основних метрик усіх комбінацій представлено на стовпчиковій діаграмі (рис. 3.4), а розподіл метрик по фолдах — на діаграмі розмаху (рис. 3.5).

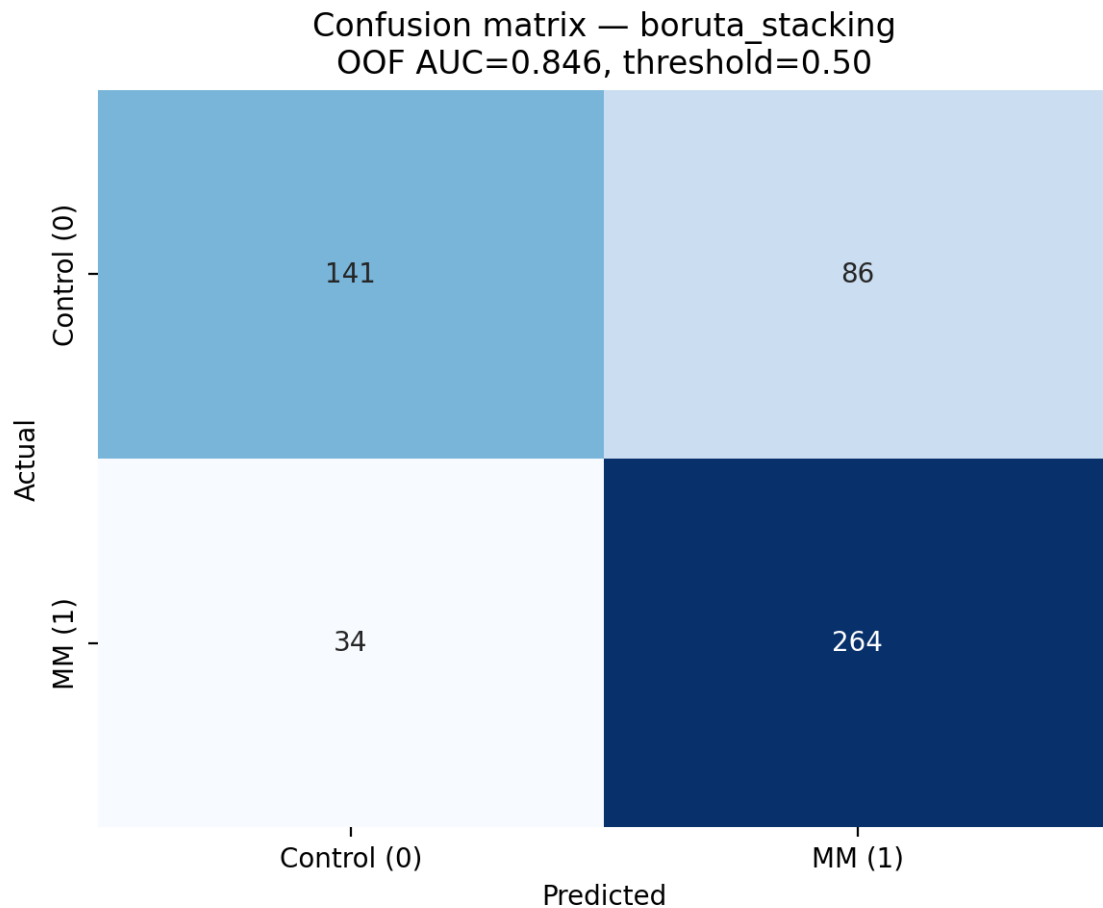


Рисунок 3.6 – Матриця помилок моделі Boruta + Stacking

Статистична значущість відмінностей

Точкова перевага однієї моделі над іншою ще не означає статистично значущої різниці. Для перевірки застосовано тест DeLong (порівняння AUC) та критерій Вілкоксона (порівняння по фолдах). Матриці р-значень візуалізовано на рисунках 3.7 і 3.8.

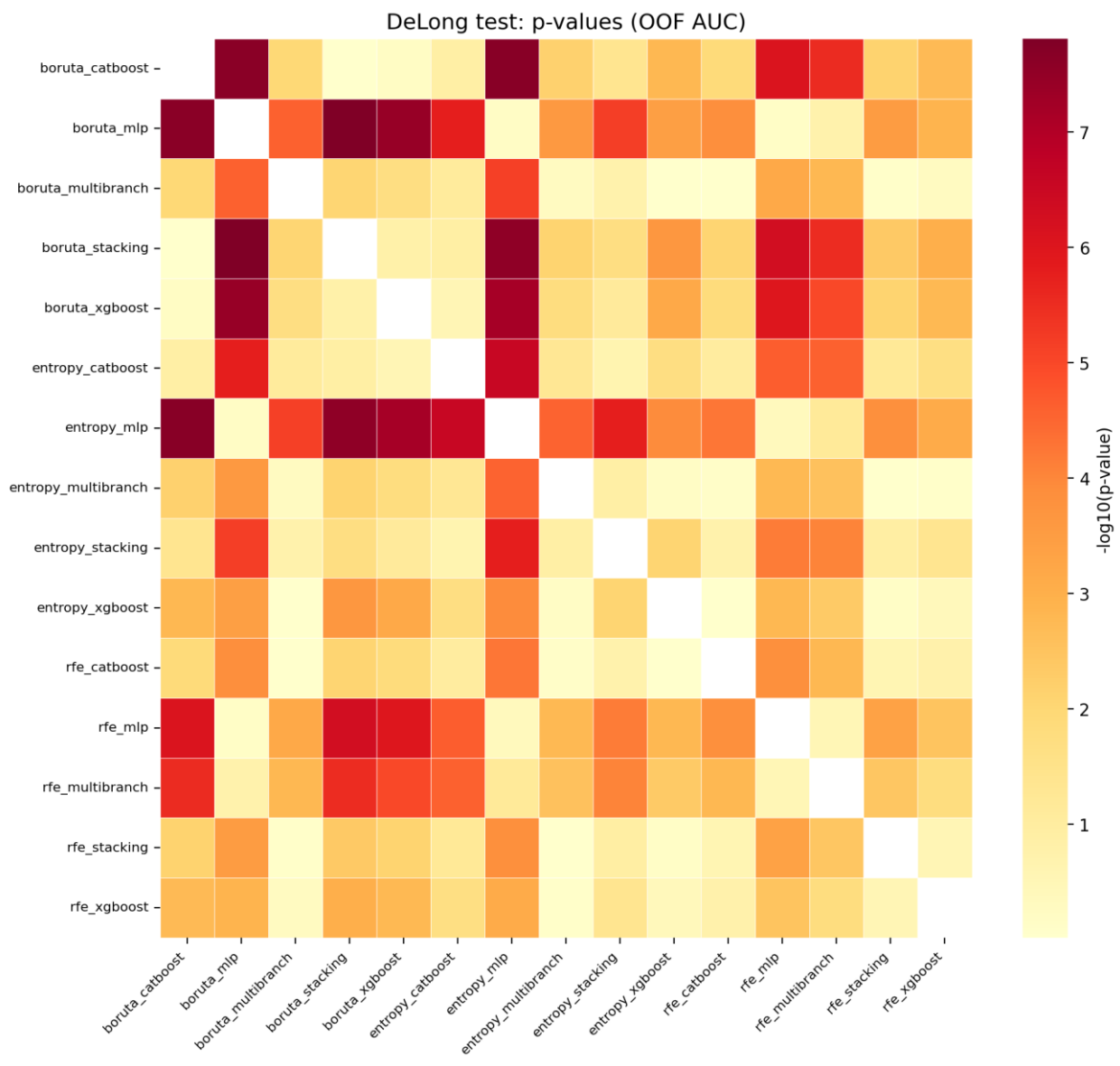


Рисунок 3.7 – Матриця р-значень тесту DeLong (попарне порівняння AUC)

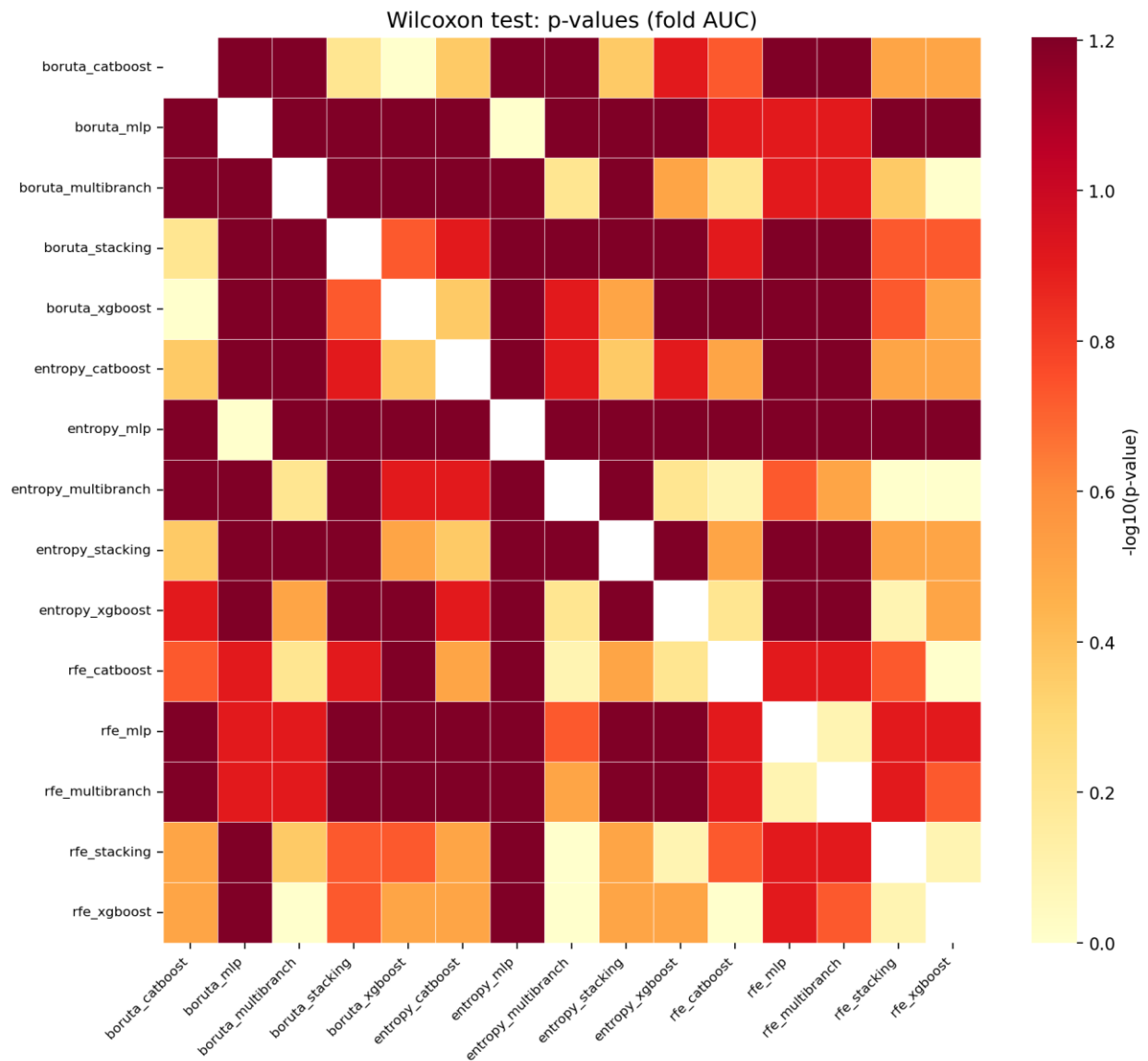


Рисунок 3.8 – Матриця р-значень критерію Вілкоксона

Аналіз показав таке. Найкраща модель (Boruta + Stacking, AUC = 0,846; 95 % ДІ [0,809; 0,879]) статистично значуще перевершує всі моделі на основі MLP ($p < 0,0001$ проти Entropy+MLP, Boruta+MLP, RFE+MLP), Multi-branch на RFE ($p < 0,0001$), а також Entropy+XGBoost ($p = 0,0002$). Водночас різниця між Boruta+Stacking і двома іншими найкращими моделями — Boruta+XGBoost ($p = 0,165$) та Boruta+CatBoost ($p = 0,964$) — статистично незначуща. Це означає, що три найкращі моделі на ознаках Boruta фактично рівнозначні за якістю, і

вибір серед них можна робити за іншими критеріями (швидкодія, простота, інтерпретованість).

Визначення оптимальної комбінації

На основі сукупного аналізу метрик і статистичної значущості оптимальною комбінацією визнано Boruta + Stacking з показниками OOF AUC = 0,846 (95 % бутстрап-довірчий інтервал [0,809; 0,879]; fold-mean AUC = 0,855 [0,831; 0,884]), F1 = 0,815, Recall = 0,862. Висока чутливість (Recall = 0,862) є вирішальною перевагою для скринінгової задачі, оскільки мінімізує пропуск хворих. Помірна специфічність (0,669) є наслідком свідомого вибору: поріг класифікації оптимізовано за F1-мірою, що зміщує баланс на користь чутливості. Для скринінгу це виправдано, оскільки хибна тривога (зайве дообстеження) значно менш небезпечна за пропуск хвороби; за потреби поріг можна налаштувати жорсткіше, підвищивши специфічність ціною чутливості. Якщо ж пріоритетом є простота і швидкодія, прийнятною альтернативою без статистично значущої втрати якості є Boruta + XGBoost або Boruta + CatBoost.

3.6 Клінічна інтерпретація відібраних ознак

Принцип інтерпретації

Висока точність моделі сама по собі не є достатньою для медичного застосування — лікар має розуміти, на яких показниках ґрунтується рішення системи. Тому окремим важливим етапом роботи є клінічна інтерпретація відібраних ознак: зіставлення їх кодових позначень з медичними назвами (за файлом Расшифровка.xlsx, польські назви) та обґрунтування їх ролі в патогенезі множинної мієломи.

Ключові ознаки, підтверджені трьома методами

Як показано в підрозділі 3.3, чотири ознаки були незалежно відібрані всіма трьома методами. Їх медичну інтерпретацію наведено в таблиці 3.6.

Таблиця 3.6 – Клінічна інтерпретація ознак, відібраних усіма трьома методами

<i>Код</i>	<i>Медична назва</i>	<i>Роль при ММ</i>
x15	Гемоглобін (Hemoglobina)	Знижений рівень — нормохромна анемія, одна з найчастіших і найхарактерніших лабораторних ознак ММ (критерій «А» у CRAB)
x121	β 1-глобулін (Globuliny beta 1)	Підвищення відображає парапротеїнемію — наявність моноклонального компонента (М-протеїну), патогномонічного біохімічного маркера ММ
x17	МСН (середній вміст гемоглобіну в еритроциті)	Еритроцитарний індекс, що характеризує тип анемії при ММ
x158	Калій (Potas)	Електролітний показник; зрушення пов'язані з порушенням функції нирок, типовим для ММ (критерій «R» у CRAB)

Особливо показовими є дві ознаки. Гемоглобін (x15) прямо відповідає компоненту «А» (Anemia) діагностичних критеріїв CRAB, розглянутих у підрозділі 1.1. β 1-глобулін (x121) відображає присутність моноклонального білка — головної біохімічної ознаки множинної мієломи, на виявленні якої ґрунтується вся лабораторна діагностика хвороби (підрозділ 1.2). Те, що три

методологічно різні алгоритми незалежно виділили саме ці показники, є вагомим підтвердженням їх діагностичної значущості та, опосередковано, валідності всього підходу.

Інші значущі ознаки

Решта ознак, відібраних окремими методами, також мають клінічне обґрунтування. Еритроцитарні індекси MCHC (x18), MCV (x19) та RDW (x20), відібрані ентропійним методом і Boruta, разом з гемоглобіном і MCH формують повну картину анемії. Загальний білірубін (x70), відібраний ентропійним методом і RFE, може відображати порушення печінково-ниркової функції у пацієнтів з MM. Показники тромбоцитів (MPV — x25) та лейкоцитарної формули (% еозинофілів — x5, % лімфоцитів — x3) доповнюють гематологічний профіль.

Важливо зазначити уточнення щодо розшифровки: ознака x70 відповідає загальному білірубіну (Total Bili), а не бікарбонатам, а x158 — калію (Potas), а не натрійуретичному пептиду, як могло б впливати з попередніх чернеткових припущень. Остаточна інтерпретація ґрунтується на польській версії файлу розшифровки, оскільки дані зібрані в польській клініці.

Важливості ознак за трьома методами візуалізовано на рисунках 3.9–3.11.

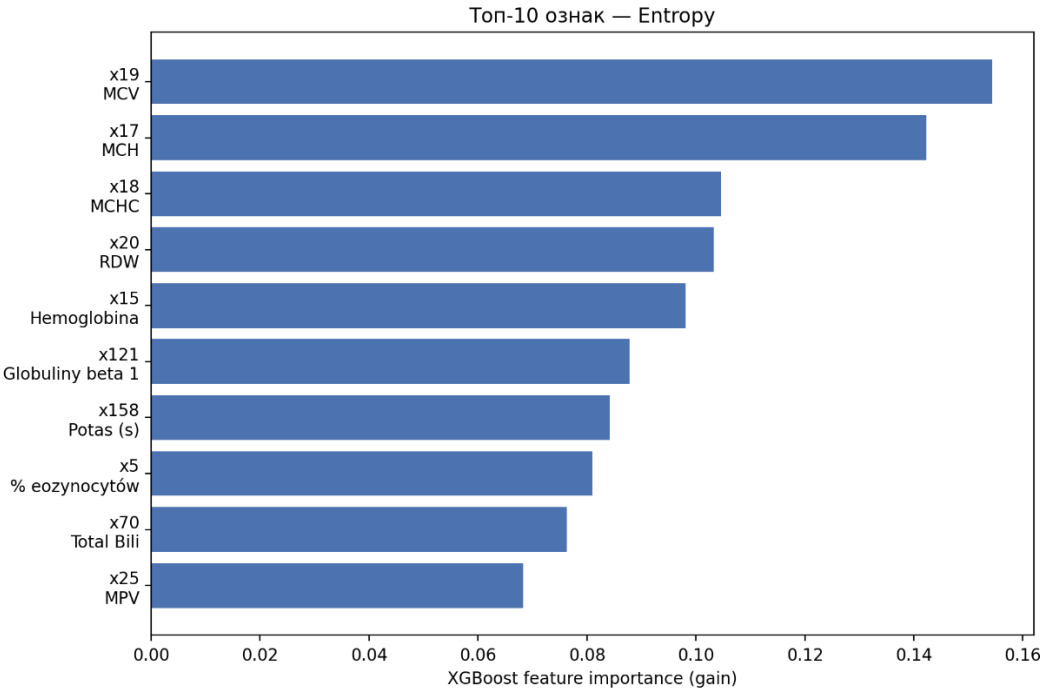


Рисунок 3.9 – Важливість ознак за ентропійним методом

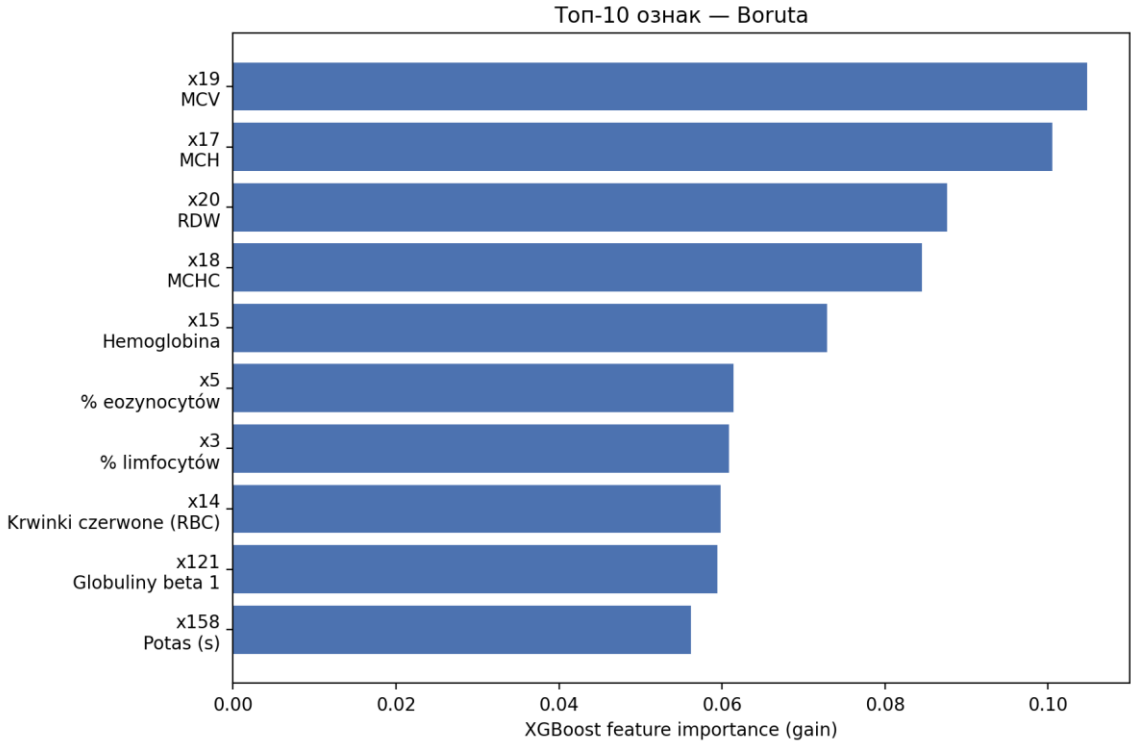


Рисунок 3.10 – Важливість ознак за методом Boruta

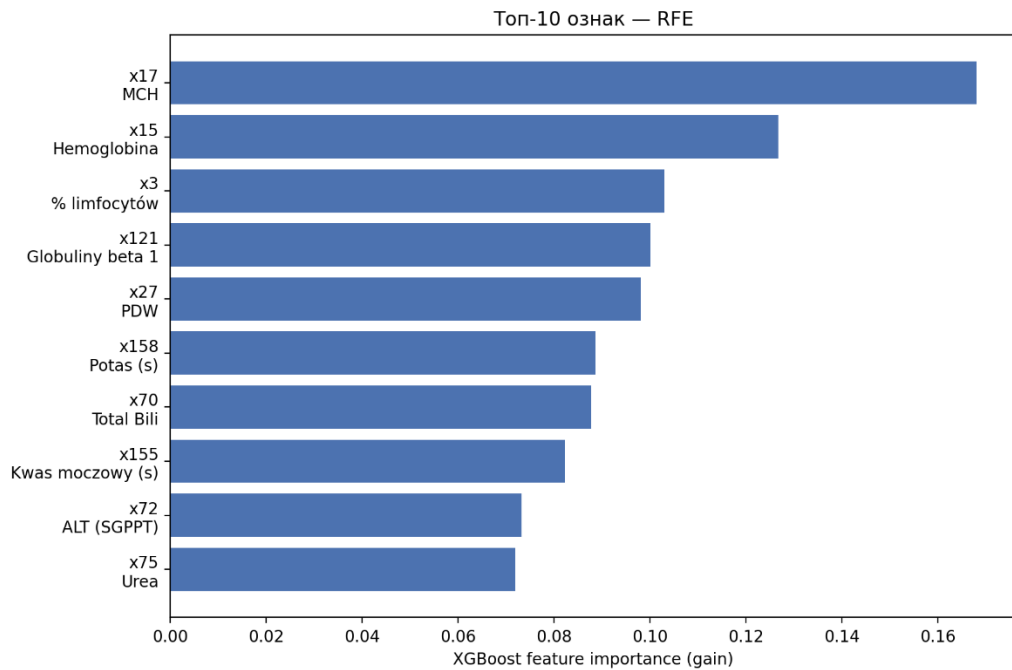


Рисунок 3.11 – Важливість ознак за методом RFE

Клінічне значення

Сукупність відібраних ознак формує осмислений з медичної точки зору профіль: анемія (гемоглобін, еритроцитарні індекси), парапротеїнемія (β 1-глобулін) та електролітно-метаболичні порушення (калій, білірубін). Це саме той комплекс відхилень, який характерний для невиявленої множинної мієломи на момент звернення пацієнта. Таким чином, розроблена система не лише досягає прийнятної точності класифікації, а й спирається на клінічно обґрунтовані показники, що підвищує довіру до неї з боку лікаря і відповідає вимозі інтерпретованості медичних ML-систем (підрозділ 1.3).

Висновки до розділу 3

У третьому розділі описано розробку та експериментальне дослідження програмної системи інтелектуального скринінгу множинної мієломи.

Розроблено систему модульної архітектури, що охоплює повний цикл — від завантаження даних до статистичного аналізу результатів. Забезпечено відтворюваність через фіксацію генератора випадкових чисел і централізовану конфігурацію. Деталі програмної реалізації наведено у Розділі 4.

Дослідження виконано на реальному наборі даних 525 пацієнтів польської гематологічної клініки (298 з ММ, 227 контроль). Реалізовано три методи відбору ознак (Information Gain, Boruta, RFE) та п'ять класифікаційних моделей (XGBoost, CatBoost, стекінг, Multi-branch, MLP), що склало матрицю з 15 експериментів, оцінених стратифікованою 5-кратною крос-валідацією.

Найкращою комбінацією визнано Boruta + Stacking з OOF AUC = 0,846 (95 % бутстрап-довірчий інтервал [0,809; 0,879]; fold-mean AUC = 0,855), F1 = 0,815 та чутливістю 0,862. Статистична значущість переваги над слабшими моделями підтверджена тестами DeLong і Вілкоксона ($p < 0,001$). Встановлено, що метод відбору Boruta дає найкращі результати, а ансамблеві моделі стабільно переважають багатосаровий перцептрон на цьому типі даних.

Проведено клінічну інтерпретацію відібраних ознак. Чотири показники — гемоглобін, $\beta 1$ -глобулін, МСН і калій — були незалежно відібрані всіма трьома методами, причому гемоглобін і $\beta 1$ -глобулін безпосередньо відповідають класичним діагностичним критеріям ММ (анемія та парапротеїнемія). Це підтверджує як діагностичну значущість цих маркерів, так і валідність розробленого підходу. Отримані результати демонструють придатність системи для використання як допоміжного інструмента попереднього скринінгу ММ на основі рутинних клініко-лабораторних показників.

РОЗДІЛ 4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОЇ ДІАГНОСТИКИ ММ

4.1 Загальна структура програмного проєкту

У попередньому розділі описано методику побудови системи та представлено експериментальні результати. У цьому розділі деталізовано її програмну реалізацію — конкретні модулі коду, які забезпечують роботу кожного етапу обчислювального конвеєра. Систему реалізовано мовою Python 3 як модульний проєкт з ієрархічною організацією каталогів. Поділ на окремі модулі дозволяє запускати будь-який етап обробки самостійно — підготовку даних, відбір ознак, навчання моделей, оцінювання чи статистичний аналіз — не виконуючи щоразу весь конвеєр спочатку. Такий підхід полегшує налагодження окремих компонентів і робить дослідження відтворюваним.

Загальну організацію проєкту показано на рисунку 4.1.

Структура програмного проєкту

```
myeloma-screening/
├── data/
│   ├── raw/ ← вихідні .xlsx
│   └── interim/ ← проміжні дані
├── src/screening/ ← програмний код
│   ├── config.py
│   ├── data_loader.py
│   ├── preprocessing.py
│   ├── feature_selection.py
│   ├── models.py
│   ├── cross_validation.py
│   ├── run_experiments.py
│   ├── statistics.py
│   ├── plots.py
│   └── clinical_interpretation.py
├── results/ ← таблиці, графіки, статистика
├── tests/ ← модульні тести (pytest)
├── notebooks/ ← дослідницькі ноутбуки
└── requirements.txt
```

Рисунок 4.1 – Структура програмного проєкту

Каталог data/ розділено на raw/ та interim/. У raw/ зберігаються вихідні файли у такому вигляді, як їх отримано (Bone_normalized_data.xlsx, DatasetCleanedEntropia.xlsx, Расшифровка.xlsx), а в interim/ потрапляють проміжні результати роботи скриптів. Вихідні дані ніколи не перезаписуються, що гарантує відтворюваність обчислень. Увесь програмний код зосереджено в пакеті src/screening/, а всі результати роботи системи — таблиці, графіки та статистичні матриці — зберігаються в каталозі results/ з поділом на підкаталоги tables/, figures/, stats/ і predictions/.

Середовищем розробки слугував Visual Studio Code. Перелік залежностей зафіксовано у файлі requirements.txt, що дозволяє відтворити робоче оточення однією командою. Основні бібліотеки, задіяні в системі, та їх призначення наведено в таблиці 4.1.

Таблиця 4.1 – Програмні бібліотеки та їх призначення

<i>Бібліотека</i>	<i>Версія</i>	<i>Призначення</i>
NumPy	≥1.26	Числові масиви та векторні операції
pandas	≥2.1	Завантаження та обробка табличних даних
SciPy	≥1.12	Статистичні функції (критерій Вілкоксона)
scikit-learn	≥1.4	RFE, крос-валідація, метрики, логістична регресія
openpyxl	≥3.1	Читання Excel-файлів
Boruta	≥0.3	Відбір ознак методом тіньових атрибутів
XGBoost	≥2.0	Градiєнтний бустинг (модель XGBoost)
CatBoost	≥1.2	Градiєнтний бустинг з упорядкуванням
PyTorch	≥2.2	Нейромережні моделі (MLP, Multi-branch)
matplotlib	≥3.8	Побудова графіків навчання та результатів
seaborn	≥0.13	Візуалізація теплових карт
matplotlib-venn	≥0.11	Діаграма Венна для перетину ознак
pytest	≥8.0	Модульне тестування

4.2 Модуль конфігурації та завантаження даних

Усі параметри експерименту зосереджено в одному модулі `config.py`, що містить клас `Config`. У ньому задано шляхи до даних і результатів, назви вхідних файлів, цільову колонку (Y), а також ключові параметри відтворюваності: фіксоване зерно генератора випадкових чисел (`RANDOM_SEED = 123`, успадковане з референсних R-скриптів), кількість фолдів крос-валідації (`CV_FOLDS = 5`), частку внутрішньої валідаційної підмножини (`INNER_VAL_SIZE = 0.15`) та число бутстрап-повторів (`BOOTSTRAP_N = 1000`). Тут же визначено розподіл ознак на три гілки для багатогілкової мережі. Винесення всіх констант у єдиний клас виключає «магічні числа» в коді й централізує керування експериментом.

Модуль `data_loader.py` відповідає за зчитування даних. Функція `load_entropia()` повертає робочий набір з десяти ознак, `load_bone()` — повний набір з 213 ознак для алгоритмів відбору, а `load_decoder()` зчитує таблицю розшифровки. Окрема функція `_normalize_target()` приводить цільову змінну до єдиного формату, а `get_feature_description()` дозволяє отримати медичну назву ознаки за її кодом — це використовується на етапі клінічної інтерпретації. Модуль `preprocessing.py` готує повний набір ознак для Boruta та RFE, перевіряючи відсутність пропусків і коректність типів даних.

4.3 Модуль відбору ознак

Усі три методи відбору реалізовано в модулі `feature_selection.py`. Ентропійний набір зчитується безпосередньо з готового файлу. Метод Boruta обгортає класифікатор випадкового лісу зі збалансованими вагами класів і

виконує статистичне тестування ознак проти їх перемішаних копій (тіньових атрибутів) упродовж заданої кількості ітерацій. Метод RFE рекурсивно вилучає найменш важливі ознаки на основі логістичної регресії, доки не залишиться задана їх кількість.

Окрема функція обчислює перетин трьох наборів ознак і будує діаграму Венна. Результати кожного методу зберігаються у вигляді CSV-таблиць із рангами ознак, а зведена таблиця перетинів фіксує, які показники відібрані кількома методами одночасно. Саме ця функція виявила чотири ознаки, спільні для всіх трьох методів.

4.4 Модуль моделей

Модуль `models.py` містить побудову всіх п'яти класифікаторів та допоміжні функції оцінювання. Функція `compute_scale_pos_weight()` обчислює коефіцієнт балансування класів за співвідношенням кількості негативних і позитивних прикладів (N_0/N_1), `find_best_threshold()` підбирає оптимальний поріг класифікації за максимумом F1-міри, а `compute_metrics()` обчислює повний набір метрик якості.

Моделі XGBoost і CatBoost будуються функціями `build_xgboost()` та `build_catboost()` з гіперпараметрами, успадкованими з референсних скриптів керівника. Стекінг реалізовано класом `StackingModel`, який навчає базові моделі XGBoost і CatBoost, а потім логістичну регресію на їх прогнозах. Нейромережні моделі описано як класи PyTorch: `SimpleMLP` — повнозв'язна мережа зі структурою «вхід → 32 → 16 → 1» з активаціями ReLU та проміжними шарами Dropout; `MyelomaMultiBranchNet` — багатогілкова мережа, у якій кілька кодувальників `BranchEncoder` обробляють різні групи ознак, а модуль `GatedFusion` об'єднує їх виходи через зважування softmax-

вагами. Універсальна функція `fit_with_early_stopping()` забезпечує єдиний інтерфейс навчання з ранньою зупинкою для деревних моделей, а `predict_proba_positive()` повертає ймовірності позитивного класу для будь-якої моделі.

4.5 Модуль крос-валідації

Модуль `cross_validation.py` реалізує схему оцінювання якості. Функція `run_cv()` організовує стратифіковану п'ятикратну крос-валідацію: на кожній ітерації один фолд відводиться під тестування, а решта розбивається на навчальну та внутрішню валідаційну підмножини (функція `_inner_train_val_split()`). Валідаційна підмножина використовується для ранньої зупинки навчання та підбору порога класифікації, що запобігає підгляданню в тестові дані.

Для нейромережних моделей навчання виконує окрема функція `_fit_torch_classifier()` з оптимізатором AdamW (Adam with Weight Decay — Алгоритм оптимізації, який є вдосконаленою версією класичного оптимізатора Adam — Adaptive Moment Estimation) та зваженою функцією втрат для врахування дисбалансу класів. Після проходження всіх фолдів функція `_aggregate_cv_metrics()` усереднює метрики та обчислює стандартні відхилення, а `save_oof_predictions()` зберігає прогнози (out-of-fold) для подальшого статистичного аналізу. Збереження прогнозів є обов'язковим, оскільки тести DeLong і бутстрап працюють саме з ними, а не лише з підсумковими метриками.

4.6 Модуль статистичного аналізу

Модуль `statistics.py` реалізує три методи оцінки значущості. Тест DeLong для порівняння AUC двох моделей реалізовано з нуля за оригінальним алгоритмом — допоміжні функції `compute_midrank()` та `_fast_delong()` обчислюють статистику на основі мідрангів, а `delong_test()` повертає р-значення. Критерій Вілкоксона (`wilcoxon_test()`) спирається на реалізацію зі SciPy і порівнює метрики моделей за фолдами. Бутстрап-довірчі інтервали обчислюються функціями `bootstrap_ci()` та `bootstrap_auc_ci()` методом перцентилів на основі тисячі повторних вибірок із поверненням.

Функції `compute_delong_matrix()` і `compute_wilcoxon_matrix()` будують матриці попарних р-значень для всіх 15 експериментів, які потім візуалізуються тепловими картами через `plot_pvalue_heatmap()`. Головна функція `run_all_statistics()` об'єднує весь аналіз і зберігає результати у відповідні CSV-файли та зображення.

4.7 Оркестрація експериментів та візуалізація

Модуль `run_experiments.py` керує повним циклом дослідження. Функція `prepare_feature_sets()` формує три набори ознак, `get_dataset_for_method()` повертає відповідні дані для кожного методу відбору, а `run_all_experiments()` послідовно виконує всі 15 комбінацій «метод відбору × модель», для кожної з яких викликає крос-валідацію, збирає метрики у спільну таблицю та зберігає прогнози. Підсумкова таблиця результатів формується функцією `_summary_row_from_cv()` і виводиться у консоль та у файл `experiment_results.csv`.

Модуль `plots.py` відповідає за всю візуалізацію: ROC-криві для кожного методу та зведений графік усіх кривих, стовпчикові діаграми порівняння метрик, матрицю помилок найкращої моделі, графіки важливості ознак, діаграми розмаху за фолдами та теплові карти p -значень. Модуль `clinical_interpretation.py` зіставляє відібрані ознаки з їх медичними назвами з таблиці розшифровки та формує підсумкову таблицю клінічної інтерпретації.

4.8 Тестування системи

Тестування проводилося на трьох рівнях — модульному, інтеграційному та системному.

На модульному рівні перевірялася коректність роботи окремих функцій. Для модуля завантаження даних контролювалося, що набори мають очікувану розмірність (525 записів), цільова колонка приводиться до значень 0 і 1, а пропуски відсутні. Для модуля відбору ознак перевірялося, що кожен метод повертає несутеречливий список ознак, а перетин обчислюється правильно. Для моделей контролювалося, що прогнозовані ймовірності лежать у діапазоні $[0, 1]$, а коефіцієнт балансування обчислюється коректно.

На інтеграційному рівні перевірялася злагоджена робота ланцюга «завантаження → відбір → навчання → оцінювання» на одному фолді, а також коректність збереження проміжних артефактів. Окрему увагу приділено відтворюваності: повторний запуск з тим самим зерном генератора випадкових чисел давав ідентичні результати.

На системному рівні виконувався повний прогін усіх 15 експериментів зі збереженням таблиць, графіків і статистичних матриць. Перевірялося, що підсумкові метрики узгоджуються між собою (наприклад, $F1$ відповідає обчисленим $precision$ і $recall$), а статистичні тести повертають значення в

допустимих межах. Результати системного прогону повністю відповідають даним, наведеним у розділі 3.

Висновки до розділу 4

У четвертому розділі описано програмну реалізацію системи інтелектуального скринінгу множинної мієломи. Систему побудовано мовою Python 3 як модульний проєкт, де кожен етап обробки винесено в окремий компонент пакета `src/screening/`. Централізований модуль конфігурації забезпечує відтворюваність через фіксацію всіх параметрів та зерна генератора випадкових чисел.

Реалізовано повний конвеєр: завантаження даних, три методи відбору ознак, п'ять класифікаційних моделей, стратифіковану крос-валідацію з підбором порога, а також блок статистичного аналізу з власною реалізацією тесту DeLong, критерію Вілкоксона та бутстрап-інтервалів. Окремі модулі відповідають за візуалізацію результатів і клінічну інтерпретацію відібраних ознак. Трирівневе тестування — модульне, інтеграційне та системне — підтвердило коректність роботи системи та відтворюваність отриманих результатів.

ВИСНОВКИ

У дипломній роботі розв’язано актуальну науково-практичну задачу — розроблено програмну систему інтелектуального скринінгу множинної мієломи на основі клініко-лабораторних показників пацієнта з використанням методів інтелектуального аналізу даних. У ході виконання роботи отримано такі основні результати.

Проаналізовано предметну область. Розглянуто множинну мієлому як медичну задачу: її епідеміологію, клінічну картину, діагностичні критерії та проблему пізнього виявлення. Встановлено, що значна частина показників, які використовуються при діагностиці ММ, входить до рутинного клініко-лабораторного обстеження, що створює підґрунтя для застосування методів машинного навчання як інструмента попереднього скринінгу.

Досліджено теоретичні основи методів. Розглянуто три методологічно різнотипні методи відбору ознак — ентропійний (Information Gain), обгортковий (RFE) та на основі ансамблю (Boruta), а також п’ять класифікаційних моделей: XGBoost, CatBoost, стекінг, багатогілкову нейронну мережу та багат шаровий перцептрон. Описано метрики оцінювання якості та методи статистичної перевірки значущості відмінностей.

Виконано попередню обробку даних. Підготовлено реальний набір даних 525 пацієнтів польської гематологічної клініки (298 з підтвердженою ММ, 227 — контрольна група) з нормалізацією показників та аналізом збалансованості класів. Дисбаланс класів компенсовано зважуванням у моделях замість синтетичної генерації прикладів.

Реалізовано програмну систему мовою Python. Систему побудовано за модульним принципом (вісім функціональних модулів — детальний опис у Розділі 4), із забезпеченням повної відтворюваності результатів через фіксацію генератора випадкових чисел. Гіперпараметри моделей узгоджено з референсними скриптами наукового керівника.

Проведено експериментальне дослідження. Виконано 15 експериментів за схемою «три методи відбору × п'ять моделей» з оцінюванням стратифікованою 5-кратною крос-валідацією та повним набором метрик (Accuracy, Precision, Recall, Specificity, F1, AUC).

Виконано порівняльний аналіз зі статистичною оцінкою. Найкращою комбінацією визнано Boruta + Stacking з показниками OOF AUC = 0,846 (95 % бутстрап-довірчий інтервал [0,809; 0,879]), F1 = 0,815 та чутливістю 0,862. Статистичну значущість переваги над слабшими моделями підтверджено тестами DeLong і Вілкоксона ($p < 0,001$). Встановлено, що метод відбору Boruta забезпечує найкращі результати, а ансамблеві методи стабільно переважають багат шаровий перцептрон на даному типі табличних медичних даних.

Досліджено клінічну значущість ознак. Виявлено, що чотири показники — гемоглобін, $\beta 1$ -глобулін, МСН та калій — незалежно відібрані всіма трьома методами. Гемоглобін і $\beta 1$ -глобулін безпосередньо відповідають класичним діагностичним критеріям ММ (анемія та парапротеїнемія), що підтверджує як діагностичну цінність цих маркерів, так і валідність розробленого підходу.

Сформульовано рекомендації. Розроблена система придатна для використання як допоміжний інструмент попереднього скринінгу ММ на основі рутинних клініко-лабораторних показників. Завдяки високій чутливості (0,862) вона мінімізує ризик пропуску хворих, що є критичним для скринінгових задач. Для практичного застосування рекомендовано комбінацію Boruta + Stacking; за потреби у швидкодії прийнятними альтернативами без статистично значущої втрати якості є Boruta + XGBoost або Boruta + CatBoost.

Наукова новизна роботи полягає у систематичному порівнянні трьох методологічно різнотипних методів відбору ознак у поєднанні з сучасними ансамблевими та нейромережними класифікаторами для задачі скринінгу множинної мієломи, із обов'язковою статистичною оцінкою значущості

відмінностей — підхід, який у доступній літературі для цієї задачі практично не представлений.

Практичне значення полягає у створенні працездатної програмної системи, яка автоматизує аналіз клініко-лабораторних показників, виявляє найбільш інформативні з них та оцінює ймовірність наявності захворювання, сприяючи ранньому виявленню ММ.

Обмеження дослідження. Робота має такі обмеження: помірний розмір вибірки (525 пацієнтів); дані зібрані в одному медичному центрі (Люблін, Польща), що обмежує пряме узагальнення на інші популяції; розглянуто лише бінарну класифікацію без розрізнення стадій захворювання. Отримане значення OOF AUC (0,846) дещо поступається окремим роботам зі значно більшими вибірками, що є очікуваним для набору такого обсягу.

Напрями подальших досліджень. Перспективними є: розширення та зовнішня валідація моделі на мультицентрових даних; перехід від бінарної класифікації до розрізнення стадій ММ; інтеграція методів пояснюваності (SHAP) для індивідуальних прогнозів; розробка повноцінного програмного інтерфейсу для використання системи лікарями-практиками.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bray F., Laversanne M., Sung H., et al. Global cancer statistics 2022: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: A Cancer Journal for Clinicians*. 2024. Vol. 74, No. 3. P. 229–263. DOI: <https://doi.org/10.3322/caac.21834>.
2. Kazandjian D. Multiple myeloma epidemiology and survival: A unique malignancy. *Seminars in Oncology*. 2016. Vol. 43, No. 6. P. 676–681. DOI: <https://doi.org/10.1053/j.seminoncol.2016.11.004>.
3. Rajkumar S.V. Multiple myeloma: 2022 update on diagnosis, risk stratification, and management. *American Journal of Hematology*. 2022. Vol. 97, No. 8. P. 1086–1107. DOI: <https://doi.org/10.1002/ajh.26590>.
4. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. Cambridge: MIT Press, 2016. 800 p.
5. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. P. 785–794. DOI: <https://doi.org/10.1145/2939672.2939785>.
6. Wolpert D.H. Stacked generalization. *Neural Networks*. 1992. Vol. 5, No. 2. P. 241–259. DOI: [https://doi.org/10.1016/S0893-6080\(05\)80023-1](https://doi.org/10.1016/S0893-6080(05)80023-1).
7. LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature*. 2015. Vol. 521, No. 7553. P. 436–444. DOI: <https://doi.org/10.1038/nature14539>.
8. Kursu M.B., Rudnicki W.R. Feature Selection with the Boruta Package. *Journal of Statistical Software*. 2010. Vol. 36, No. 11. P. 1–13. DOI: <https://doi.org/10.18637/jss.v036.i11>.
9. Guyon I., Weston J., Barnhill S., Vapnik V. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*. 2002. Vol. 46, No. 1–3. P. 389–422. DOI: <https://doi.org/10.1023/A:1012487302797>.
10. Palumbo A., Avet-Loiseau H., Oliva S., et al. Revised International Staging System for Multiple Myeloma: A Report From International Myeloma Working Group. *Journal of Clinical Oncology*. 2015. Vol. 33, No. 26. P. 2863–2869. DOI: <https://doi.org/10.1200/JCO.2015.61.2267>.
11. Greipp P.R., San Miguel J., Durie B.G.M., et al. International Staging System for Multiple Myeloma. *Journal of Clinical Oncology*. 2005. Vol. 23, No. 15. P. 3412–3420. DOI: <https://doi.org/10.1200/JCO.2005.04.242>.

12. Kyle R.A., Rajkumar S.V. Multiple myeloma. *Blood*. 2008. Vol. 111, No. 6. P. 2962–2972. DOI: <https://doi.org/10.1182/blood-2007-10-078022>.
13. Kumar S.K., Rajkumar S.V., Dispenzieri A., et al. Improved survival in multiple myeloma and the impact of novel therapies. *Blood*. 2008. Vol. 111, No. 5. P. 2516–2520. DOI: <https://doi.org/10.1182/blood-2007-10-116129>.
14. Durie B.G.M., Salmon S.E. A clinical staging system for multiple myeloma. *Cancer*. 1975. Vol. 36, No. 3. P. 842–854. DOI: [https://doi.org/10.1002/1097-0142\(197509\)36:3<842::AID-CNCR2820360303>3.0.CO;2-U](https://doi.org/10.1002/1097-0142(197509)36:3<842::AID-CNCR2820360303>3.0.CO;2-U).
15. Rajkumar S.V., Dimopoulos M.A., Palumbo A., et al. International Myeloma Working Group updated criteria for the diagnosis of multiple myeloma. *The Lancet Oncology*. 2014. Vol. 15, No. 12. P. e538–e548. DOI: [https://doi.org/10.1016/S1470-2045\(14\)70442-5](https://doi.org/10.1016/S1470-2045(14)70442-5).
16. Mitchell T.M. *Machine Learning*. New York: McGraw Hill, 1997. 414 p.
17. Національний канцер-реєстр України. Бюлетень: Рак в Україні. Київ: Національний інститут раку, 2023. URL: <http://www.ncru.inf.ua/publications/>.
18. Kyle R.A., Rajkumar S.V. Criteria for diagnosis, staging, risk stratification and response assessment of multiple myeloma. *Leukemia*. 2009. Vol. 23, No. 1. P. 3–9. DOI: <https://doi.org/10.1038/leu.2008.291>.
19. Hillengass J., Usmani S., Rajkumar S.V., et al. International myeloma working group consensus recommendations on imaging in monoclonal plasma cell disorders. *The Lancet Oncology*. 2019. Vol. 20, No. 6. P. e302–e312. DOI: [https://doi.org/10.1016/S1470-2045\(19\)30309-2](https://doi.org/10.1016/S1470-2045(19)30309-2).
20. Dimopoulos M.A., Moreau P., Terpos E., et al. Multiple myeloma: EHA-ESMO Clinical Practice Guidelines for diagnosis, treatment and follow-up. *Annals of Oncology*. 2021. Vol. 32, No. 3. P. 309–322. DOI: <https://doi.org/10.1016/j.annonc.2020.11.014>.
21. Topol E.J. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*. 2019. Vol. 25, No. 1. P. 44–56. DOI: <https://doi.org/10.1038/s41591-018-0300-7>.
22. Esteva A., Robicquet A., Ramsundar B., et al. A guide to deep learning in healthcare. *Nature Medicine*. 2019. Vol. 25, No. 1. P. 24–29. DOI: <https://doi.org/10.1038/s41591-018-0316-z>.
23. Litjens G., Kooi T., Bejnordi B.E., et al. A survey on deep learning in medical image analysis. *Medical Image Analysis*. 2017. Vol. 42. P. 60–88. DOI: <https://doi.org/10.1016/j.media.2017.07.005>.

24. Mosquera Orgueira A., González Pérez M.S., Díaz Arias J.Á., et al. Survival prediction and treatment optimization of multiple myeloma patients using machine-learning models based on clinical and gene expression data. *Leukemia*. 2021. Vol. 35, No. 10. P. 2924–2935. DOI: <https://doi.org/10.1038/s41375-021-01286-2>.
25. Rudin C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*. 2019. Vol. 1, No. 5. P. 206–215. DOI: <https://doi.org/10.1038/s42256-019-0048-x>.
26. Lundberg S.M., Lee S.-I. A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. Vol. 30. P. 4765–4774.
27. Cai J., Liu Z., Wang Y., Yang W., Sun Z., You C. Construction of the prediction model for multiple myeloma based on machine learning. *International Journal of Laboratory Hematology*. 2024. Vol. 46, No. 5. P. 918–926. DOI: <https://doi.org/10.1111/ijlh.14324>.
28. Peng T., Liu L., Liu F., et al. Machine learning-based infection prediction model for newly diagnosed multiple myeloma patients. *Frontiers in Neuroinformatics*. 2023. Vol. 16. Article 1063610. DOI: <https://doi.org/10.3389/fninf.2022.1063610>.
29. Allegra A., Tonacci A., Sciacotta R., et al. Machine learning and deep learning applications in multiple myeloma diagnosis, prognosis, and treatment selection. *Cancers*. 2022. Vol. 14, No. 3. Article 606. DOI: <https://doi.org/10.3390/cancers14030606>.
30. Bao L., Wang Y.-T., Zhuang J.-L., et al. Machine Learning-Based Overall Survival Prediction of Elderly Patients With Multiple Myeloma From Multicentre Real-Life Data. *Frontiers in Oncology*. 2022. Vol. 12. Article 922039. DOI: <https://doi.org/10.3389/fonc.2022.922039>.
31. Ren L., Xu B., Xu J., et al. A Machine Learning Model to Predict Survival and Therapeutic Responses in Multiple Myeloma. *International Journal of Molecular Sciences*. 2023. Vol. 24, No. 7. Article 6683. DOI: <https://doi.org/10.3390/ijms24076683>.
32. Hussain Z., De Brouwer E., Boiarsky R., et al. Joint AI-driven event prediction and longitudinal modeling in newly diagnosed and relapsed multiple myeloma. *npj Digital Medicine*. 2024. Vol. 7, No. 1. Article 174. DOI: <https://doi.org/10.1038/s41746-024-01189-3>.
33. Macalled S., Houfani D., Bouchareb S. Preliminary analysis of explainable machine learning methods for multiple myeloma chemotherapy treatment

- recognition. *Evolutionary Intelligence*. 2023. Vol. 16. DOI: <https://doi.org/10.1007/s12065-023-00833-3>.
34. Chawla N.V., Bowyer K.W., Hall L.O., Kegelmeyer W.P. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*. 2002. Vol. 16. P. 321–357. DOI: <https://doi.org/10.1613/jair.953>.
 35. Shannon C.E. A Mathematical Theory of Communication. *The Bell System Technical Journal*. 1948. Vol. 27, No. 3. P. 379–423. DOI: <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>.
 36. Guyon I., Elisseeff A. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*. 2003. Vol. 3. P. 1157–1182.
 37. Breiman L. Random Forests. *Machine Learning*. 2001. Vol. 45, No. 1. P. 5–32. DOI: <https://doi.org/10.1023/A:1010933404324>.
 38. Friedman J.H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*. 2001. Vol. 29, No. 5. P. 1189–1232. DOI: <https://doi.org/10.1214/aos/1013203451>.
 39. Prokhorenkova L., Gusev G., Vorobev A., Dorogush A.V., Gulin A. CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems (NeurIPS)*. 2018. Vol. 31. P. 6639–6649.
 40. Glorot X., Bordes A., Bengio Y. Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011. Vol. 15. P. 315–323.
 41. Rumelhart D.E., Hinton G.E., Williams R.J. Learning representations by back-propagating errors. *Nature*. 1986. Vol. 323, No. 6088. P. 533–536. DOI: <https://doi.org/10.1038/323533a0>.
 42. Kingma D.P., Ba J. Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. 2015. arXiv:1412.6980.
 43. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014. Vol. 15, No. 1. P. 1929–1958.
 44. Fawcett T. An introduction to ROC analysis. *Pattern Recognition Letters*. 2006. Vol. 27, No. 8. P. 861–874. DOI: <https://doi.org/10.1016/j.patrec.2005.10.010>.
 45. DeLong E.R., DeLong D.M., Clarke-Pearson D.L. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*. 1988. Vol. 44, No. 3. P. 837–845. DOI: <https://doi.org/10.2307/2531595>.

46. Efron B., Tibshirani R.J. *An Introduction to the Bootstrap*. New York: Chapman & Hall/CRC, 1993. 456 p.
47. Powers D.M.W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*. 2011. Vol. 2, No. 1. P. 37–63.
48. Babichev S., Lytvynenko V., Gozhyj A., Korobchynskiy M., Voronenko M. A Fuzzy Model for Gene Expression Profiles Reducing Based on the Complex Use of Statistical Criteria and Shannon Entropy. *Advances in Computer Science for Engineering and Education (ICCSEEA 2018). Advances in Intelligent Systems and Computing*. Cham: Springer, 2019. Vol. 754. P. 545–554. DOI: https://doi.org/10.1007/978-3-319-91008-6_55.
49. Sineglazov V., Riazanovskiy K., Klanovets A., Chumachenko E., Linnik N. Intelligent tuberculosis activity assessment system based on an ensemble of neural networks. *Computers in Biology and Medicine*. 2022. Vol. 147. Article 105800. DOI: <https://doi.org/10.1016/j.compbimed.2022.105800>.

ДОДАТОК А ЛІСТИНГ КОДУ

clinical_interpretation.py :

```
"""Клінічна інтерпретація відібраних ознак ."""
```

```
from __future__ import annotations
```

```
import argparse
```

```
import logging
```

```
from pathlib import Path
```

```
from typing import Any
```

```
import pandas as pd
```

```
from src.screening.config import Config
```

```
from src.screening.data_loader import load_decoder
```

```
from src.screening.feature_selection import ENTROPY_FEATURES
```

```
from src.screening.run_experiments import FEATURE_METHODS,  
prepare_feature_sets
```

```
logger = logging.getLogger(__name__)
```

```
METHOD_LABELS: dict[str, str] = {
```

```
    "entropy": "Entropy",
```

```
    "boruta": "Boruta",
```

```
    "rfe": "RFE",
```

```
}
```

Патогномонічні / ключові ознаки для ММ (з ТЗ §10)

MM_PATHOGNOMONIC: dict[str, dict[str, str]] = {

"x15": {

"label": "Hb (гемоглобін)",

"comment": (

"Низький гемоглобін — типова нормохромна анемія при ММ; "

"одна з найчастіших лабораторних ознак для скринінгу."

),

},

"x121": {

"label": "β1-глобулін",

"comment": (

"Підвищення β1-глобуліну відображає парапротеїнемію / "

"моноклональний компонент — ключовий біохімічний маркер
ММ."

),

},

"x70": {

"label": "Білірубін загальний (Total Bili)",

"comment": (

"Загальний білірубін (НЕ бікарбонати): можливі порушення "

"печінково-ниркової функції у пацієнтів з ММ."

),

},

}

```
def _load_rank_map(method: str) -> dict[str, int]:
    """Завантажити ранги ознак з CSV (Boruta/RFE)."""
    path = Config.RESULTS_TABLES_DIR / f"{method}_features.csv"
    if not path.exists():
        return {}
    df = pd.read_csv(path)
    if "rank" not in df.columns:
        return {}
    col = "x_id" if "x_id" in df.columns else df.columns[0]
    return {str(row[col]): int(row["rank"]) for _, row in df.iterrows()}
```

```
def _lookup_names(x_id: str, decoder: pd.DataFrame) -> tuple[str, str]:
    """Повернути (name_pl, name_ru) для x_id."""
    row = decoder.loc[decoder["x_id"] == x_id]
    if row.empty:
        alt = decoder.loc[decoder["x_id_alt"] == x_id]
        if alt.empty:
            return "?", "?"
        row = alt
    return str(row.iloc[0]["name_pl"]), str(row.iloc[0]["name_ru"])
```

```
def build_clinical_table(
    feature_sets: dict[str, list[str]] | None = None,
) -> pd.DataFrame:
```

"""Побудувати таблицю клінічної інтерпретації для всіх методів відбору.

Args:

feature_sets: Словник {method: [x_id, ...]}; за замовчуванням з CSV.

Returns:

DataFrame з розшифровкою та коментарями для кожної унікальної ознаки.

```

"""
feature_sets = feature_sets or
prepare_feature_sets(methods=FEATURE_METHODS)
decoder = load_decoder()

boruta_ranks = _load_rank_map("boruta")
rfe_ranks = _load_rank_map("rfe")

all_features = sorted(
    set(feature_sets.get("entropy", ENTROPY_FEATURES))
    | set(feature_sets.get("boruta", []))
    | set(feature_sets.get("rfe", []))
)

rows: list[dict[str, Any]] = []
for x_id in all_features:
    name_pl, name_ru = _lookup_names(x_id, decoder)
    in_entropy = x_id in feature_sets.get("entropy", [])
    in_boruta = x_id in feature_sets.get("boruta", [])
    in_rfe = x_id in feature_sets.get("rfe", [])
    n_methods = sum([in_entropy, in_boruta, in_rfe])

```

```

mm_info = MM_PATHOGNOMONIC.get(x_id)
rows.append(
    {
        "x_id": x_id,
        "name_pl": name_pl,
        "name_ru": name_ru,
        "entropy": int(in_entropy),
        "boruta": int(in_boruta),
        "rfe": int(in_rfe),
        "n_methods": n_methods,
        "boruta_rank": boruta_ranks.get(x_id),
        "rfe_rank": rfe_ranks.get(x_id),
        "mm_key_marker": int(x_id in MM_PATHOGNOMONIC),
        "mm_key_label": mm_info["label"] if mm_info else "",
        "clinical_comment": mm_info["comment"] if mm_info else "",
    }
)

df = pd.DataFrame(rows)

df = df.sort_values(["mm_key_marker", "n_methods", "x_id"],
ascending=[False, False, True])

return df.reset_index(drop=True)

def build_method_summary(feature_sets: dict[str, list[str]]) ->
pd.DataFrame:
    """Короткий підсумок по кожному методу відбору."""

```

```

rows: list[dict[str, Any]] = []
for method in FEATURE_METHODS:
    features = feature_sets.get(method, [])
    key_hits = [f for f in MM_PATHOGNOMONIC if f in features]
    rows.append(
        {
            "method": METHOD_LABELS.get(method, method),
            "n_features": len(features),
            "features": ", ".join(features),
            "mm_key_present": ", ".join(key_hits) if key_hits else "—",
            "mm_key_count": len(key_hits),
        }
    )
return pd.DataFrame(rows)

```

```

def save_clinical_interpretation(
    df: pd.DataFrame,
    path: Path | None = None,
) -> Path:
    """Зберегти таблицю у
    ``results/tables/clinical_interpretation.csv``."""
    Config.ensure_dirs()
    path = path or (Config.RESULTS_TABLES_DIR /
"clinical_interpretation.csv")
    df.to_csv(path, index=False, encoding="utf-8-sig")
    logger.info("Клінічна інтерпретація: %s (%d ознак)", path, len(df))
    return path

```

```
def _print_summary(df: pd.DataFrame, feature_sets: dict[str, list[str]])
-> None:
```

```
    """Вивести підсумок у консоль."""
```

```
    summary = build_method_summary(feature_sets)
```

```
    print("\n" + "=" * 70)
```

```
    print("КЛІНІЧНА ІНТЕРПРЕТАЦІЯ ВІДІБРАНИХ ОЗНАК")
```

```
    print("=" * 70)
```

```
    for _, row in summary.iterrows():
```

```
        print(
```

```
            f"\n{row['method']}: {row['n_features']} ознак | "
```

```
            f"ключові          ММ          ({row['mm_key_count']}/3):  
{row['mm_key_present']}"
```

```
        )
```

```
        print(f" {row['features']}")
```

```
    print("\n--- Патогномонічні маркери ММ ---")
```

```
    for x_id, info in MM_PATHOGNOMONIC.items():
```

```
        subset = df.loc[df["x_id"] == x_id]
```

```
        if subset.empty:
```

```
            methods_str = "не відібрано"
```

```
        else:
```

```
            flags = []
```

```
            if subset.iloc[0]["entropy"]:
```

```
                flags.append("Entropy")
```

```

if subset.iloc[0]["boruta"]:
    flags.append("Boruta")
if subset.iloc[0]["rfe"]:
    flags.append("RFE")
methods_str = ", ".join(flags) if flags else "не відібрано"
print(f" {x_id} — {info['label']}")
print(f" Методи: {methods_str}")
print(f" {info['comment']}")

overlap_all = df[df["n_methods"] == 3]["x_id"].tolist()
print(f"\nПеретин усіх 3 методів ({len(overlap_all)}):
{overlap_all}")
print("=" * 70 + "\n")

def run_clinical_interpretation(
    refresh_features: bool = False,
    save_path: Path | None = None,
) -> pd.DataFrame:
    """Запустити повний пайплайн клінічної інтерпретації."""
    feature_sets = prepare_feature_sets(refresh=refresh_features,
    methods=FEATURE_METHODS)
    df = build_clinical_table(feature_sets)
    save_clinical_interpretation(df, save_path)
    _print_summary(df, feature_sets)
    return df

```

```

def main() -> None:

    parser = argparse.ArgumentParser(description="Клінічна
інтерпретація ознак")

    parser.add_argument(
        "--refresh-features",
        action="store_true",
        help="Перерахувати Boruta/RFE перед інтерпретацією",
    )

    parser.add_argument("--output", type=Path, default=None)

    args = parser.parse_args()

    logging.basicConfig(level=logging.INFO, format="%(levelname)s:
%(message)s")

    run_clinical_interpretation(refresh_features=args.refresh_features,
save_path=args.output)

if __name__ == "__main__":
    main()

```

config.py:

```

"""Конфігурація проєкту скринінгу: шляхи, константи,
відтворюваність."""

```

```

from __future__ import annotations

```

```

import os

```

```

from pathlib import Path

```

```

from dotenv import load_dotenv

# Корінь проєкту — три рівні вище від src/screening/config.py
PROJECT_ROOT = Path(__file__).resolve().parent.parent.parent

load_dotenv(PROJECT_ROOT / ".env")


class Config:
    """Централізовані шляхи та параметри експериментів."""

    # --- Дані ---
    DATA_RAW_DIR: Path = PROJECT_ROOT / "data" / "raw"
    DATA_INTERIM_DIR: Path = PROJECT_ROOT / "data" / "interim"

    BONE_DATASET: str = "Bone_normalized_data.xlsx"
    ENTROPIA_DATASET: str = "DatasetCleanedEntropia.xlsx"
    DECODER_FILE: str = "Расшифровка.xlsx"
    TARGET_COLUMN: str = "Y"

    # --- Результати ---
    RESULTS_DIR: Path = PROJECT_ROOT / "results"
    RESULTS_TABLES_DIR: Path = RESULTS_DIR / "tables"
    RESULTS_FIGURES_DIR: Path = RESULTS_DIR / "figures"
    RESULTS_STATS_DIR: Path = RESULTS_DIR / "stats"
    RESULTS_PREDICTIONS_DIR: Path = RESULTS_DIR / "predictions"

    # --- Відтворюваність (як у R-скриптах керівника) ---
    RANDOM_SEED: int = int(os.getenv("RANDOM_SEED", "123"))
    CV_FOLDS: int = int(os.getenv("CV_FOLDS", "5"))

```

```

INNER_VAL_SIZE: float = float(os.getenv("INNER_VAL_SIZE", "0.15"))
BOOTSTRAP_N: int = int(os.getenv("BOOTSTRAP_N", "1000"))
FIGURE_DPI: int = 200

```

```

# --- Multi-branch: групи ознак для Entropia (10 ознак) ---

```

```

MULTIBRANCH_GROUPS: list[list[str]] = [
    ["x15", "x17", "x18", "x19", "x20"], # еритроцитарні
    ["x5", "x25"], # інші клітинні
    ["x70", "x121", "x158"], # біохімія / маркери
]

```

```

@classmethod

```

```

def ensure_dirs(cls) -> None:

```

```

    """Створити директорії для артефактів, якщо їх ще немає."""

```

```

    for path in (

```

```

        cls.DATA_RAW_DIR,

```

```

        cls.DATA_INTERIM_DIR,

```

```

        cls.RESULTS_TABLES_DIR,

```

```

        cls.RESULTS_FIGURES_DIR,

```

```

        cls.RESULTS_STATS_DIR,

```

```

        cls.RESULTS_PREDICTIONS_DIR,

```

```

    ):

```

```

        path.mkdir(parents=True, exist_ok=True)

```

cross_validation.py:

"""Крос-валідація з внутрішнім validation для early stopping."""

from __future__ import annotations

import logging

from pathlib import Path

from typing import Any, Callable

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

from catboost import CatBoostClassifier

from sklearn.model_selection import StratifiedKFold, train_test_split

from torch.utils.data import DataLoader, TensorDataset

from xgboost import XGBClassifier

from src.screening.config import Config

from src.screening.models import (

MyelomaMultiBranchNet,

SimpleMLP,

StackingModel,

compute_metrics,

compute_scale_pos_weight,

find_best_threshold,

fit_with_early_stopping,

predict_proba_positive,

)

```
logger = logging.getLogger(__name__)
```

```
ModelBuilder = Callable[[float], Any]
```

```
METRIC_COLUMNS: tuple[str, ...] = (
    "accuracy",
    "precision",
    "recall",
    "specificity",
    "f1",
    "auc",
)
```

```
def _inner_train_val_split(
    X: pd.DataFrame,
    y: pd.Series,
    val_size: float,
    random_state: int,
) -> tuple[pd.DataFrame, pd.DataFrame, pd.Series, pd.Series]:
    """Розбити fold-train на train + val для early stopping."""
    return train_test_split(
        X,
        y,
        test_size=val_size,
        stratify=y,
        random_state=random_state,
    )
```

```

def _fit_torch_classifier(
    model: nn.Module,
    X_train: pd.DataFrame,
    y_train: pd.Series,
    X_val: pd.DataFrame,
    y_val: pd.Series,
    random_state: int = Config.RANDOM_SEED,
    max_epochs: int = 150,
    patience: int = 20,
    batch_size: int = 32,
) -> nn.Module:
    """Навчити PyTorch-класифікатор з early stopping за val AUC (Multi-branch.R)."""
    torch.manual_seed(random_state)
    np.random.seed(random_state)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

    n_pos = max(int((y_train == 1).sum()), 1)
    n_neg = int((y_train == 0).sum())
    pos_weight = torch.tensor([n_neg / n_pos], dtype=torch.float32,
device=device)

    criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3,
weight_decay=1e-4)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode="max", factor=0.5, patience=5)

```

```

train_loader = DataLoader(
    TensorDataset(
        torch.tensor(X_train.values, dtype=torch.float32),
        torch.tensor(y_train.values, dtype=torch.float32).unsqueeze(1),
    ),
    batch_size=batch_size,
    shuffle=True,
    drop_last=len(X_train) > batch_size,
)

val_loader = DataLoader(
    TensorDataset(
        torch.tensor(X_val.values, dtype=torch.float32),
        torch.tensor(y_val.values, dtype=torch.float32).unsqueeze(1),
    ),
    batch_size=batch_size,
    shuffle=False,
)

best_auc = -1.0
best_state = {k: v.cpu().clone() for k, v in model.state_dict().items()}
no_improve = 0

from sklearn.metrics import roc_auc_score

for _epoch in range(max_epochs):
    model.train()
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        optimizer.zero_grad()
        loss = criterion(model(xb), yb)

```

```

loss.backward()
optimizer.step()

model.eval()
val_probs, val_true = [], []
with torch.no_grad():
    for xb, yb in val_loader:
        xb = xb.to(device)
        prob = torch.sigmoid(model(xb)).cpu().numpy().ravel()
        val_probs.extend(prob)
        val_true.extend(yb.numpy().ravel())

val_auc = roc_auc_score(val_true, val_probs)
scheduler.step(val_auc)

if val_auc > best_auc:
    best_auc = val_auc
    best_state = {k: v.cpu().clone() for k, v in model.state_dict().items()}
    no_improve = 0
else:
    no_improve += 1
    if no_improve >= patience:
        break

model.load_state_dict(best_state)
model.to(device)
return model

def _fit_model(

```

```

model: Any,
X_train: pd.DataFrame,
y_train: pd.Series,
X_val: pd.DataFrame,
y_val: pd.Series,
random_state: int,
) -> Any:
    """Навчити модель відповідного типу."""
    if isinstance(model, StackingModel):
        return model.fit(X_train, y_train, X_val, y_val)
    if isinstance(model, (MyelomaMultiBranchNet, SimpleMLP)):
        return _fit_torch_classifier(model, X_train, y_train, X_val, y_val,
random_state=random_state)
        return fit_with_early_stopping(model, X_train, y_train, X_val, y_val)

def _evaluate_fold(
    y_test: pd.Series,
    y_val: pd.Series,
    y_proba_test: np.ndarray,
    y_proba_val: np.ndarray,
    threshold_default: float = 0.5,
) -> list[dict[str, Any]]:
    """Метрики на test-fold для порогу 0.5 та оптимального F1 з val."""
    rows: list[dict[str, Any]] = []
    thr_f1 = find_best_threshold(y_val, y_proba_val)

    for thr, label in ((threshold_default, "0.5"), (thr_f1, "f1_opt")):
        pred = (y_proba_test >= thr).astype(int)
        metrics = compute_metrics(y_test, pred, y_proba_test)

```

```

metrics["threshold"] = thr
metrics["threshold_type"] = label
rows.append(metrics)
return rows

def _aggregate_cv_metrics(fold_df: pd.DataFrame) -> pd.DataFrame:
    """Додати рядки mean ± std для кожного threshold_type."""
    summary_rows: list[dict[str, Any]] = []
    for thr_type, group in fold_df.groupby("threshold_type"):
        row: dict[str, Any] = {"fold": f"mean_{thr_type}", "threshold_type":
thr_type}
        for col in METRIC_COLUMNS:
            mean_v = group[col].mean()
            std_v = group[col].std(ddof=1)
            row[col] = mean_v
            row[f"{col}_std"] = std_v
            row["threshold"] = group["threshold"].mean()
        summary_rows.append(row)
    return pd.concat([fold_df, pd.DataFrame(summary_rows)],
ignore_index=True)

def run_cv(
    X: pd.DataFrame,
    y: pd.Series,
    model_builder: ModelBuilder,
    model_name: str = "model",
    n_splits: int = Config.CV_FOLDS,
    random_state: int = Config.RANDOM_SEED,

```

```

inner_val_size: float = Config.INNER_VAL_SIZE,
save_predictions: bool = True,
) -> tuple[pd.DataFrame, np.ndarray, np.ndarray]:
    """Stratified K-fold CV з внутрішнім val для early stopping.

```

Args:

X: Ознаки (Entropy, 10 колонок).

y: Цільова змінна.

model_builder: ``lambda spw: build\_xgboost(spw)`` тощо.

model_name: Ім'я для збереження прогнозів (entropy_xgboost).

n_splits: Кількість фолдів.

random_state: Seed.

inner_val_size: Частка val усередині train-fold.

save_predictions: Зберегти OOF-прогнози у ``results/predictions/``.

Returns:

DataFrame метрик по фолдах + mean/std, OOF y_true, OOF y_proba (f1_opt threshold path).

```

"""

```

```

Config.ensure_dirs()

```

```

assert X.shape[0] == y.shape[0]

```

```

skf = StratifiedKFold(n_splits=n_splits, shuffle=True,
random_state=random_state)

```

```

oof_proba = np.zeros(len(y), dtype=float)

```

```

oof_true = y.values.copy()

```

```

fold_rows: list[dict[str, Any]] = []

```

```

for fold_idx, (train_idx, test_idx) in enumerate(skf.split(X, y), start=1):

```

```

X_train_fold = X.iloc[train_idx]
y_train_fold = y.iloc[train_idx]
X_test_fold = X.iloc[test_idx]
y_test_fold = y.iloc[test_idx]

X_tr, X_val, y_tr, y_val = _inner_train_val_split(
    X_train_fold,
    y_train_fold,
    val_size=inner_val_size,
    random_state=random_state + fold_idx,
)

spw = compute_scale_pos_weight(y_tr)
model = model_builder(spw)
model = _fit_model(model, X_tr, y_tr, X_val, y_val,
    random_state=random_state + fold_idx)

proba_val = predict_proba_positive(model, X_val)
proba_test = predict_proba_positive(model, X_test_fold)
oof_proba[test_idx] = proba_test

for metrics in _evaluate_fold(y_test_fold, y_val, proba_test, proba_val):
    row = {"fold": fold_idx, **metrics}
    fold_rows.append(row)

logger.info(
    "Fold %d/%d | AUC=%.4f | F1@0.5=%.4f",
    fold_idx,
    n_splits,
    fold_rows[-2]["auc"],

```

```

fold_rows[-2]["f1"],
)

fold_df = pd.DataFrame(fold_rows)
# Лиша рядки з f1_opt для підсумкової таблиці по fold (один рядок на
fold)

fold_summary = fold_df[fold_df["threshold_type"] == "f1_opt"].copy()
fold_summary = fold_summary.reset_index(drop=True)
result_df = _aggregate_cv_metrics(fold_df)

if save_predictions:
    pred_path = Config.RESULTS_PREDICTIONS_DIR /
f"{model_name}_oof.npz"
    np.savez(
        pred_path,
        y_true=oof_true,
        y_proba=oof_proba,
        fold_metrics=fold_summary[list(METRIC_COLUMNS)].values,
    )
    logger.info("OOF-прогнози: %s", pred_path)

return result_df, oof_true, oof_proba

def save_oof_predictions(
    path: Path | str,
    y_true: np.ndarray,
    y_proba: np.ndarray,
) -> None:
    """Зберегти out-of-fold прогнози для DeLong."""

```

```
path = Path(path)
path.parent.mkdir(parents=True, exist_ok=True)
np.savez(path, y_true=y_true, y_proba=y_proba)
```

data_loader.py:

"""Завантаження датасетів та таблиці розшифровки ознак."""

from __future__ import annotations

import logging

from pathlib import Path

import pandas as pd

from src.screening.config import Config

logger = logging.getLogger(__name__)

_TARGET_ALIASES: tuple[str, ...] = ("Y", "Clases", "classes", "target")

Очікуваний порядок колонок у Расшифровка.xlsx

_DECODER_COL_RU = 0

_DECODER_COL_PL = 1

_DECODER_COL_XID = 2

_DECODER_COL_XID_ALT = 3

def _find_raw_file(filename: str) -> Path:

"""Знайти файл у data/raw/ або data/."""

for directory in (Config.DATA_RAW_DIR,
Config.DATA_RAW_DIR.parent):

path = directory / filename

if path.exists():

return path

```

        raise FileNotFoundError(f'Не знайдено файл: {filename} у
{Config.DATA_RAW_DIR}')

```

```

def _normalize_target(df: pd.DataFrame) -> pd.DataFrame:

```

```

    """Перейменувати цільову колонку на Config.TARGET_COLUMN."""

```

```

    if Config.TARGET_COLUMN in df.columns:

```

```

        return df

```

```

    for alias in _TARGET_ALIASES:

```

```

        if alias in df.columns:

```

```

            out = df.rename(columns={alias: Config.TARGET_COLUMN})

```

```

            logger.info("Ціль '%s' → '%s'", alias, Config.TARGET_COLUMN)

```

```

        return out

```

```

    raise KeyError(

```

```

        f'Цільова колонка не знайдена. Очікується одна з:
{_TARGET_ALIASES + (Config.TARGET_COLUMN,)}'
    )

```

```

def _split_xy(df: pd.DataFrame) -> tuple[pd.DataFrame, pd.Series]:

```

```

    """Відокремити ознаки від цільової змінної Y."""

```

```

    df = _normalize_target(df)

```

```

    y = df[Config.TARGET_COLUMN].astype(int)

```

```

    X = df.drop(columns=[Config.TARGET_COLUMN])

```

```

    assert X.shape[0] == y.shape[0]

```

```

    return X, y

```

```
def load_entropyia() -> tuple[pd.DataFrame, pd.Series]:
    """Завантажити датасет після ентропійного відбору (10 ознак + Y).

    Returns:
    X: 10 нормалізованих ознак.
    y: бінарна ціль (0 — контроль, 1 — MM).
    """
    try:
        path = _find_raw_file(Config.ENTROPIA_DATASET)
    except FileNotFoundError:
        path = _find_raw_file("Dataset-Cleaned-Entropyia.xlsx")
    df = pd.read_excel(path)
    X, y = _split_xy(df)
    logger.info("Entropyia: %s, ознак=%d, y=%s", path.name, X.shape[1],
y.value_counts().to_dict())
    return X, y
```

```
def load_bone() -> tuple[pd.DataFrame, pd.Series]:
    """Завантажити повний нормалізований датасет Bone (213 ознак + Y).

    Returns:
    X: усі клінічні ознаки.
    y: бінарна ціль (0 — контроль, 1 — MM).
    """
    path = _find_raw_file(Config.BONE_DATASET)
    df = pd.read_excel(path)
    X, y = _split_xy(df)
    logger.info("Bone: %s, ознак=%d, y=%s", path.name, X.shape[1],
y.value_counts().to_dict())
```

return X, y

```
def load_decoder() -> pd.DataFrame:
    """Завантажити розшифровку ознак з польською назвою як основною.

    Returns:
    DataFrame з колонками: x_id, name_pl, name_ru, x_id_alt.
    """
    path = _find_raw_file(Config.DECODER_FILE)
    if not path.exists():
        # резервний пошук: єдиний.xlsx у raw, що не є датасетом
        candidates = [
            p
            for p in Config.DATA_RAW_DIR.glob("*.xlsx")
            if p.name not in (Config.BONE_DATASET,
                               Config.ENTROPIA_DATASET, "Dataset-Cleaned-Entropia.xlsx")
        ]
        if len(candidates) == 1:
            path = candidates[0]
        else:
            raise FileNotFoundError(f"Не знайдено {Config.DECODER_FILE}")

    raw = pd.read_excel(path)
    if raw.shape[1] < 4:
        raise ValueError(f"Очікується  $\geq 4$  колонки у розшифровці, отримано {raw.shape[1]}")

    decoder = pd.DataFrame(
        {
```

```

"name_ru": raw.iloc[:, _DECODER_COL_RU].astype(str).str.strip(),
"name_pl": raw.iloc[:, _DECODER_COL_PL].astype(str).str.strip(),
"x_id": raw.iloc[:, _DECODER_COL_XID].astype(str).str.strip(),
"x_id_alt": raw.iloc[:, _DECODER_COL_XID_ALT].astype(str).str.strip(),
}
)

# Залишити лише рядки з валідним x_id (x1, x15, ...)
decoder = decoder[decoder["x_id"].str.match(r"^\d+",
na=False)].reset_index(drop=True)
decoder = decoder[["x_id", "name_pl", "name_ru", "x_id_alt"]]

logger.info("Розшифровка: %d ознак з %s", len(decoder), path.name)
return decoder

def get_feature_description(x_id: str, decoder: pd.DataFrame) -> str:
    """Повернути польську назву ознаки за її id (наприклад, ``x15``).

    Args:
        x_id: Ідентифікатор ознаки (колонка x_id у розшифровці).
        decoder: Таблиця з ``load_decoder()``.

    Returns:
        Польська назва ознаки (name_pl).

    Raises:
        KeyError: якщо x_id відсутній у розшифровці.
    """
    x_id = str(x_id).strip()

```

```
row = decoder.loc[decoder["x_id"] == x_id]
if row.empty:
    alt = decoder.loc[decoder["x_id_alt"] == x_id]
    if alt.empty:
        raise KeyError(f"Ознаку '{x_id}' не знайдено у розшифровці")
    return str(alt.iloc[0]["name_pl"])
return str(row.iloc[0]["name_pl"])
```

feature_selection.py:

"""Відбір ознак: Entropy (готовий), Boruta, RFE та аналіз перетину."""

from __future__ import annotations

import logging

from pathlib import Path

from typing import Any

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from boruta import BorutaPy

from matplotlib_venn import venn3

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import RFE

from sklearn.linear_model import LogisticRegression

from src.screening.config import Config

from src.screening.data_loader import load_bone, load_decoder

from src.screening.preprocessing import clean_bone_features

logger = logging.getLogger(__name__)

ENTROPY_FEATURES: list[str] = [

"x5",

"x15",

"x17",

"x18",

"x19",

```

"x20",
"x25",
"x70",
"x121",
"x158",
]

```

```
MM_KEY_FEATURES: tuple[str, ...] = ("x15", "x121")
```

```

def select_entropy_features(save: bool = True) -> list[str]:
    """Повернути 10 ознак з ентропійного відбору (готовий результат
керівника).

```

Args:

save: Зберегти список у ``results/tables/entropy\_features.csv``.

Returns:

Список ідентифікаторів ознак.

```

"""
features = list(ENTROPY_FEATURES)
if save:
    Config.ensure_dirs()
    path = Config.RESULTS_TABLES_DIR / "entropy_features.csv"
    pd.DataFrame({"x_id": features}).to_csv(path, index=False, encoding="utf-
8-sig")
    logger.info("Entropy: збережено %s (%d ознак)", path, len(features))
    return features

```

```
def select_boruta_features(
    X: pd.DataFrame,
    y: pd.Series,
    random_state: int = Config.RANDOM_SEED,
    save: bool = True,
) -> dict[str, Any]:
    """Відбір ознак методом Boruta на RandomForest.
```

Args:

X: Матриця ознак (Bone, після очищення).

y: Цільова змінна.

random_state: Seed для відтворюваності.

save: Зберегти confirmed у CSV.

Returns:

Словник з ключами confirmed, tentative, rejected, ranking.

"""

```
    rf = RandomForestClassifier(
        n_estimators=100,
        n_jobs=-1,
        class_weight="balanced",
        max_depth=5,
        random_state=random_state,
    )
    boruta = BorutaPy(
        estimator=rf,
        n_estimators="auto",
        max_iter=100,
        random_state=random_state,
        verbose=0,
```

```

    )
    logger.info("Boruta: старт на %d ознаках, %d спостереженнях...",
X.shape[1], len(y))
    boruta.fit(X.values, y.values)

    feature_names = list(X.columns)
    support = boruta.support_
    ranking = boruta.ranking_

    confirmed = [feature_names[i] for i, s in enumerate(support) if s == 1]
    tentative = [feature_names[i] for i, s in enumerate(support) if s == -1]
    rejected = [feature_names[i] for i, s in enumerate(support) if s == 0]

    ranking_dict = {feature_names[i]: int(ranking[i]) for i in
range(len(feature_names))}

    if len(confirmed) == 0:
        logger.warning(
            "Boruta: 0 confirmed ознак — використовуємо tentative (%d) як fallback",
            len(tentative),
        )
        confirmed = tentative.copy()

    result: dict[str, Any] = {
        "confirmed": confirmed,
        "tentative": tentative,
        "rejected": rejected,
        "ranking": ranking_dict,
    }

```

```

logger.info(
    "Boruta: confirmed=%d, tentative=%d, rejected=%d",
    len(confirmed),
    len(tentative),
    len(rejected),
)

if save:
    Config.ensure_dirs()
    rows = [{"x_id": fid, "rank": ranking_dict[fid]} for fid in confirmed]
    path = Config.RESULTS_TABLES_DIR / "boruta_features.csv"
    pd.DataFrame(rows).to_csv(path, index=False, encoding="utf-8-sig")
    logger.info("Boruta: збережено %s", path)

```

```

return result

```

```

def select_rfe_features(
    X: pd.DataFrame,
    y: pd.Series,
    n_features: int = 10,
    random_state: int = Config.RANDOM_SEED,
    save: bool = True,
) -> list[str]:
    """Відбір топ-N ознак рекурсивним RFE з LogisticRegression.

```

Args:

X: Матриця ознак.

y: Цільова змінна.

n_features: Кількість ознак для відбору.

random_state: Seed.

save: Зберегти у CSV.

Returns:

Список відібраних x_id.

"""

```
estimator = LogisticRegression(
```

```
max_iter=1000,
```

```
class_weight="balanced",
```

```
random_state=random_state,
```

```
solver="liblinear",
```

```
)
```

```
rfe = RFE(estimator=estimator, n_features_to_select=n_features, step=1)
```

```
rfe.fit(X, y)
```

```
selected_mask = rfe.support_
```

```
ranking = rfe.ranking_
```

```
feature_names = list(X.columns)
```

```
selected = [feature_names[i] for i, supported in enumerate(selected_mask) if
supported]
```

```
rows = sorted(
```

```
[(feature_names[i], int(ranking[i])) for i in range(len(feature_names)) if
selected_mask[i]],
```

```
key=lambda t: t[1],
```

```
)
```

```
if save:
```

```
Config.ensure_dirs()
```

```
path = Config.RESULTS_TABLES_DIR / "rfe_features.csv"
```

```
pd.DataFrame([{"x_id": fid, "rank": rank} for fid, rank in rows]).to_csv(
    path, index=False, encoding="utf-8-sig"
)
logger.info("RFE: збережено %s (%d ознак)", path, len(selected))

return selected
```

```
def compute_feature_overlap(
    entropy: list[str],
    boruta: list[str],
    rfe: list[str],
    save: bool = True,
) -> dict[str, list[str]]:
    """Обчислити перетини трьох наборів відібраних ознак.
```

Args:

entropy: Ознаки Entropy.

boruta: Confirmed (або fallback) Boruta.

rfe: Ознаки RFE.

save: Зберегти у CSV.

Returns:

Словник множин перетину та унікальних ознак.

```
"""
```

```
set_e = set(entropy)
```

```
set_b = set(boruta)
```

```
set_r = set(rfe)
```

```
overlap: dict[str, list[str]] = {
```

```

    "all_three": sorted(set_e & set_b & set_r),
    "entropy_boruta": sorted(set_e & set_b),
    "entropy_rfe": sorted(set_e & set_r),
    "boruta_rfe": sorted(set_b & set_r),
    "only_entropy": sorted(set_e - set_b - set_r),
    "only_boruta": sorted(set_b - set_e - set_r),
    "only_rfe": sorted(set_r - set_e - set_b),
}

```

```

if save:

```

```

    Config.ensure_dirs()

```

```

    path = Config.RESULTS_TABLES_DIR / "feature_overlap.csv"

```

```

    rows = [

```

```

        {

```

```

            "Множина": key,

```

```

            "Кількість": len(vals),

```

```

            "Ознаки": ", ".join(vals) if vals else "",

```

```

        }

```

```

    for key, vals in overlap.items()

```

```

]

```

```

pd.DataFrame(rows).to_csv(path, index=False, encoding="utf-8-sig")

```

```

logger.info("Перетин: збережено %s", path)

```

```

return overlap

```

```

def plot_feature_overlap_venn(

```

```

    entropy: list[str],

```

```

    boruta: list[str],

```

```

    rfe: list[str],

```

```

save_path: Path | None = None,
) -> None:
    """Побудувати Venn-діаграму перетину трьох методів відбору.

    Args:
        entropy: Ознаки Entropy.
        boruta: Ознаки Boruta (confirmed / fallback).
        rfe: Ознаки RFE.
        save_path: Шлях до PNG (за замовчуванням
results/figures/feature_overlap_venn.png).
    """

    save_path = save_path or (Config.RESULTS_FIGURES_DIR /
"feature_overlap_venn.png")
    Config.ensure_dirs()

    set_e, set_b, set_r = set(entropy), set(boruta), set(rfe)
    common = sorted(set_e & set_b & set_r)

    fig, ax = plt.subplots(figsize=(10, 8))
    venn3(
        [set_e, set_b, set_r],
        set_labels=(
            f"Entropy (n={len(set_e)})",
            f"Boruta (n={len(set_b)})",
            f"RFE (n={len(set_r)})",
        ),
        ax=ax,
    )
    ax.set_title("Перетин відібраних ознак між методами", fontsize=14)

```

if common:

```
ax.text(
    0.5,
    -0.12,
    "Спільні (усі 3): " + ", ".join(common),
    transform=ax.transAxes,
    ha="center",
    fontsize=10,
    wrap=True,
)
```

```
plt.tight_layout()
plt.savefig(save_path, dpi=Config.FIGURE_DPI, bbox_inches="tight")
plt.close()
logger.info("Venn: збережено %s", save_path)
```

```
def _print_summary(
    entropy: list[str],
    boruta_result: dict[str, Any],
    rfe: list[str],
    overlap: dict[str, list[str]],
) -> None:
    """Вивести підсумок відбору ознак у консоль."""
    boruta_used = boruta_result["confirmed"]
    decoder = load_decoder()

    print("\n" + "=" * 60)
    print("ПІДСУМОК ВІДБОРУ ОЗНАК")
    print("=" * 60)
```

```

print(f"Entropy: {len(entropy)} ознак")
print(f"Boruta (confirmed/fallback): {len(boruta_used)} ознак")
print(f"    —    tentative:    {len(boruta_result['tentative'])},    rejected:
{len(boruta_result['rejected'])}")

print(f"RFE: {len(rfe)} ознак")
print(f"\nПеретин усіх 3 методів ({len(overlap['all_three'])}):
{overlap['all_three']}")

print(f"Тільки Entropy: {overlap['only_entropy']}")
print(f"Тільки Boruta: {overlap['only_boruta']}")
print(f"Тільки RFE: {overlap['only_rfe']}")


print("\nКлючові ознаки для MM:")
all_selected = set(entropy) | set(boruta_used) | set(rfe)
for feat in MM_KEY_FEATURES:
    try:
        name_pl = decoder.loc[decoder["x_id"] == feat, "name_pl"].iloc[0]
    except (IndexError, KeyError):
        name_pl = "?"
    status = "ТАК" if feat in all_selected else "НІ"
    print(f" {feat} ({name_pl}): у щонайменше 1 методі — {status}")


print("=" * 60 + "\n")


def run_all_feature_selection() -> dict[str, Any]:
    """Запустити повний пайплайн відбору ознак і зберегти артефакти.

    Returns:
        Словник з результатами всіх методів та перетинами.
    """

```

```

logging.basicConfig(level=logging.INFO,          format="%(levelname)s:
%(message)s")
Config.ensure_dirs()

X_bone, y_bone = load_bone()
X_bone, _ = clean_bone_features(X_bone)

entropy = select_entropy_features(save=True)
boruta_result = select_boruta_features(X_bone, y_bone, save=True)
boruta_used = boruta_result["confirmed"]
rfe = select_rfe_features(X_bone, y_bone, n_features=10, save=True)

overlap = compute_feature_overlap(entropy, boruta_used, rfe, save=True)
plot_feature_overlap_venn(entropy, boruta_used, rfe)

_print_summary(entropy, boruta_result, rfe, overlap)

return {
    "entropy": entropy,
    "boruta": boruta_result,
    "rfe": rfe,
    "overlap": overlap,
}

if __name__ == "__main__":
    run_all_feature_selection()

```

models.py:

```
"""Побудова моделей класифікації за R-скриптами керівника."""
```

```
from __future__ import annotations
```

```
import logging
```

```
from dataclasses import dataclass
```

```
from typing import Any, Callable
```

```
import numpy as np
```

```
import pandas as pd
```

```
import torch
```

```
import torch.nn as nn
```

```
from catboost import CatBoostClassifier
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import (
```

```
    accuracy_score,
```

```
    f1_score,
```

```
    precision_score,
```

```
    recall_score,
```

```
    roc_auc_score,
```

```
)
```

```
from xgboost import XGBClassifier
```

```
from src.screening.config import Config
```

```
logger = logging.getLogger(__name__)
```

```
ModelBuilder = Callable[[float], Any]
```

```

def compute_scale_pos_weight(y: pd.Series | np.ndarray) -> float:
    """Обчислити scale_pos_weight = n(0) / n(1) для дисбалансу класів."""
    y_arr = np.asarray(y)
    n_pos = max(int((y_arr == 1).sum()), 1)
    n_neg = int((y_arr == 0).sum())
    return float(n_neg / n_pos)

def find_best_threshold(
    y_true: pd.Series | np.ndarray,
    y_proba: np.ndarray,
) -> float:
    """Підібрати поріг класифікації за максимумом F1 (як у R, крок 0.01)."""
    best_thr = 0.5
    best_f1 = -1.0
    for thr in np.arange(0.05, 0.96, 0.01):
        pred = (y_proba >= thr).astype(int)
        f1 = f1_score(y_true, pred, zero_division=0)
        if f1 > best_f1:
            best_f1 = f1
            best_thr = float(thr)
    return best_thr

def compute_metrics(
    y_true: pd.Series | np.ndarray,
    y_pred: np.ndarray,
    y_proba: np.ndarray,
) -> dict[str, float]:

```

"""Шість метрик бінарної класифікації + confusion matrix.

Args:

y_true: Істинні мітки.

y_pred: Передбачені класи (0/1).

y_proba: Ймовірності класу 1.

Returns:

accuracy, precision, recall, specificity, f1, auc.

"""

```

y_true = np.asarray(y_true)
tn = int(np.sum((y_pred == 0) & (y_true == 0)))
fp = int(np.sum((y_pred == 1) & (y_true == 0)))
fn = int(np.sum((y_pred == 0) & (y_true == 1)))
tp = int(np.sum((y_pred == 1) & (y_true == 1)))
specificity = tn / (tn + fp) if (tn + fp) > 0 else 0.0

return {
    "accuracy": float(accuracy_score(y_true, y_pred)),
    "precision": float(precision_score(y_true, y_pred, zero_division=0)),
    "recall": float(recall_score(y_true, y_pred, zero_division=0)),
    "specificity": float(specificity),
    "f1": float(f1_score(y_true, y_pred, zero_division=0)),
    "auc": float(roc_auc_score(y_true, y_proba)),
}
```

def build_xgboost(scale_pos_weight: float) -> XGBClassifier:

"""XGBoost з гіперпараметрами з XGBOOST_CATBOOST.R."""

return XGBClassifier(

```

objective="binary:logistic",
eval_metric="auc",
max_depth=4,
learning_rate=0.03,
min_child_weight=3,
subsample=0.85,
colsample_bytree=0.75,
gamma=0.0,
reg_lambda=1.0,
reg_alpha=0.0,
scale_pos_weight=scale_pos_weight,
n_estimators=1000,
early_stopping_rounds=40,
random_state=Config.RANDOM_SEED,
n_jobs=-1,
)

```

```

def build_catboost(scale_pos_weight: float) -> CatBoostClassifier:
    """CatBoost з гіперпараметрами з XGBOOST_CATBOOST.R."""
    return CatBoostClassifier(
        loss_function="Logloss",
        eval_metric="AUC",
        iterations=1500,
        learning_rate=0.03,
        depth=5,
        l2_leaf_reg=5,
        random_seed=Config.RANDOM_SEED,
        od_type="Iter",
        od_wait=50,
    )

```

```

class_weights=[1.0, scale_pos_weight],
verbose=False,
)

```

```

class BranchEncoder(nn.Module):
    """MLP-енкодер однієї гілки (Multi-branch.R)."""

    def __init__(
        self,
        in_features: int,
        hidden1: int = 64,
        hidden2: int = 32,
        dropout: float = 0.2,
    ):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(in_features, hidden1),
            nn.BatchNorm1d(hidden1),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),
            nn.BatchNorm1d(hidden2),
            nn.ReLU(),
            nn.Dropout(dropout),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.net(x)

```

```

class GatedFusion(nn.Module):
    """Зважене об'єднання гілок через softmax-гейт."""

    def __init__(self, n_branches: int, emb_dim: int):
        super().__init__()
        self.gate = nn.Linear(n_branches * emb_dim, n_branches)

    def forward(self, branch_outputs: list[torch.Tensor]) -> torch.Tensor:
        z = torch.cat(branch_outputs, dim=1)
        weights = torch.softmax(self.gate(z), dim=1)
        weighted = [branch_outputs[i] * weights[:, i : i + 1] for i in
range(len(branch_outputs))]
        return torch.cat(weighted, dim=1)

class MyelomaMultiBranchNet(nn.Module):
    """Multi-branch мережа за Multi-branch.R."""

    def __init__(
        self,
        feature_group_indices: list[list[int]],
        branch_h1: int = 64,
        branch_h2: int = 32,
        head_hidden: int = 64,
        dropout: float = 0.2,
    ):
        super().__init__()
        self.feature_group_indices = feature_group_indices
        self.branches = nn.ModuleList(

```

```

[
    BranchEncoder(len(indices), branch_h1, branch_h2, dropout)
    for indices in feature_group_indices
]
)

self.fusion = GatedFusion(len(feature_group_indices), branch_h2)
fused_dim = len(feature_group_indices) * branch_h2
self.head = nn.Sequential(
    nn.Linear(fused_dim, head_hidden),
    nn.BatchNorm1d(head_hidden),
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.Linear(head_hidden, 32),
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.Linear(32, 1),
)

def forward(self, x: torch.Tensor) -> torch.Tensor:
    branch_outputs = [self.branches[i](x[:, idx]) for i, idx in
enumerate(self.feature_group_indices)]
    fused = self.fusion(branch_outputs)
    return self.head(fused)

class SimpleMLP(nn.Module):
    """Простий MLP: вхід  $\rightarrow 32 \rightarrow 16 \rightarrow 1$ ."""

    def __init__(self, in_features: int, dropout: float = 0.2):
        super().__init__()

```

```

self.net = nn.Sequential(
    nn.Linear(in_features, 32),
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.Linear(32, 16),
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.Linear(16, 1),
)

```

```

def forward(self, x: torch.Tensor) -> torch.Tensor:
    return self.net(x)

```

```

def filter_multibranch_groups(
    feature_groups: list[list[str]],
    available: list[str] | set[str],
) -> list[list[str]]:
    """Залишити в групах лише ознаки, присутні у поточному наборі
    (Boruta/RFE)."""

```

```

    avail = set(available)
    filtered = [[feat for feat in group if feat in avail] for group in feature_groups]
    return [group for group in filtered if group]

```

```

def _resolve_group_indices(
    feature_groups: list[list[str]],
    column_names: list[str],
) -> list[list[int]]:
    """Перетворити імена ознак у індекси колонок."""

```

```

name_to_idx = {name: i for i, name in enumerate(column_names)}
indices: list[list[int]] = []
for group in feature_groups:
    indices.append([name_to_idx[name] for name in group])
return indices

```

```

def build_multibranch(
    feature_groups: list[list[str]] | None = None,
    column_names: list[str] | None = None,
) -> MyelomaMultiBranchNet:
    """Створити multi-branch мережу з групами ознак Entropia.

```

Args:

feature_groups: Список груп імен ознак (за замовчуванням з Config).

column_names: Порядок колонок X (усі 10 ознак Entropia).

Returns:

Неініціалізована PyTorch-модель.

```

"""

```

```

groups = feature_groups or Config.MULTIBRANCH_GROUPS
if column_names is None:
    column_names = [f for g in groups for f in g]
    # унікальний порядок як у Entropia
seen: set[str] = set()
ordered: list[str] = []
for g in groups:
    for f in g:
        if f not in seen:
            seen.add(f)

```

```
ordered.append(f)
```

```
column_names = ordered
```

```
group_indices = _resolve_group_indices(groups, column_names)
```

```
return MyelomaMultiBranchNet(group_indices)
```

```
def build_mlp(in_features: int) -> SimpleMLP:
```

```
    """Простий MLP на PyTorch (2 приховані шари 32→16)."""
```

```
    return SimpleMLP(in_features=in_features)
```

```
@dataclass
```

```
class StackingModel:
```

```
    """Стекінг: XGBoost + CatBoost → LogisticRegression (як STACKING-1.R)."""
```

```
    scale_pos_weight: float
```

```
    xgb: XGBClassifier | None = None
```

```
    catboost: CatBoostClassifier | None = None
```

```
    meta: LogisticRegression | None = None
```

```
    random_state: int = Config.RANDOM_SEED
```

```
    def fit(
```

```
        self,
```

```
        X_train: pd.DataFrame,
```

```
        y_train: pd.Series,
```

```
        X_val: pd.DataFrame,
```

```
        y_val: pd.Series,
```

```
    ) -> StackingModel:
```

```

"""Навчити базові моделі на train, мета-модель на val-прогнозах."""
self.xgb = build_xgboost(self.scale_pos_weight)
self.catboost = build_catboost(self.scale_pos_weight)

self.xgb.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=False)
self.catboost.fit(X_train, y_train, eval_set=(X_val, y_val), verbose=False)

meta_X = np.column_stack(
    [
        self.xgb.predict_proba(X_val)[:, 1],
        self.catboost.predict_proba(X_val)[:, 1],
    ]
)

self.meta = LogisticRegression(max_iter=2000,
random_state=self.random_state)
self.meta.fit(meta_X, y_val)
return self

def predict_proba(self, X: pd.DataFrame) -> np.ndarray:
    """Ймовірності класу 1 на тестовій вибірці."""
    if self.xgb is None or self.catboost is None or self.meta is None:
        raise RuntimeError("StackingModel не навчена — викличте fit()
спочатку")
    xgb_p = self.xgb.predict_proba(X)[:, 1]
    cat_p = self.catboost.predict_proba(X)[:, 1]
    meta_X = np.column_stack([xgb_p, cat_p])
    return self.meta.predict_proba(meta_X)[:, 1]

def build_stacking(scale_pos_weight: float) -> StackingModel:

```

```

"""Фабрика стекінгу XGB + CatBoost → LogisticRegression."""
return StackingModel(scale_pos_weight=scale_pos_weight)

# Реєстр імен моделей для run_experiments (Крок 7)
MODEL_NAMES: tuple[str, ...] = ("xgboost", "catboost", "stacking",
"multibranch", "mlp")

def get_model_builder(
    name: str,
    scale_pos_weight: float,
    in_features: int = 10,
    column_names: list[str] | None = None,
) -> Any:
    """Повернути конструктор / об'єкт моделі за іменем."""
    if name == "xgboost":
        return build_xgboost(scale_pos_weight)
    if name == "catboost":
        return build_catboost(scale_pos_weight)
    if name == "stacking":
        return build_stacking(scale_pos_weight)
    if name == "multibranch":
        return build_multibranch(column_names=column_names)
    if name == "mlp":
        return build_mlp(in_features)
    raise ValueError(f"Невідома модель: {name}. Доступні:
{MODEL_NAMES}")

```

```

def fit_with_early_stopping(
    model: Any,
    X_train: pd.DataFrame,
    y_train: pd.Series,
    X_val: pd.DataFrame,
    y_val: pd.Series,
) -> Any:
    """Універсальне навчання з early stopping для XGBoost / CatBoost."""
    if isinstance(model, XGBClassifier):
        model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=False)
    elif isinstance(model, CatBoostClassifier):
        model.fit(X_train, y_train, eval_set=(X_val, y_val), verbose=False)
    else:
        model.fit(X_train, y_train)
    return model


def predict_proba_positive(model: Any, X: pd.DataFrame) -> np.ndarray:
    """Отримати ймовірність позитивного класу."""
    if hasattr(model, "predict_proba"):
        proba = np.asarray(model.predict_proba(X))
        if proba.ndim == 2:
            return proba[:, 1]
        return proba.ravel()
    if isinstance(model, (MyelomaMultiBranchNet, SimpleMLP)):
        model.eval()
        with torch.no_grad():
            x_t = torch.tensor(X.values, dtype=torch.float32)
            logits = model(x_t)
            return torch.sigmoid(logits).numpy().ravel()

```

```
raise TypeError(f"Невідомий тип моделі: {type(model)}")
```

plots.py:

```
"""Візуалізації результатів 15 експериментів (ROC, barplot, CM, importance,
boxplot)."""
```

```
from __future__ import annotations
```

```
import argparse
```

```
import logging
```

```
from pathlib import Path
```

```
import matplotlib
```

```
matplotlib.use("Agg")
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
from sklearn.metrics import RocCurveDisplay, confusion_matrix
```

```
from src.screening.config import Config
```

```
from src.screening.cross_validation import METRIC_COLUMNS
```

```
from src.screening.data_loader import load_decoder
```

```
from src.screening.models import build_xgboost, compute_scale_pos_weight
```

```
from src.screening.run_experiments import FEATURE_METHODS,
get_dataset_for_method, prepare_feature_sets
```

```
from src.screening.statistics import load_experiment_predictions
```

```
logger = logging.getLogger(__name__)
```

```
METHOD_LABELS: dict[str, str] = {
```

```

"entropy": "Entropy",
"boruta": "Boruta",
"rfe": "RFE",
}

```

```

MODEL_LABELS: dict[str, str] = {
    "xgboost": "XGBoost",
    "catboost": "CatBoost",
    "stacking": "Stacking",
    "multibranch": "Multi-branch",
    "mlp": "MLP",
}

```

```

BAR_METRICS: tuple[str, ...] = ("oof_auc", "f1", "recall")

```

```

BOX_METRICS: tuple[str, ...] = ("auc", "f1", "recall")

```

```

def load_experiment_results(path: Path | None = None) -> pd.DataFrame:
    """Завантажити ``experiment_results.csv``."""
    path = path or (Config.RESULTS_TABLES_DIR / "experiment_results.csv")
    if not path.exists():
        raise FileNotFoundError(f"Не знайдено {path} — спочатку запустіть
run_experiments")
    return pd.read_csv(path)

```

```

def _save_fig(fig: plt.Figure, save_path: Path) -> None:
    Config.ensure_dirs()
    fig.tight_layout()
    fig.savefig(save_path, dpi=Config.FIGURE_DPI, bbox_inches="tight")

```

```

plt.close(fig)
logger.info("Графік: %s", save_path)

def plot_roc_by_method(
    method: str,
    results: pd.DataFrame,
    predictions: dict[str, dict[str, np.ndarray]],
    save_path: Path | None = None,
) -> None:
    """ROC-криві для 5 моделей одного методу відбору ознак."""
    save_path = save_path or (Config.RESULTS_FIGURES_DIR /
f'roc_{method}.png')
    subset = results[results["feature_method"] == method].sort_values("model")

    fig, ax = plt.subplots(figsize=(8, 7))
    for _, row in subset.iterrows():
        exp_id = row["experiment_id"]
        if exp_id not in predictions:
            continue
        y_true = predictions[exp_id]["y_true"]
        y_proba = predictions[exp_id]["y_proba"]
        label = f'{MODEL_LABELS.get(row["model"], row["model"])}
(AUC={row["oof_auc"]:.3f})'
        RocCurveDisplay.from_predictions(y_true, y_proba, ax=ax, name=label)

    ax.plot([0, 1], [0, 1], "k--", lw=1, alpha=0.5)
    ax.set_title(f'ROC — {METHOD_LABELS.get(method, method)}')
    ax.set_xlabel("1 - Specificity")
    ax.set_ylabel("Sensitivity (Recall)")

```

```

ax.legend(loc="lower right", fontsize=8)
_save_fig(fig, save_path)

def plot_roc_all(
    results: pd.DataFrame,
    predictions: dict[str, dict[str, np.ndarray]],
    save_path: Path | None = None,
) -> None:
    """Загальний ROC: найкраща модель кожного методу + решта
    напівпрозора."""
    save_path = save_path or (Config.RESULTS_FIGURES_DIR /
    "roc_all.png")
    fig, ax = plt.subplots(figsize=(9, 8))

    best_ids = (
        results.sort_values("oof_auc", ascending=False)
        .groupby("feature_method", as_index=False)
        .first()["experiment_id"]
        .tolist()
    )

    for _, row in results.sort_values("oof_auc", ascending=False).iterrows():
        exp_id = row["experiment_id"]
        if exp_id not in predictions:
            continue
        y_true = predictions[exp_id]["y_true"]
        y_proba = predictions[exp_id]["y_proba"]
        is_best = exp_id in best_ids
        label = (

```

```

f"{exp_id} ({row['oof_auc']:.3f})"
if is_best
else None
)
curve_kwargs = (
{"alpha": 0.85, "linewidth": 2.0}
if is_best
else {"alpha": 0.25, "linewidth": 1.0}
)
display = RocCurveDisplay.from_predictions(
y_true,
y_proba,
ax=ax,
name=label or "_other",
curve_kwargs=curve_kwargs,
)
if not is_best:
display.line_.set_label("_nolegend_")

ax.plot([0, 1], [0, 1], "k--", lw=1, alpha=0.5)
ax.set_title("ROC — усі експерименти (найкращі по методу в легенді)")
ax.set_xlabel("1 – Specificity")
ax.set_ylabel("Sensitivity (Recall)")
ax.legend(loc="lower right", fontsize=7)
_save_fig(fig, save_path)

def plot_metrics_barplot(
results: pd.DataFrame,
save_path: Path | None = None,

```

```

) -> None:

"""Barplot OOF AUC, F1 та Recall для 15 експериментів."""
save_path = save_path or (Config.RESULTS_FIGURES_DIR /
"barplot_metrics.png")
df = results.copy()
df["label"] = df["feature_method"] + "\n" + df["model"]
df = df.sort_values(["feature_method", "oof_auc"], ascending=[True, False])

fig, axes = plt.subplots(1, 3, figsize=(16, 6), sharey=False)
titles = {"oof_auc": "OOF AUC", "f1": "F1 (f1_opt)", "recall": "Recall
(f1_opt)"}
colors = {"entropy": "#4C72B0", "boruta": "#55A868", "rfe": "#C44E52"}

for ax, metric in zip(axes, BAR_METRICS, strict=True):
    bars = ax.bar(
        df["label"],
        df[metric],
        color=[colors[m] for m in df["feature_method"]],
        edgecolor="white",
        linewidth=0.5,
    )
    ax.set_title(titles[metric])
    ax.set_ylim(0, 1.05)
    ax.tick_params(axis="x", rotation=90, labelsz=7)
    ax.grid(axis="y", alpha=0.3)
    for bar, val in zip(bars, df[metric], strict=True):
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            bar.get_height() + 0.01,
            f"{val:.2f}",

```

```

    ha="center",
    va="bottom",
    fontsize=6,
)

fig.suptitle("Метрики 15 експериментів", fontsize=13, y=1.02)
_save_fig(fig, save_path)

def plot_confusion_matrix_best(
    results: pd.DataFrame,
    predictions: dict[str, dict[str, np.ndarray]],
    save_path: Path | None = None,
) -> str:
    """Confusion matrix для експерименту з найвищим OOF AUC."""
    save_path = save_path or (Config.RESULTS_FIGURES_DIR /
"confusion_matrix_best.png")
    best = results.loc[results["oof_auc"].idxmax()]
    exp_id = best["experiment_id"]
    y_true = predictions[exp_id]["y_true"]
    y_proba = predictions[exp_id]["y_proba"]
    threshold = float(best["threshold"])
    y_pred = (y_proba >= threshold).astype(int)

    cm = confusion_matrix(y_true, y_pred, labels=[0, 1])
    fig, ax = plt.subplots(figsize=(6, 5))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",

```

```

cmap="Blues",
xticklabels=["Control (0)", "MM (1)"],
yticklabels=["Control (0)", "MM (1)"],
ax=ax,
cbar=False,
)
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
ax.set_title(
f'Confusion matrix — {exp_id}\n'
f'OOF AUC={best['oof_auc']:.3f}, threshold={threshold:.2f}'
)
_save_fig(fig, save_path)
return exp_id

def _train_xgb_importance(
    method: str,
    feature_sets: dict[str, list[str]],
    top_n: int = 10,
) -> pd.DataFrame:
    """Навчити XGBoost на повній вибірці та повернути топ-N
    importance."""
    X, y, features = get_dataset_for_method(method, feature_sets)
    model = build_xgboost(compute_scale_pos_weight(y))
    model.set_params(n_estimators=300, early_stopping_rounds=None)
    model.fit(X, y, verbose=False)

    importances = model.feature_importances_
    df = pd.DataFrame({"x_id": features, "importance": importances})

```

```

return df.sort_values("importance", ascending=False).head(top_n)

def plot_feature_importance(
    method: str,
    feature_sets: dict[str, list[str]],
    save_path: Path | None = None,
    top_n: int = 10,
) -> None:
    """Топ-N feature importance (XGBoost) для методу відбору."""
    save_path = save_path or (Config.RESULTS_FIGURES_DIR /
f"importance_{method}.png")
    imp_df = _train_xgb_importance(method, feature_sets, top_n=top_n)

    try:
        decoder = load_decoder()
        imp_df = imp_df.copy()
        imp_df["label"] = imp_df["x_id"].apply(
            lambda x: f"{x}\n{decoder.loc[decoder['x_id'] == x, 'name_pl'].iloc[0]}"
            if x in decoder["x_id"].values
            else x
        )
    except (FileNotFoundError, IndexError, KeyError):
        imp_df["label"] = imp_df["x_id"]

    fig, ax = plt.subplots(figsize=(9, 6))
    imp_sorted = imp_df.sort_values("importance", ascending=True)
    ax.barh(imp_sorted["label"], imp_sorted["importance"], color="#4C72B0")
    ax.set_xlabel("XGBoost feature importance (gain)")

```

```

ax.set_title(f"Топ-{top_n} ознак — {METHOD_LABELS.get(method,
method)}")
_save_fig(fig, save_path)

```

```

def plot_fold_metrics_boxplot(
    results: pd.DataFrame,
    predictions: dict[str, dict[str, np.ndarray]],
    save_path: Path | None = None,
) -> None:
    """Boxplot метрик по фолдах CV для всіх експериментів."""
    save_path = save_path or (Config.RESULTS_FIGURES_DIR /
"boxplot_fold_metrics.png")
    rows: list[dict[str, object]] = []

    for _, exp_row in results.iterrows():
        exp_id = exp_row["experiment_id"]
        if exp_id not in predictions:
            continue
        fold_metrics = predictions[exp_id]["fold_metrics"]
        for fold_idx in range(fold_metrics.shape[0]):
            for metric in BOX_METRICS:
                col_idx = list(METRIC_COLUMNS).index(metric)
                rows.append(
                    {
                        "experiment_id": exp_id,
                        "feature_method": exp_row["feature_method"],
                        "model": exp_row["model"],
                        "fold": fold_idx + 1,
                        "metric": metric.upper(),

```

```

"value": float(fold_metrics[fold_idx, col_idx]),
}
)

df = pd.DataFrame(rows)
fig, ax = plt.subplots(figsize=(16, 6))
order = (
    results.sort_values(["feature_method",    "oof_auc"],    ascending=[True,
False])[ "experiment_id"]
    .tolist()
)
sns.boxplot(
    data=df,
    x="experiment_id",
    y="value",
    hue="metric",
    order=order,
    ax=ax,
    palette="Set2",
)
ax.set_title("Розподіл метрик по фолдах CV (fl_opt)")
ax.set_xlabel("")
ax.set_ylabel("Значення")
ax.tick_params(axis="x", rotation=90, labelsz=7)
ax.legend(title="Метрика", loc="upper right")
ax.grid(axis="y", alpha=0.3)
_save_fig(fig, save_path)

def run_all_plots(

```

```

results_path: Path | None = None,
predictions_dir: Path | None = None,
) -> dict[str, str]:
    """Побудувати всі графіки кроку 9."""
    Config.ensure_dirs()
    results = load_experiment_results(results_path)
    predictions = load_experiment_predictions(predictions_dir)
    feature_sets = prepare_feature_sets(methods=FEATURE_METHODS)

    artifacts: dict[str, str] = {}

    for method in FEATURE_METHODS:
        path = Config.RESULTS_FIGURES_DIR / f'roc_{method}.png'
        plot_roc_by_method(method, results, predictions, path)
        artifacts[f'roc_{method}'] = str(path)

    all_roc = Config.RESULTS_FIGURES_DIR / "roc_all.png"
    plot_roc_all(results, predictions, all_roc)
    artifacts["roc_all"] = str(all_roc)

    bar_path = Config.RESULTS_FIGURES_DIR / "barplot_metrics.png"
    plot_metrics_barplot(results, bar_path)
    artifacts["barplot_metrics"] = str(bar_path)

    best_id = plot_confusion_matrix_best(results, predictions)
    artifacts["confusion_matrix_best"] = str(Config.RESULTS_FIGURES_DIR
/ "confusion_matrix_best.png")
    artifacts["best_experiment"] = best_id

    for method in FEATURE_METHODS:

```

```

        imp_path = Config.RESULTS_FIGURES_DIR /
f"importance_{method}.png"
        plot_feature_importance(method, feature_sets, imp_path)
        artifacts[f"importance_{method}"] = str(imp_path)

        box_path = Config.RESULTS_FIGURES_DIR /
"boxplot_fold_metrics.png"
        plot_fold_metrics_boxplot(results, predictions, box_path)
        artifacts["boxplot_fold_metrics"] = str(box_path)

        logger.info("Усі графіки збережено у %s",
Config.RESULTS_FIGURES_DIR)
        return artifacts

def main() -> None:
    parser = argparse.ArgumentParser(description="Побудова графіків
результатів")
    parser.add_argument("--results", type=Path, default=None)
    parser.add_argument("--predictions-dir", type=Path, default=None)
    args = parser.parse_args()

    logging.basicConfig(level=logging.INFO, format="%%(levelname)s:
%(message)s")
    artifacts = run_all_plots(results_path=args.results,
predictions_dir=args.predictions_dir)
    print("\nЗгенеровано графіків:", len(artifacts))
    print(f"Найкраща модель: {artifacts.get('best_experiment')}")

```

```
if __name__ == "__main__":  
    main()
```

preprocessing.py:

```
"""Попередня обробка датасету Bone для Boruta / RFE."""
```

```
from __future__ import annotations
```

```
import logging
```

```
import numpy as np
```

```
import pandas as pd
```

```
from sklearn.feature_selection import VarianceThreshold
```

```
logger = logging.getLogger(__name__)
```

```
def remove_constant_features(X: pd.DataFrame) -> tuple[pd.DataFrame,  
list[str]]:
```

```
    """Видалити ознаки з нульовою дисперсією.
```

```
    Args:
```

```
    X: Матриця ознак.
```

```
    Returns:
```

```
    Очищений X і список видалених колонок.
```

```
    """
```

```
    selector = VarianceThreshold(threshold=0.0)
```

```
    selector.fit(X)
```

```
    kept_mask = selector.get_support()
```

```
    removed = X.columns[~kept_mask].tolist()
```

```
    logger.info("Видалено константних ознак: %d", len(removed))
```

```
    return X.loc[:, kept_mask], removed
```

```
def remove_duplicate_features(X: pd.DataFrame) -> tuple[pd.DataFrame,
list[str]]:
```

```
    """Видалити дубльовані колонки з ідентичним вмістом.
```

```
    Args:
```

```
    X: Матриця ознак.
```

```
    Returns:
```

```
    Очищений X і список видалених колонок.
```

```
    """
```

```
    X_t = X.T
```

```
    _, unique_indices = np.unique(X_t.values, axis=0, return_index=True)
```

```
    kept_cols = X_t.index[sorted(unique_indices)]
```

```
    removed = [c for c in X.columns if c not in kept_cols]
```

```
    logger.info("Видалено дубльованих ознак: %d", len(removed))
```

```
    return X[kept_cols], removed
```

```
def remove_correlated_features(
```

```
    X: pd.DataFrame,
```

```
    threshold: float = 0.95,
```

```
) -> tuple[pd.DataFrame, list[str]]:
```

```
    """Видалити одну з пари сильно корельованих ознак.
```

```
    Args:
```

```
    X: Матриця ознак.
```

```
    threshold: Поріг абсолютної кореляції.
```

Returns:

Очищений X і список видалених колонок.

```

"""
corr = X.corr().abs()
upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
to_drop = [col for col in upper.columns if any(upper[col] > threshold)]
logger.info("Видалено корельованих ознак ( $|r| > %.2f$ ): %d", threshold,
len(to_drop))
return X.drop(columns=to_drop), to_drop

```

```

def clean_bone_features(
    X: pd.DataFrame,
    correlation_threshold: float = 0.95,
) -> tuple[pd.DataFrame, dict[str, list[str]]]:
    """Повний ланцюжок очищення Bone перед Boruta / RFE.

```

Args:

X: Сирі ознаки з Bone_normalized_data.xlsx.

correlation_threshold: Поріг для видалення корельованих пар.

Returns:

Очищений X і словник із списками видалених ознак на кожному кроці.

```

"""
removed: dict[str, list[str]] = {}
X, removed["constant"] = remove_constant_features(X)
X, removed["duplicates"] = remove_duplicate_features(X)
X, removed["correlated"] = remove_correlated_features(X,
threshold=correlation_threshold)
return X, removed

```



```

run_experiments:
"""Запуск 15 експериментів: 3 методи відбору × 5 моделей."""

from __future__ import annotations

import argparse
import logging
from pathlib import Path
from typing import Any

import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score

from src.screening.config import Config
from src.screening.cross_validation import METRIC_COLUMNS, run_cv
from src.screening.data_loader import load_bone, load_entropia
from src.screening.feature_selection import (
    ENTROPY_FEATURES,
    select_boruta_features,
    select_entropy_features,
    select_rfe_features,
)
from src.screening.models import (
    MODEL_NAMES,
    ModelBuilder,
    build_catboost,
    build_mlp,
    build_multibranch,
    build_stacking,

```

```

    build_xgboost,
    filter_multibranch_groups,
)
from src.screening.preprocessing import clean_bone_features

logger = logging.getLogger(__name__)

FEATURE_METHODS: tuple[str, ...] = ("entropy", "boruta", "rfe")

def _load_feature_list_from_csv(method: str) -> list[str] | None:
    """Прочитати збережений список ознак, якщо CSV існує."""
    path = Config.RESULTS_TABLES_DIR / f"{method}_features.csv"
    if not path.exists():
        return None
    df = pd.read_csv(path)
    col = "x_id" if "x_id" in df.columns else df.columns[0]
    return df[col].astype(str).tolist()

def prepare_feature_sets(
    refresh: bool = False,
    methods: tuple[str, ...] = FEATURE_METHODS,
) -> dict[str, list[str]]:
    """Отримати списки ознак для методів (з CSV або перерахунок на
Bone)."""
    sets: dict[str, list[str]] = {}

    if "entropy" in methods:
        if refresh:

```

```

sets["entropy"] = select_entropy_features(save=True)
else:
    sets["entropy"] = _load_feature_list_from_csv("entropy") or
list(ENTROPY_FEATURES)

need_bone = refresh or any(m in methods for m in ("boruta", "rfe"))
boruta_csv = _load_feature_list_from_csv("boruta") if "boruta" in methods
else None
rfe_csv = _load_feature_list_from_csv("rfe") if "rfe" in methods else None

if "boruta" in methods and boruta_csv and not refresh:
    sets["boruta"] = boruta_csv
if "rfe" in methods and rfe_csv and not refresh:
    sets["rfe"] = rfe_csv

if need_bone and (
    ("boruta" in methods and "boruta" not in sets)
    or ("rfe" in methods and "rfe" not in sets)
    or refresh
):
    logger.info("Boruta/RFE: завантаження Bone...")
    X_bone, y_bone = load_bone()
    X_bone, _ = clean_bone_features(X_bone)
    if "boruta" in methods and ("boruta" not in sets or refresh):
        sets["boruta"] = select_boruta_features(X_bone, y_bone,
save=True)["confirmed"]
    if "rfe" in methods and ("rfe" not in sets or refresh):
        sets["rfe"] = select_rfe_features(X_bone, y_bone, n_features=10, save=True)

return sets

```

```

def get_dataset_for_method(
    method: str,
    feature_sets: dict[str, list[str]],
) -> tuple[pd.DataFrame, pd.Series, list[str]]:
    """Підготувати (X, y) для методу відбору ознак."""
    features = feature_sets[method]

    if method == "entropy":
        X, y = load_entropy()
        missing = [f for f in features if f not in X.columns]
        if missing:
            raise KeyError(f"Entropy: відсутні ознаки {missing}")
        return X[features].copy(), y, features

    X_bone, y = load_bone()
    X_bone, _ = clean_bone_features(X_bone)
    missing = [f for f in features if f not in X_bone.columns]
    if missing:
        raise KeyError(f"Bone ({method}): відсутні ознаки {missing}")
    return X_bone[features].copy(), y, features


def make_model_builder(
    model_name: str,
    column_names: list[str],
) -> ModelBuilder:
    """Фабрика model_builder для run_cv (приймає scale_pos_weight)."""
    n_features = len(column_names)

```

```

if model_name == "xgboost":
    return build_xgboost
if model_name == "catboost":
    return build_catboost
if model_name == "stacking":
    return build_stacking
if model_name == "mlp":

def _mlp_builder(spw: float) -> Any:
    return build_mlp(n_features)

return _mlp_builder

if model_name == "multibranch":
    groups = filter_multibranch_groups(Config.MULTIBRANCH_GROUPS,
column_names)
    if not groups:
        raise ValueError(
            f"Multi-branch: жодна гілка не перетинається з ознаками
{column_names}"
        )
    cols = list(column_names)

def _mb_builder(spw: float) -> Any:
    return build_multibranch(feature_groups=groups, column_names=cols)

return _mb_builder

```

```

        raise ValueError(f"Невідома модель: {model_name}. Доступні:
{MODEL_NAMES}")

```

```

def _summary_row_from_cv(
    method: str,
    model_name: str,
    n_features: int,
    cv_df: pd.DataFrame,
    y_true: np.ndarray,
    y_proba: np.ndarray,
) -> dict[str, Any]:
    """Один рядок підсумку експерименту (mean_f1_opt + OOF AUC)."""
    mean_row = cv_df[cv_df["fold"] == "mean_f1_opt"].iloc[0]
    row: dict[str, Any] = {
        "feature_method": method,
        "model": model_name,
        "experiment_id": f"{method}_{model_name}",
        "n_features": n_features,
        "n_samples": len(y_true),
        "threshold_type": "f1_opt",
        "oof_auc": float(roc_auc_score(y_true, y_proba)),
    }
    for col in METRIC_COLUMNS:
        row[col] = mean_row[col]
        std_col = f"{col}_std"
        if std_col in mean_row.index:
            row[std_col] = mean_row[std_col]
        row["threshold"] = mean_row["threshold"]
    return row

```

```

def save_predictions_npy(
    method: str,
    model_name: str,
    y_true: np.ndarray,
    y_proba: np.ndarray,
) -> Path:
    """Зберегти OOF-прогнози у форматі (n, 2): y_true, y_proba."""
    path = Config.RESULTS_PREDICTIONS_DIR /
    f"{method}_{model_name}.npz"
    np.save(path, np.column_stack([y_true.astype(float),
    y_proba.astype(float)]))
    return path

```

```

def run_all_experiments(
    methods: tuple[str, ...] = FEATURE_METHODS,
    models: tuple[str, ...] = MODEL_NAMES,
    refresh_features: bool = False,
    n_splits: int | None = None,
    save_npz: bool = True,
) -> pd.DataFrame:
    """Запустити всі експерименти CV і зберегти таблицю + прогнози.

```

Args:

methods: Методи відбору ознак.

models: Імена моделей.

refresh_features: Перерахувати Boruta/RFE на Bone.

n_splits: Кількість фолдів (за замовчуванням Config.CV_FOLDS).

save_npz: Додатково зберегти ``{method}\_{model}\_oof.npz`` для DeLong.

Returns:

DataFrame з підсумками 15 експериментів.

"""

Config.ensure_dirs()

n_splits = n_splits or Config.CV_FOLDS

feature_sets = prepare_feature_sets(refresh=refresh_features,
methods=methods)

summary_rows: list[dict[str, Any]] = []

for method in methods:

X, y, features = get_dataset_for_method(method, feature_sets)

logger.info(

"=== %s: %d ознак, %d спостережень ===",

method,

len(features),

len(y),

)

for model_name in models:

exp_id = f"{method}_{model_name}"

logger.info("--- %s ---", exp_id)

builder = make_model_builder(model_name, features)

cv_df, y_true, y_proba = run_cv(

X,

y,

```

builder,
model_name=exp_id if save_npz else exp_id,
n_splits=n_splits,
random_state=Config.RANDOM_SEED,
save_predictions=save_npz,
)

numpy_path = save_predictions_numpy(method, model_name, y_true, y_proba)
row = _summary_row_from_cv(method, model_name, len(features), cv_df,
y_true, y_proba)
summary_rows.append(row)

logger.info(
"%s | OOF AUC=%.4f | mean AUC=%.4f | F1=%.4f",
exp_id,
row["oof_auc"],
row["auc"],
row["f1"],
)
logger.info("Прогнози: %s", numpy_path)

results = pd.DataFrame(summary_rows)
out_path = Config.RESULTS_TABLES_DIR / "experiment_results.csv"
results.to_csv(out_path, index=False, encoding="utf-8-sig")
logger.info("Таблиця результатів: %s (%d рядків)", out_path, len(results))
return results

def _print_results_table(results: pd.DataFrame) -> None:
    """Компактний вивід таблиці в консоль."""

```

```

cols = ["experiment_id", "n_features", "oof_auc", "auc", "f1", "recall",
"specificity"]
cols = [c for c in cols if c in results.columns]
print("\n" + "=" * 72)
print("РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ (mean по фолдах, f1_opt)")
print("=" * 72)
print(results[cols].to_string(index=False,      float_format=lambda      x:
f"{x:.4f}"))
print("=" * 72 + "\n")

```

```

def main() -> None:
    parser = argparse.ArgumentParser(description="15 експериментів
скринінгу MM")
    parser.add_argument(
        "--refresh-features",
        action="store_true",
        help="Перерахувати Boruta/RFE на Bone",
    )
    parser.add_argument(
        "--n-splits",
        type=int,
        default=None,
        help=f"Кількість фолдів CV (за замовчуванням {Config.CV_FOLDS})",
    )
    parser.add_argument(
        "--methods",
        nargs="+",
        default=list(FEATURE_METHODS),
        choices=list(FEATURE_METHODS),
    )

```

```

)
parser.add_argument(
    "--models",
    nargs="+",
    default=list(MODEL_NAMES),
    choices=list(MODEL_NAMES),
)
parser.add_argument("--no-npz", action="store_true", help="Не зберігати
.npz для DeLong")
args = parser.parse_args()

logging.basicConfig(level=logging.INFO, format="%(levelname)s:
%(message)s")

results = run_all_experiments(
    methods=tuple(args.methods),
    models=tuple(args.models),
    refresh_features=args.refresh_features,
    n_splits=args.n_splits,
    save_npz=not args.no_npz,
)
_print_results_table(results)

if __name__ == "__main__":
    main()

```

statistics.py:

```
"""Статистичні тести для порівняння 15 експериментів: DeLong, Wilcoxon,
bootstrap CI."""
```

```
from __future__ import annotations
```

```
import argparse
```

```
import logging
```

```
from itertools import combinations
```

```
from pathlib import Path
```

```
from typing import Any
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
from scipy import stats
```

```
from sklearn.metrics import roc_auc_score
```

```
from src.screening.config import Config
```

```
from src.screening.cross_validation import METRIC_COLUMNS
```

```
logger = logging.getLogger(__name__)
```

```
FOLD_METRIC_INDEX: dict[str, int] = {name: i for i, name in
enumerate(METRIC_COLUMNS)}
```

```
def compute_midrank(x: np.ndarray) -> np.ndarray:
```

```
    """Midrank для обробки зв'язок (DeLong et al., 1988)."""
```

```

order = np.argsort(x)
sorted_x = x[order]
n = len(x)
midranks = np.zeros(n, dtype=np.float64)
i = 0
while i < n:
    j = i
    while j < n and sorted_x[j] == sorted_x[i]:
        j += 1
    midranks[i:j] = 0.5 * (i + j - 1)
    i = j
out = np.empty(n, dtype=np.float64)
out[order] = midranks + 1
return out

```

```

def _fast_delong(
    predictions: np.ndarray,
    n_pos: int,
) -> tuple[np.ndarray, np.ndarray]:
    """Коваріаційна матриця AUC для k класифікаторів (positives перші у
    вибірці)."""
    m = n_pos
    n_neg = predictions.shape[1] - m
    n_models = predictions.shape[0]

    tx = np.empty((n_models, m))
    ty = np.empty((n_models, n_neg))
    tz = np.empty((n_models, m + n_neg))

```

```

for r in range(n_models):
    tx[r, :] = compute_midrank(predictions[r, :m])
    ty[r, :] = compute_midrank(predictions[r, m:])
    tz[r, :] = compute_midrank(predictions[r, :])

aucs = (tz[:, :m].sum(axis=1) - m * (m + 1) / 2.0) / (m * n_neg)

v01 = (tz[:, :m] - tx) / n_neg
v10 = 1.0 - (tz[:, m:] - ty) / m

sx = np.atleast_2d(np.cov(v01))
sy = np.atleast_2d(np.cov(v10))
delongcov = sx / m + sy / n_neg
return aucs, delongcov

```

```

def delong_test(
    y_true: np.ndarray,
    probas_a: np.ndarray,
    probas_b: np.ndarray,
) -> dict[str, float]:
    """Порівняти два корельовані ROC AUC тестом DeLong (1988).

```

Args:

y_true: Бінарні мітки (0/1).
 probas_a: Ймовірності моделі А.
 probas_b: Ймовірності моделі В.

Returns:

auc_a, auc_b, auc_diff, z, p_value.

```
"""
```

```
y_true = np.asarray(y_true).astype(int)
```

```
probas_a = np.asarray(probas_a, dtype=float)
```

```
probas_b = np.asarray(probas_b, dtype=float)
```

```
if not (len(y_true) == len(probas_a) == len(probas_b)):
```

```
    raise ValueError("y_true, probas_a, probas_b мають однакову довжину")
```

```
n_pos = int(y_true.sum())
```

```
n_neg = len(y_true) - n_pos
```

```
if n_pos == 0 or n_neg == 0:
```

```
    return {
```

```
        "auc_a": float("nan"),
```

```
        "auc_b": float("nan"),
```

```
        "auc_diff": float("nan"),
```

```
        "z": float("nan"),
```

```
        "p_value": float("nan"),
```

```
    }
```

```
order = np.argsort(-y_true)
```

```
preds = np.vstack([probas_a[order], probas_b[order]])
```

```
aucs, cov = _fast_delong(preds, n_pos)
```

```
var_diff = float(cov[0, 0] + cov[1, 1] - 2.0 * cov[0, 1])
```

```
var_diff = max(var_diff, 1e-12)
```

```
z = float((aucs[0] - aucs[1]) / np.sqrt(var_diff))
```

```
p_value = float(2.0 * stats.norm.sf(abs(z)))
```

```
return {
```

```
    "auc_a": float(aucs[0]),
```

```

"auc_b": float(aucs[1]),
"auc_diff": float(aucs[0] - aucs[1]),
"z": z,
"p_value": p_value,
}

```

```

def wilcoxon_test(
    metrics_a: np.ndarray,
    metrics_b: np.ndarray,
) -> dict[str, float]:
    """Парний тест Вілсона для метрик по фолдах CV.

```

Args:

metrics_a: Вектор метрик (наприклад, AUC по 5 фолдах).

metrics_b: Вектор метрик другої моделі.

Returns:

statistic, p_value.

"""

```

a = np.asarray(metrics_a, dtype=float)

```

```

b = np.asarray(metrics_b, dtype=float)

```

```

if a.shape != b.shape:

```

```

    raise ValueError(f"Різна довжина векторів: {a.shape} vs {b.shape}")

```

```

if np.allclose(a, b):

```

```

    return {"statistic": 0.0, "p_value": 1.0}

```

```

try:

```

```

    statistic, p_value = stats.wilcoxon(a, b, zero_method="wilcox")

```

```
except ValueError:
    return {"statistic": float("nan"), "p_value": float("nan")}

return {"statistic": float(statistic), "p_value": float(p_value)}
```

```
def bootstrap_ci(
    values: np.ndarray,
    n: int = Config.BOOTSTRAP_N,
    alpha: float = 0.05,
    random_state: int = Config.RANDOM_SEED,
) -> tuple[float, float, float]:
    """95% bootstrap CI методом перцентилів.
```

Args:

values: Вибірка значень (наприклад, AUC по фолдах).

n: Кількість bootstrap-ітерацій.

alpha: Рівень значущості (0.05 → 95% CI).

random_state: Seed.

Returns:

(mean, ci_low, ci_high).

"""

```
values = np.asarray(values, dtype=float)
```

```
if values.size == 0:
```

```
    return float("nan"), float("nan"), float("nan")
```

```
rng = np.random.default_rng(random_state)
```

```
boot_means = np.empty(n, dtype=float)
```

```
n_samples = len(values)
```

```

for i in range(n):
    sample = values[rng.integers(0, n_samples, n_samples)]
    boot_means[i] = sample.mean()

lo = float(np.percentile(boot_means, 100 * alpha / 2))
hi = float(np.percentile(boot_means, 100 * (1 - alpha / 2)))
return float(values.mean()), lo, hi

```

```

def bootstrap_auc_ci(
    y_true: np.ndarray,
    y_proba: np.ndarray,
    n: int = Config.BOOTSTRAP_N,
    alpha: float = 0.05,
    random_state: int = Config.RANDOM_SEED,
) -> tuple[float, float, float]:
    """Bootstrap 95% CI для OOF AUC (перцентильный метод)."""
    y_true = np.asarray(y_true).astype(int)
    y_proba = np.asarray(y_proba, dtype=float)
    rng = np.random.default_rng(random_state)
    n_samples = len(y_true)
    boot_aucs = []

    for _ in range(n):
        idx = rng.integers(0, n_samples, n_samples)
        if len(np.unique(y_true[idx])) < 2:
            continue
        boot_aucs.append(roc_auc_score(y_true[idx], y_proba[idx]))

```

```

if not boot_aucs:
    point = roc_auc_score(y_true, y_proba)
    return point, float("nan"), float("nan")

boot_aucs_arr = np.asarray(boot_aucs)
lo = float(np.percentile(boot_aucs_arr, 100 * alpha / 2))
hi = float(np.percentile(boot_aucs_arr, 100 * (1 - alpha / 2)))
return float(roc_auc_score(y_true, y_proba)), lo, hi

def load_experiment_predictions(
    predictions_dir: Path | None = None,
) -> dict[str, dict[str, np.ndarray]]:
    """Завантажити OOF-прогнози та fold-метрики з ``*_oof.npz``."""
    pred_dir = predictions_dir or Config.RESULTS_PREDICTIONS_DIR
    experiments: dict[str, dict[str, np.ndarray]] = {}

    for npz_path in sorted(pred_dir.glob("*_oof.npz")):
        exp_id = npz_path.stem.replace("_oof", "")
        if exp_id.endswith("_smoke") or exp_id == "xgb_entropia_smoke":
            continue
        data = np.load(npz_path)
        experiments[exp_id] = {
            "y_true": data["y_true"],
            "y_proba": data["y_proba"],
            "fold_metrics": data["fold_metrics"],
        }

    if not experiments:
        raise FileNotFoundError(f"Не знайдено *_oof.npz у {pred_dir}")

```

```

        logger.info("Завантажено %d експериментів з %s", len(experiments),
pred_dir)

```

```

    return experiments

```

```

def _pairwise_pvalue_matrix(
    experiment_ids: list[str],
    experiments: dict[str, dict[str, np.ndarray]],
    test_fn: Any,
) -> pd.DataFrame:
    """Побудувати симетричну матрицю p-values для всіх пар."""
    n = len(experiment_ids)
    matrix = np.eye(n, dtype=float)

```

```

    for i, j in combinations(range(n), 2):
        id_a, id_b = experiment_ids[i], experiment_ids[j]
        p = test_fn(experiments[id_a], experiments[id_b])
        matrix[i, j] = p
        matrix[j, i] = p

```

```

    return pd.DataFrame(matrix, index=experiment_ids,
columns=experiment_ids)

```

```

def compute_delong_matrix(
    experiments: dict[str, dict[str, np.ndarray]],
) -> pd.DataFrame:
    """Матриця p-values DeLong (OOF AUC) для всіх пар експериментів."""
    ids = sorted(experiments.keys())

```

```

def _test(exp_a: dict[str, np.ndarray], exp_b: dict[str, np.ndarray]) -> float:
    y_a, y_b = exp_a["y_true"], exp_b["y_true"]
    if not np.array_equal(y_a, y_b):
        raise ValueError("y_true різні між експериментами — порівняння
неможливе")
    return delong_test(y_a, exp_a["y_proba"], exp_b["y_proba"])["p_value"]

return _pairwise_pvalue_matrix(ids, experiments, _test)

```

```

def compute_wilcoxon_matrix(
    experiments: dict[str, dict[str, np.ndarray]],
    metric: str = "auc",
) -> pd.DataFrame:
    """Матриця p-values Wilcoxon для fold-метрик (за замовчуванням
AUC)."""
    ids = sorted(experiments.keys())
    idx = FOLD_METRIC_INDEX[metric]

```

```

def _test(exp_a: dict[str, np.ndarray], exp_b: dict[str, np.ndarray]) -> float:
    fold_a = exp_a["fold_metrics"][:, idx]
    fold_b = exp_b["fold_metrics"][:, idx]
    return wilcoxon_test(fold_a, fold_b)["p_value"]

return _pairwise_pvalue_matrix(ids, experiments, _test)

```

```

def compute_bootstrap_summary(
    experiments: dict[str, dict[str, np.ndarray]],

```

```

) -> pd.DataFrame:

"""Bootstrap 95% CI для OOF AUC та fold AUC кожного
експерименту."""
rows: list[dict[str, Any]] = []
for exp_id in sorted(experiments.keys()):
    exp = experiments[exp_id]
    oof_auc, oof_lo, oof_hi = bootstrap_auc_ci(exp["y_true"], exp["y_proba"])
    fold_auc = exp["fold_metrics"][:, FOLD_METRIC_INDEX["auc"]]
    fold_mean, fold_lo, fold_hi = bootstrap_ci(fold_auc)

    rows.append(
        {
            "experiment_id": exp_id,
            "oof_auc": oof_auc,
            "oof_auc_ci_low": oof_lo,
            "oof_auc_ci_high": oof_hi,
            "fold_auc_mean": fold_mean,
            "fold_auc_ci_low": fold_lo,
            "fold_auc_ci_high": fold_hi,
        }
    )
return pd.DataFrame(rows)

def plot_pvalue_heatmap(
    matrix: pd.DataFrame,
    title: str,
    save_path: Path,
    log_scale: bool = True,
) -> None:

```

```

"""Heatmap p-values (-log10) для матриці порівнянь."""
Config.ensure_dirs()
values = matrix.astype(float).values.copy()
np.fill_diagonal(values, np.nan)

if log_scale:
    plot_values = -np.log10(np.clip(values, 1e-300, 1.0))
    cbar_label = "-log10(p-value)"
else:
    plot_values = values
    cbar_label = "p-value"

fig_size = max(10, 0.55 * len(matrix))
fig, ax = plt.subplots(figsize=(fig_size, fig_size * 0.9))
sns.heatmap(
    plot_values,
    xticklabels=matrix.columns,
    yticklabels=matrix.index,
    cmap="YlOrRd",
    annot=False,
    linewidths=0.3,
    cbar_kws={"label": cbar_label},
    ax=ax,
)
ax.set_title(title, fontsize=13)
plt.xticks(rotation=45, ha="right", fontsize=8)
plt.yticks(rotation=0, fontsize=8)
plt.tight_layout()
plt.savefig(save_path, dpi=Config.FIGURE_DPI, bbox_inches="tight")
plt.close()

```

```

logger.info("Heatmap: %s", save_path)

def run_all_statistics(
    predictions_dir: Path | None = None,
) -> dict[str, pd.DataFrame]:
    """Запустити DeLong, Wilcoxon, bootstrap CI; зберегти CSV та
    heatmaps."""
    Config.ensure_dirs()
    experiments = load_experiment_predictions(predictions_dir)

    delong_matrix = compute_delong_matrix(experiments)
    wilcoxon_matrix = compute_wilcoxon_matrix(experiments, metric="auc")
    bootstrap_df = compute_bootstrap_summary(experiments)

    delong_path = Config.RESULTS_STATS_DIR / "delong_pvalues.csv"
    wilcoxon_path = Config.RESULTS_STATS_DIR / "wilcoxon_pvalues.csv"
    bootstrap_path = Config.RESULTS_STATS_DIR / "bootstrap_ci.csv"

    delong_matrix.to_csv(delong_path, encoding="utf-8-sig")
    wilcoxon_matrix.to_csv(wilcoxon_path, encoding="utf-8-sig")
    bootstrap_df.to_csv(bootstrap_path, index=False, encoding="utf-8-sig")

    plot_pvalue_heatmap(
        delong_matrix,
        "DeLong test: p-values (OOF AUC)",
        Config.RESULTS_FIGURES_DIR / "delong_heatmap.png",
    )
    plot_pvalue_heatmap(
        wilcoxon_matrix,

```

```

"Wilcoxon test: p-values (fold AUC)",
Config.RESULTS_FIGURES_DIR / "wilcoxon_heatmap.png",
)

```

```

logger.info("DeLong p-values: %s", delong_path)
logger.info("Wilcoxon p-values: %s", wilcoxon_path)
logger.info("Bootstrap CI: %s", bootstrap_path)

```

```

return {
    "delong": delong_matrix,
    "wilcoxon": wilcoxon_matrix,
    "bootstrap": bootstrap_df,
}

```

```

def _print_significant_pairs(
    matrix: pd.DataFrame,
    test_name: str,
    alpha: float = 0.05,
    top_n: int = 10,
) -> None:
    """Вывести топ пар з  $p < \alpha$ ."""
    pairs: list[tuple[str, str, float]] = []
    ids = list(matrix.index)
    for i in range(len(ids)):
        for j in range(i + 1, len(ids)):
            p = float(matrix.iloc[i, j])
            if p < alpha:
                pairs.append((ids[i], ids[j], p))

```

```

pairs.sort(key=lambda t: t[2])
print(f"\n{test_name}: пари з  $p < \{\alpha\}$  (топ- $\{\text{top\_n}\}$ )")
print("-" * 60)
for a, b, p in pairs[:top_n]:
    print(f" {a} vs {b}:  $p=\{p:.4f\}$ ")
if not pairs:
    print(" (немає значущих пар)")
print()

def main() -> None:
    parser = argparse.ArgumentParser(description="Статистика порівняння 15
експериментів")
    parser.add_argument(
        "--predictions-dir",
        type=Path,
        default=None,
        help="Каталог з *_oof.npz (за замовчуванням results/predictions/)",
    )
    args = parser.parse_args()

    logging.basicConfig(level=logging.INFO,          format="%%(levelname)s:
%(message)s")

    results = run_all_statistics(predictions_dir=args.predictions_dir)
    _print_significant_pairs(results["delong"], "DeLong")
    _print_significant_pairs(results["wilcoxon"], "Wilcoxon")

if __name__ == "__main__":
    main()

```