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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Аналіз збіжності та надійності бінарної класифікації текстових даних на основі стохастичних методів та латентних представлень»

Виконав:
студент IV курсу, групи КА-21
Христов Максим Богданович _____

Керівник:
Доцент кафедри ММСА, к.ф.-м.н., доц.
Спекторський Ігор Якович _____

Консультант з економічного розділу:
Доцент кафедри ЕК, к.е.н., доц.
Рощина Надія Василівна _____

Консультант з нормоконтролю:
Доцент кафедри ММСА, к.ф.-м.н.
Статкевич Віталій Михайлович _____

Рецензент:
Доцент кафедри МА та ТЙ, к.ф.-м.н., доц.
Ільєнко Андрій Борисович _____

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 математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Христову Максиму Богдановичу

1. Тема роботи «Аналіз збіжності та надійності бінарної класифікації текстових даних на основі стохастичних методів та латентних представлень», керівник роботи Спекторський Ігор Якович, доцент кафедри ММСА, к.ф.-м.н., доц., затверджені наказом по університету від 27.05.2026 р. № 1944-с.
2. Термін подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: основні відомості теорії ймовірностей, теорії класифікації і машинного навчання.
4. Зміст роботи: дослідження предметної області задачі бінарної класифікації тональності текстів соціальних мереж; математичне обґрунтування збіжності метрик класифікатора на основі закону великих чисел та трьох довірчих інтервалів (Чебишова, Вальда, Хеффдінга); програмна реалізація системи класифікації з архітектурою Word2Vec → Denoising Autoencoder → MLP; експериментальна верифікація теоретичних результатів через bootstrap-аналіз на датасеті Sentiment140; функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання ___26 лютого 2026_____

Календарний план

№ з/п	Назва етапів виконання бакалаврської дипломної роботи (БДР)	Термін виконання етапів роботи	Примітка
1	Затвердження теми БДР, опанування ДСТУ 3008:2015	26.02.2026-28.02.2026	виконано
2	Огляд літератури за темою БДР та вибір датасету	01.03.2026-28.03.2026	виконано
3	Структурування основних теоретичних конструкцій та розробка програмної частини БДР	20.03.2026-13.04.2026	виконано
4	Аналіз, отриманих експериментальним шляхом, даних та їх поєднання з теоретичним апаратом	13.04.2026-15.05.2026	виконано
5	Оформлення пояснювальної записки (ПЗ), перевірка правильності структури та оформлення	03.05.2026-14.05.2026	виконано
6	Загальне редагування розділів 1-3, виконання функціонально-вартісного аналізу	15.05.2026-01.06.2026	виконано
7	Остаточне оформлення ПЗ, підготовка презентації	01.06.2026-08.06.2026	виконано
8	Передзахист БДР		виконано

Студент _____ Максим ХРИСТОВ

Керівник _____ Ігор СПЕКТОРСЬКИЙ

РЕФЕРАТ

Дипломна робота: 140 с., 15 табл., 16 рис., 2 додатки , 20 джерел.

БІНАРНА КЛАСИФІКАЦІЯ, ДОВІРЧИЙ ІНТЕРВАЛ, ЗАКОН ВЕЛИКИХ ЧИСЕЛ, BOOTSTRAP, СТОХАСТИЧНА ЗБІЖНІСТЬ, ДЕНОЙЗИНГОВИЙ АВТОЕНКОДЕР, WORD2VEC, SENTIMENT140.

Об'єкт дослідження — процес бінарної класифікації текстових даних соціальних мереж.

Предмет дослідження — метрики якості бінарного класифікатора та закономірності їх стохастичної стабільності залежно від обсягу вибірки.

Мета роботи — дослідити вплив обсягу вибірки на стабільність метрик бінарної класифікації тексту та визначити мінімальний обсяг даних, достатній для отримання математично надійних результатів.

Результат роботи — побудовано стохастичну модель бінарного класифікатора, збіжність оцінок якої обґрунтовано через закон великих чисел Хінчина та посилений закон Колмогорова. Виведено три довірчі інтервали для оцінок параметрів — на основі нерівності Чебишова, нерівності Хеффдінга та інтервалу Вальда зі сталим співвідношенням ширини 2.28 : 1.39 : 1.00. Реалізовано програмний pipeline з архітектурою Word2Vec → Denoising Autoencoder → MLP на датасеті Sentiment140 (1.6 млн твітів), що досягає F1-міри 0.7566. Експериментально верифіковано теоретичні результати через bootstrap-аналіз: швидкість збіжності $O(\frac{1}{\sqrt{n}})$ відтворено з точністю 10.52 проти 10.00, ймовірність покриття для меж Чебишова та Хеффдінга — 1.000, для межі Вальда — 0.96. Встановлено дві точки насичення: $n^*\_train \approx 100\,000$ та $n^*\_test \approx 25\,000$.

Перспективи подальшого розвитку — створення основи для поширення розробленої методології аналізу надійності на багатокласову класифікацію, задачі з незбалансованими класами та сучасні трансформерні архітектури.

ABSTRACT

Thesis: 140 pages, 15 tables, 16 figures, 2 appendices, 20 references.

BINARY CLASSIFICATION, CONFIDENCE INTERVAL, LAW OF LARGE NUMBERS, BOOTSTRAP, STOCHASTIC CONVERGENCE, DENOISING AUTOENCODER, WORD2VEC, SENTIMENT140.

Object of research — the process of binary classification of textual data from social networks.

Subject of research is the set of the quality metrics for a binary classifier and the regularities of their stochastic stability depending on the sample size.

The aim of the work — to investigate the influence of sample size on the stability of binary text classification metrics and to determine the minimum amount of data sufficient to obtain mathematically reliable results.

The main results — a stochastic model of a binary classifier is constructed, the convergence of model estimates is justified through Khinchin's law of large numbers and Kolmogorov's strong law. Three confidence intervals for parameter estimates are derived — based on Chebyshev's inequality, Hoeffding's inequality, and the Wald interval, with a stable width ratio of 2.28 : 1.39 : 1.00. A software pipeline with the architecture Word2Vec → Denoising Autoencoder → MLP is implemented on the Sentiment140 dataset (1.6 M tweets), achieving an F1-score of 0.7566. The theoretical results are experimentally verified via bootstrap analysis: the convergence rate $O(\frac{1}{\sqrt{n}})$ is reproduced with an accuracy of 10.52 against 10.00, the coverage probability for Chebyshev and Hoeffding bounds equals 1.000, and for Wald it is 0.96. Two saturation points are established: $n^*\_train \approx 100\,000$ and $n^*\_test \approx 25\,000$.

As the prospects for further development we provide a foundation for extending the developed reliability analysis methodology to multiclass classification, imbalanced class problems, and modern transformer architectures .

ЗМІСТ

ВСТУП

Сучасні системи автоматичної обробки тексту щодня класифікують мільярди повідомлень у соціальних мережах, фільтрують спам, аналізують відгуки споживачів та виявляють токсичний контент [13]. За цією практичною складовою стоїть фундаментальна наукова задача: наскільки можна довіряти результатам такої класифікації, і яка кількість даних необхідна для того, щоб ці результати стали надійними?

Більшість досліджень у галузі машинного навчання зосереджена на досягненні максимальної загальної точності (Accuracy або скорочено Acc) на конкретному наборі даних, залишаючи поза увагою питання стабільності отриманих оцінок [5]. Повідомлення про значення Acc без зазначення довірчого інтервалу та мінімального обсягу вибірки є науково неповним — адже ця оцінка могла б суттєво відрізнятись при іншій підвибірці тих самих даних. Класичні результати теорії ймовірностей, зокрема закон великих чисел та центральна гранична теорема [1, 2], дають теоретичну основу для аналізу такої стабільності, проте на практиці їх рідко поєднують з реальними експериментами на масштабних датасетах.

Існуючі роботи, як правило, розглядають окремо або теоретичні нерівності (Чебишова, Хеффдінга), або емпіричні методи статистичного оцінювання, такі як bootstrap-аналіз [5]. Відсутнє системне дослідження, у якому всі три класичні межі довірчого інтервалу — Чебишова, Вальда та Хеффдінга — одночасно виводяться, перевіряються експериментально на одному реальному датасеті та зіставляються між собою за швидкістю збіжності та ймовірністю покриття. Це створює прогалину між математичною теорією та практикою побудови класифікаторів, особливо в контексті сучасних методів вилучення латентних ознак на основі векторних представлень слів [9] та автоенкодерів [10].

Дана робота присвячена дослідженню впливу обсягу вибірки на стабільність метрик бінарної класифікації тональності твітів. Центральним результатом є визначення мінімального обсягу вибірки, при якому оцінки метрик стабілізуються у межах заданого довірчого інтервалу, що підтверджується як теоретичним аналізом збіжності, так і експериментами на датасеті Sentiment140 обсягом 1 600 000 твітів [13].

Об'єктом дослідження є процес бінарної класифікації текстових даних соціальних мереж.

Предметом дослідження є метрики якості бінарного класифікатора та закономірності їх стохастичної стабільності залежно від обсягу вибірки.

Мета роботи — дослідити вплив обсягу вибірки на стабільність метрик бінарної класифікації тональності тексту та визначити мінімальний обсяг даних, достатній для отримання математично надійних результатів.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Задача бінарної класифікації тексту

Класифікація тексту — це автоматичне віднесення текстового документу до однієї з наперед визначених категорій. Бінарна класифікація є окремим випадком, де таких категорій рівно дві. Формально: маючи текстовий об'єкт x, необхідно побудувати функцію $f: X \rightarrow \{0, 1\}$, яка відображає простір текстів у простір міток класів. У даній роботі клас 1 відповідає позитивній тональності твіту, клас 0 — негативній.

Задача аналізу тональності тексту є однією з найпоширеніших практичних задач обробки природної мови. Щодня платформа Twitter генерує сотні мільйонів публікацій, і автоматичне визначення їх емоційного забарвлення є важливим інструментом для моніторингу суспільних настроїв, аналізу репутації брендів та дослідження соціальних явищ.

Особливістю текстів соціальних мереж є їх неформальний характер: скорочення, сленг, емодзі, граматичні помилки. Це ускладнює автоматичну обробку порівняно з традиційними текстами та вимагає спеціальних методів попередньої обробки та вилучення ознак.

1.2 Огляд існуючих методів бінарної класифікації тексту

Методи автоматичної класифікації тексту пройшли тривалий шлях розвитку. Розглянемо основні підходи та їх характеристики з точки зору задачі даної роботи.

1.2.1 Класичні статистичні методи

Наївний класифікатор Байєса спирається на теорему Байєса з припущенням про незалежність ознак. Попри простоту він демонструє прийнятну загальну точність (Acc) для задач аналізу тональності. Метод опорних векторів знаходить оптимальну гіперплощину, що розділяє класи у просторі ознак і при використанні TF-IDF векторизації показує конкурентні результати навіть порівняно з нейронними мережами. Логістична регресія, завдяки інтерпретованості, широко застосовується як базовий метод порівняння у задачах бінарної класифікації тексту [3].

1.2.2 Нейронні мережі та трансформери

Рекурентні мережі LSTM враховують порядок слів у тексті та здатні фіксувати довгострокові залежності. Проте вони вимагають значних обчислювальних ресурсів та погано паралелізуються. Трансформерні моделі BERT та GPT демонструють результати на рівні людини у багатьох задачах обробки природної мови, однак є складними з точки зору інтерпретації та потребують великих обчислювальних потужностей [4].

1.2.3 Автоенкодери для вилучення ознак

Автоенкодер навчається стискати вхідні дані у компактне латентне представлення, зберігаючи максимально важливу інформацію. На відміну від наскрізного навчання, автоенкодер явно виділяє ознаки, які передаються до простішого класифікатора. Це дозволяє досліджувати якість ознак окремо від якості класифікатора, що є важливим для цілей даної роботи [10, 12].

1.2.4 Багатошарові персептрони (MLP)

Як остаточний класифікатор, у даній роботі використовується багатошаровий персептрон з повнозв'язаною архітектурою. Теоретичним обґрунтуванням є теорема універсальної апроксимації [7], яка стверджує, що багатошаровий персептрон з достатньою кількістю прихованих нейронів та неполіноміальною функцією активації здатний апроксимувати будь-яку неперервну функцію на компактному просторі з довільною точністю. Це гарантує, що навіть проста архітектура MLP теоретично спроможна вирішити задачу класифікації, якщо ознаки на її вході є інформативними.

Порівняльний аналіз розглянутих підходів з точки зору вимог даної роботи наведено у таблиці 1.1:

Таблиця 1.1 — Порівняльний аналіз методів класифікації тексту

Метод	**Якість класифікації**	**Масштабованість**	**Інтерпретованість**	**Придатність для аналізу збіжності**
Naive Bayes / SVM	Середня	Висока	Висока	Висока
LSTM / RNN	Висока	Низька	Низька	Низька
BERT / GPT	Дуже висока	Дуже низька	Дуже низька	Дуже низька
Автоенкодер + MLP	Висока	Висока	Середня	Висока

1.3 Проблематика надійності оцінок метрик у машинному навчанні

Стандартною практикою у дослідженнях з машинного навчання є повідомлення єдиного числового значення метрики якості класифікатора — наприклад, F1-міри або загальної точності — отриманого на тестовій вибірці фіксованого обсягу. Така практика, попри свою поширеність, є науково неповною, оскільки не враховує стохастичної природи самого процесу оцінювання: метрика, обчислена на конкретній вибірці, є випадковою величиною, значення якої може відчутно змінюватися при заміні тестової вибірки на іншу, отриману з того самого розподілу [3, 5].

Цю проблему добре ілюструє така ситуація: модель, що демонструє точність 0.85 на одній тестовій вибірці, може показати 0.83 або 0.87 на інших вибірках того самого обсягу. Жодне з цих значень саме по собі не дає повної картини якості моделі. Повне статистичне твердження має містити не лише точкову оцінку, а й довірчий інтервал, що характеризує її надійність, наприклад «точність становить 0.85 ± 0.02 при обсязі вибірки 25 000 та рівні довіри 0.95» [1].

З цієї проблеми випливає важливе питання, що має не лише теоретичну, але й суто практичну вагу: який мінімальний обсяг тестової вибірки потрібен для отримання надійної оцінки метрик якості з контрольованою похибкою? Відповідь залежить як від властивостей самого класифікатора (близькість істинних параметрів до значень 0 або 1, що визначає дисперсію оцінки), так і від вимог до точності оцінювання. Для індикаторних випадкових величин, з якими оперують у задачах класифікації, дисперсія оцінок параметрів описується класичною формулою Бернуллі, а швидкість їх збіжності до істинних значень — фундаментальними теоремами теорії ймовірностей [1, 2].

Аналогічне питання постає і щодо тренувальної вибірки: при якому її обсязі досягається стелі якості, що відповідає виразності обраної архітектури

моделі? Це питання має іншу природу — воно стосується не точності вимірювання, а здатності моделі вчитися з даних. Експериментально воно досліджується через побудову кривих навчання (learning curve), теоретичне обґрунтування для яких також спирається на закони великих чисел.

У сукупності ці два питання — про точку насичення тренувальної вибірки та про точку надійної оцінки тестової вибірки — становлять предмет цього дослідження. Їх систематичне вивчення з використанням сучасного апарату теорії ймовірностей дозволяє замінити поширену емпіричну практику повідомлення єдиного значення метрики на статистично обґрунтоване твердження з кількісною характеристикою надійності.

1.4 Датасет Sentiment140

Для дослідження обрано датасет Sentiment140 — 1 600 000 твітів з бінарною розміткою тональності, розроблений дослідниками Стенфордського університету у 2009 році [13]. Датасет сформований методом дистантного навчання: твіти автоматично розмічались на основі наявності позитивних або негативних емодзі, що забезпечує велику вибірку без ручної розмітки.

Мітки у форматі $\{0, 4\}$ тривіально перетворюються у стандартний формат $\{0, 1\}$. Баланс класів 50/50 дозволяє коректно застосовувати всі метрики, включно із Загальною точністю (Acc). Головна перевага датасету — масштаб: 1 600 000 об'єктів дозволяють досліджувати збіжність метрик на діапазоні від $n = 500$ до $n = 1\,600\,000$, що охоплює чотири порядки величини.

Специфічною характеристикою є неформальний характер твітів — скорочення, емодзі, граматичні помилки. Це підвищує варіативність векторних представлень та є додатковим фактором дисперсії при оцінюванні метрик, вплив якого також буде досліджений у практичній частині.

1.5 Постановка задачі та методологічний підхід

На основі проведеного огляду предметної області сформульована задача дослідження: розробити та експериментально верифікувати методологію аналізу збіжності та надійності метрик бінарної класифікації текстових даних з використанням стохастичних методів та латентних представлень.

Об'єктом дослідження виступає процес бінарної класифікації текстових даних соціальних мереж. Предметом дослідження є метрики якості бінарного класифікатора та закономірності їх стохастичної стабільності залежно від обсягу вибірки.

Для досягнення поставленої мети розв'язуються такі три задачі:

1) побудова стохастичної моделі бінарного класифікатора з обґрунтуванням збіжності її параметрів через закони великих чисел;

2) виведення трьох фундаментальних довірчих інтервалів для оцінок параметрів (за нерівністю Чебишова, за інтервалом Вальда та за нерівністю Хеффдінга);

3) реалізація системи класифікації тональності текстів з використанням сучасних методів вилучення латентних ознак; експериментальна перевірка теоретичних результатів на масштабному реальному датасеті через bootstrap-аналіз.

Обраний методологічний підхід поєднує теоретичний та експериментальний рівні аналізу. На теоретичному рівні використовуються фундаментальні результати теорії ймовірностей, що забезпечують математичну строгість обґрунтування. На експериментальному рівні застосовується bootstrap-метод як універсальний інструмент перевірки теоретичних меж на реальних даних з обчисленням ймовірності покриття (Coverage Probability) — частки повторень, у яких побудований інтервал реально містить істинне значення параметра. Така комбінація дозволяє не

лише сформулювати методологію, але й кількісно оцінити її валідність на конкретному прикладі.

Висновки до розділу 1

У першому розділі проведено дослідження предметної області — задачі бінарної класифікації тональності текстових даних соціальних мереж. Формалізовано саму задачу як побудову функції, що відображає простір текстів у простір міток класів, та обґрунтовано її актуальність з огляду на масштаб генерованого користувачами контенту в сучасних соціальних мережах.

Виконано аналіз існуючих методів автоматичної класифікації тексту — від класичних статистичних підходів (наївний Байєсів класифікатор, метод опорних векторів) до сучасних нейромережевих архітектур (LSTM, BERT, трансформерні моделі). Розглянуто переваги та обмеження кожного підходу. Як методологічну основу для подальшого дослідження обрано схему, що поєднує вилучення латентних ознак автоенкодером із класифікацією багатошаровим персептроном, оскільки вона дозволяє досліджувати властивості ознак незалежно від властивостей класифікатора.

Окрему увагу приділено проблематиці надійності оцінок метрик у машинному навчанні. Показано, що поширена практика повідомлення єдиного значення метрики без оцінки її стабільності є науково неповною, оскільки не враховує стохастичної природи процесу оцінювання. Сформульовано центральне питання дослідження — про мінімальний обсяг вибірки, при якому оцінки метрик стають достатньо надійними у заданому довірчому інтервалі.

Обґрунтовано вибір датасету Sentiment140 як емпіричної бази дослідження. Його обсяг у 1 600 000 твітів зі збалансованим розподілом класів забезпечує умови, необхідні для систематичного дослідження залежності надійності оцінок від обсягу вибірки на широкому діапазоні масштабів.

Сформульовано об'єкт, предмет і задачі дослідження, окреслено методологічний підхід, що поєднує теоретичний аналіз через апарат теорії ймовірностей з експериментальною верифікацією через bootstrap-метод. Це створює основу для подальшого розгортання математичних основ роботи у другому розділі та програмної реалізації з аналізом результатів у третьому розділі.

РОЗДІЛ 2 МАТЕМАТИЧНІ ТА МЕТОДИЧНІ ОСНОВИ РОБОТИ

2.1 Стохастична модель бінарного класифікатора

Щоб результат класифікації мав наукову цінність, недостатньо отримати число — необхідна математично строга основа для оцінювання його надійності. У даній роботі використовується стохастична модель, що дозволяє не лише отримати числове значення метрики, а й обґрунтувати умови, за яких воно є стабільним.

Формальна постановка

Розглянемо послідовність текстових об'єктів, де кожен k-й об'єкт належить одному з двох класів:

$$b_{\mathrm{k}} \in \{0, 1\}, \quad k = 1, 2, 3, \ldots$$

Класифікатор отримує характеристики об'єкту та виносить рішення $\xi_{\mathrm{k}} \in \{0, 1\}$. Якість роботи класифікатора визначається чотирма параметрами:

$\boldsymbol{p_1} = P\{\xi_{\mathrm{k}} = 1 \mid b_{\mathrm{k}} = 1\}$ — ймовірність правильного розпізнавання позитивного об'єкту;

$\boldsymbol{p_0} = P\{\xi_{\mathrm{k}} = 0 \mid b_{\mathrm{k}} = 0\}$ — ймовірність правильного розпізнавання негативного об'єкту;

$\boldsymbol{q_0} = 1 - p_0$ — ймовірність помилкового спрацьовування: негативний об'єкт названий позитивним;

$\boldsymbol{q_1} = 1 - p_1$ — ймовірність пропуску: позитивний об'єкт названий негативним.

На основі вибірки обсягом n об'єктів будуються статистичні оцінки цих параметрів:

$$p_1^n = \frac{TP}{n_1}\ ; \quad q_0^n = \frac{FP}{n_0}; \quad p_0^n = \frac{TN}{n_0}; \quad q_1^n = \frac{FN}{n_1},$$

де TP (True Positive) — правильно розпізнані позитивні, TN (True Negative) — правильно розпізнані негативні, FP (False Positive) — негативні названі позитивними, FN (False Negative) — позитивні названі негативними; n_1 та n_0 — кількість об'єктів кожного класу.

2.2 Закон великих чисел як теоретичне обґрунтування збіжності

Кожна з побудованих оцінок є середнім арифметичним індикаторних випадкових величин. Розглянемо детальніше оцінку p_1^n — її можна записати у вигляді:

$$p_1^n = \frac{1}{n_1} \cdot \sum_{i=1}^{n_1} Y_i\,, \quad \text{де } Y_i = 1\{\xi_i = 1 | b_i = 1\}$$

Тут $Y_1, Y_2, \ldots, Y_{n1}$ — незалежні однаково розподілені випадкові величини Бернуллі з параметром p_1:

$$P(Y_i = 1) = p_1, \;\; P(Y_i = 0) = 1 - p_1, \;\; E[Y_i] = p_1, \;\; D[Y_i] = p_1(1 - p_1)$$

2.2.1 Теорема Хінчина (закон великих чисел)

Нехай $X_1, X_2, \ldots$ — послідовність незалежних однаково розподілених випадкових величин зі скінченним математичним сподіванням $\mu = E[X_1]$. Тоді вибіркове середнє збігається до μ за ймовірністю:

$$\bar{X}_n = \frac{1}{n} \cdot \sum_{i=1}^{n} X_i \longrightarrow \mu \,(\text{за ймовірністю при } n \to \infty)$$

тобто для будь-якого $\varepsilon > 0$:

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| \geq \varepsilon) = 0$$

Теорема доводиться через оцінку дисперсії вибіркового середнього: оскільки $D[\bar{X}_n] = \frac{\sigma^2}{n}$ при $n \to \infty$, ймовірність будь-якого фіксованого відхилення від μ прямує до нуля [1, 2]. Детальне виведення наведено у [1].

Більш сильна форма — посилений закон великих чисел (Колмогорова) [2] — стверджує, що збіжність відбувається не лише за ймовірністю, а й майже напевне:

$$P\left(\lim_{n\to\infty} \bar{X}_\text{n} = \mu\right) = 1$$

Це гарантує, що для майже всіх можливих реалізацій вибірки оцінка стабілізується до справжнього значення параметра, що є фундаментальним обґрунтуванням усієї подальшої методології.

2.2.2 Застосування до моделі класифікатора

Підставляючи $X_\text{i} = Y_\text{i}$, маємо безпосередньо:

$$p_1^n \longrightarrow p_1, \quad q_0^n \longrightarrow q_0, \quad p_0^n \longrightarrow p_0, \quad q_1^n \longrightarrow q_1 \quad \text{при } n \to \infty$$

Отже, статистичні оцінки всіх чотирьох параметрів стохастичної моделі збігаються до їхніх істинних значень при необмеженому зростанні обсягу вибірки.

2.2.3 Збіжність похідних метрик

Із збіжності базових оцінок p_1^n та q_0^n за теоремою про неперервне відображення (continuous mapping theorem) [14] випливає збіжність усіх неперервних функцій від них. Зокрема, для Повноти (Recall):

$$r^n = \frac{TP}{TP + FN} \longrightarrow p_1 \quad \text{при } n \to \infty$$

Для Точності (Precision) із урахуванням балансу класів $\alpha = \lim_{\text{n}\to\infty} \frac{n_0}{n_1}$:

$$p^n = \frac{TP}{TP + FP} \longrightarrow \frac{p_1}{p_1 + q_0 \cdot \alpha}, \text{ при } n \to \infty$$

При збалансованих класах $\alpha = 1$, що відповідає датасету Sentiment140 зі співвідношенням 50/50. Тоді гранична Точність (Precision) $\frac{p_1}{p_1+q_0}$ — теоретична межа, до якої прямує спостережуване значення при збільшенні вибірки.

Аналогічно, F1-міра, Загальна точність, MCC, AUC та інші метрики є неперервними функціями від p_1 та q_0, тому їхня збіжність також гарантована теоремою про неперервне відображення.

Таким чином, закон великих чисел дає вичерпну якісну відповідь: усі оцінки параметрів і похідних від них метрик збігаються до своїх істинних значень. Однак для практики цього недостатньо — важливо знати, наскільки точною є оцінка при конкретному скінченному n. Кількісну відповідь дають довірчі інтервали, розглянуті у наступному підрозділі.

2.3 Довірчі інтервали як міра надійності

Закон великих чисел стверджує, що оцінки збігаються, — але не каже, коли саме ця збіжність стає практично корисною. Для оцінки точності оцінювання при конкретному n використовуються довірчі інтервали.

Довірчий інтервал рівня γ для параметра p_1 — це інтервал $[p^n{}_1 - \Delta, p^n{}_1 + \Delta]$ такий, що з ймовірністю не нижче γ він містить істинне значення p_1:

$$P(\,|p_1{}^{\mathrm{n}} - p_1| < \Delta\,) \geq \gamma$$

Ширина Δ залежить від n та від того, які припущення ми готові прийняти щодо розподілу оцінки. У цьому підрозділі розглянуто три фундаментальні підходи — від найбільш загального до найбільш тісного.

2.3.1 Нерівність Чебишова

Нерівність Чебишова є найбільш загальним інструментом: вона працює для будь-якого розподілу з визначеною дисперсією і не вимагає жодних додаткових припущень про вигляд цього розподілу.

Теорема (нерівність Чебишова) [1]. Для випадкової величини X з математичним сподіванням μ і скінченною дисперсією σ^2 для будь-якого $t > 0$ виконується нерівність:

$$P(|X - \mu| \geq t) \leq \frac{\sigma^2}{t^2}$$

Застосовуючи цю нерівність до $\bar{X}_n$ з $D[\bar{X}_n] = \frac{\sigma^2}{n}$ та підставляючи $t = \Delta$, отримуємо, що з ймовірністю γ виконується:

$$|\bar{X}_n - \mu| < \Delta, \text{ де } \Delta = \frac{\sigma}{\sqrt{n \cdot (1-\gamma)}}$$

Для індикаторних величин Y_i маємо $D[Y_i] = p_1(1 - p_1) \leq \frac{1}{4}$ (максимум при $p_1 = 0.5$). Підставляючи цю консервативну верхню оцінку дисперсії, одержуємо ширину довірчого інтервалу для p_1 при рівні довіри $\gamma = 0.95$:

$$\Delta_1{}^{\text{Чеб}} n_1 = \frac{1}{2} \cdot \sqrt{\frac{1}{(n_1 \cdot (1-\gamma))}} = \frac{0.5}{\sqrt{n_1 \cdot 0.05}} \approx \frac{2.236}{\sqrt{n_1}}$$

З ймовірністю γ виконується:

$$p_1^{n_1} - \Delta_1 < p_1 < p_1^{n_1} + \Delta_1$$

Ширина Δ_1 спадає пропорційно $\frac{1}{\sqrt{n_1}}$ — математичний вираз закону «більше даних — надійніший результат».

Перевагою нерівності Чебишова є її універсальність — вона працює без будь-яких припущень про розподіл і дає гарантоване покриття. Недоліком є консервативність: отримані межі завжди забезпечують покриття не нижче γ, але можуть бути значно ширшими ніж реально необхідно.

2.3.2 Центральна гранична теорема та інтервал Вальда

Якщо припустити достатньо великий обсяг вибірки, можна використати асимптотичний нормальний розподіл вибіркового середнього і отримати значно тіснішу межу.

Теорема (центральна гранична теорема (ЦГТ), Lindeberg–Lévy) [1, 2].

Нехай $X_1, X_2, \ldots$ — послідовність незалежних однаково розподілених випадкових величин з математичним сподіванням μ та скінченною дисперсією σ^2. Тоді нормоване вибіркове середнє збігається за розподілом до стандартного нормального:

$$\sqrt{n} \cdot \frac{\bar{X}_n - \mu}{\sigma} \to N(0,1) \text{ при } n \to \infty$$

тобто для будь-якого x:

$$\lim_{n\to\infty} P\left(\sqrt{n} \cdot \frac{\bar{X}_n - \mu}{\sigma} \leq x\right) = \Phi(x)$$

де Φ — функція розподілу стандартного нормального розподілу.

Застосування до оцінки p_1

Згідно з ЦГТ при достатньо великому n_1 випадкова величина $p_1^{n_1}$ має розподіл, наближено нормальний з параметрами:

$$p_1^{n_1} \approx N\left(p_1, \frac{p_1(1-p_1)}{n_1}\right)$$

З цього отримуємо інтервал Вальда — довірчий інтервал для p_1 ширини:

$$\Delta_1^{\text{Вальд}}\, n_1 = z_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{p_1(1-p_1)}{n_1}}$$

де $z_{1-\frac{\alpha}{2}}$ — квантиль стандартного нормального розподілу. Для $\gamma = 0.95$ ($\alpha = 0.05$) маємо $z_{0.975} \approx 1.96$.

Підставивши консервативну верхню оцінку дисперсії $p_1(1-p_1) \leq \frac{1}{4}$, отримуємо:

$$\Delta_1^{\text{Вальд,конс}} n_1 = \frac{1.96}{2\sqrt{n_1}} = \frac{0.98}{\sqrt{n_1}}$$

Інтервал Вальда є у 2.28 разу тіснішим за нерівність Чебишова. Це і є «ціна» універсальності: оцінка Чебишова гарантує покриття для будь-якого розподілу, тоді як оцінка Вальда використовує конкретний асимптотичний нормальний розподіл і тому є тіснішим.

2.3.3 Нерівність Хеффдінга

Нерівність Хеффдінга (Hoeffding, 1963) [8] дає проміжну з трьох розглянутих меж для обмежених випадкових величин — а індикаторні $Y_\text{i} \in \{0,1\}$ є саме такими.

Теорема (нерівність Хеффдінга). Нехай $X_1, X_2, \dots, X_\text{n}$ — незалежні випадкові величини такі, що $X_\text{i} \in [a_\text{i}, b_\text{i}]$ для кожного i. Тоді для будь-якого $t > 0$:

$$P(|\bar{X}_n - E[\bar{X}_n]| \geq t) \leq 2 \cdot \exp\left(-\frac{2n^2t^2}{\sum_i (b_i - a_i)^2}\right)$$

Для індикаторних змінних $Y_\text{i} \in [0,1]$ маємо $(b_\text{i} - a_\text{i})^2 = 1$ для всіх i, тому:

$$P\left(\left|p_1^{n_1} - p_1\right| \geq t\right) \leq 2 \cdot \exp(-2n_1t^2)$$

Прирівнюючи праву частину до $1-\gamma$ та розв'язуючи відносно t, отримуємо ширину довірчого інтервалу згідно з нерівністю Хеффдінга:

$$\Delta_1^{\text{Хеф}} n_1 = \sqrt{\frac{1}{2\,n_1} \cdot \ln\frac{2}{1-\gamma}}$$

Для $\gamma = 0.95$ маємо $\ln\frac{2}{0.05} = \ln 40 \approx 3.689$, отже:

$$\Delta_1^{\text{Хеф}} n_1 \approx \frac{1.358}{\sqrt{n_1}}$$

На відміну від нерівності Чебишова, нерівність Хеффдінга використовує обмеженість випадкових величин і дає точнішу оцінку. На відміну від інтервалу Вальда, вона не потребує припущення про великий n і є строгою (неасимптотичною). Таким чином, нерівність Хеффдінга займає проміжне положення: менш вимоглива за інтервал Вальда щодо припущень, але тісніша за нерівність Чебишова.

2.3.4 Порівняльний аналіз трьох меж

Зведемо три підходи у єдину таблицю 2.1 для конкретних значень n.

Таблиця 2.1 — Порівняння ширини довірчого інтервалу при $\gamma = 0.95$

n	n_1	Δ Чебишов	Δ Хеффдінг	Δ Вальд	Чеб/Вальд
500	250	0.1414	0.0859	0.0620	2.28
1 000	500	0.1000	0.0607	0.0438	2.28
5 000	2 500	0.0447	0.0272	0.0196	2.28
10 000	5 000	0.0316	0.0192	0.0139	2.28
50 000	25 000	0.0141	0.0086	0.0062	2.28
100 000	50 000	0.0100	0.0061	0.0044	2.28
240 000	120 000	0.0065	0.0039	0.0028	2.28

Як видно з таблиці 2.1, межі співвідносяться так:

$$\Delta_{\text{Чеб}} > \Delta_{\text{Хеф}} > \Delta_{\text{Вальд}}$$

зі сталими коефіцієнтами:

$$\Delta_{\text{Чеб}} : \Delta_{\text{Хеф}} : \Delta_{\text{Вальд}} = 2.236 : 1.358 : 0.980 \approx 2.28 : 1.39 : 1.00$$

Це впорядкування відображає фундаментальний компроміс між суворістю гарантії та точністю межі:

— оцінка Чебишова не вимагає жодних припущень про розподіл → найширша межа, але абсолютна гарантія покриття для будь-якого розподілу з обмеженою дисперсією;

— оцінка Хеффдінга вимагає обмеженості випадкових величин (для індикаторів виконується автоматично) → проміжна межа, строга (неасимптотична);

— оцінка Вальда вимагає достатньо великого n для асимптотичної нормальності → найтісніша межа, але працює лише асимптотично.

Усі три межі спадають за одним законом $O(\frac{1}{\sqrt{n}})$, що підтверджується як теоретичними виведеннями, так і даними таблиці 2.1.

Знаходження мінімального n^*, при якому $\Delta_1 \leq 0.01$, є одним із ключових завдань практичної частини роботи. Згідно з нерівністю Чебишова це досягається при $n \approx 100\,000$, згідно з інтервалом Вальда — при $n \approx 20\,000$. Експериментальна перевірка всіх трьох меж через bootstrap-аналіз наведена у підрозділі 3.3.

2.4 Метрики оцінювання якості бінарного класифікатора

Кожна метрика відповідає на своє питання та висвітлює різні аспекти роботи класифікатора. Використання лише однієї метрики може дати хибне уявлення про якість системи — особливо при незбалансованих класах або коли різні типи помилок мають різну ціну. У даному розділі всі метрики наводяться в першу чергу українською назвою з відповідним позначенням у дужках.

2.4.1 Повнота (Recall)

Повнота (Recall) позначає частку всіх реально позитивних об'єктів, які класифікатор правильно визначив як позитивні

$$r^n = \frac{TP}{TP + FN}$$

Низька Повнота (Recall) означає, що система пропускає багато позитивних об'єктів — часто відносить їх до негативного класу. Повнота (Recall) є безпосередньою оцінкою параметра p_1 у стохастичній моделі [7].

2.4.2 Точність (Precision)

Точність (Precision) позначає частку об’єктів, які класифікатор визначив позитивними та вони справді є позитивними

$$p^n = \frac{TP}{TP + FP}$$

Між Повнотою (Recall) та Точністю (Precision) існує фундаментальний компроміс. Спроба знайти якнайбільше позитивних об'єктів (підвищити Повноту) неминуче збільшує кількість помилкових спрацьовувань (знижує Точність). Якщо класифікатор вважатиме кожен об'єкт позитивним — Повнота (Recall) $= 1.0$, але Точність (Precision) впаде до рівня частки позитивних у вибірці [7].

2.4.3 F1-міра (F1)

F1-міра (F1) є гармонічним середнім Повноти (Recall) і Точності (Precision) — агрегованою метрикою, що чесно відображає обидва аспекти якості:

$$F1^n = \frac{2 \cdot p^n \cdot r^n}{p^n + r^n}$$

Гармонічне середнє чутливе до низьких значень: якщо одна з метрик мала — F1-міра (F1) також буде малою, незважаючи на високе значення іншої. Це запобігає ситуації коли висока Точність (Precision) маскує катастрофічно низьку Повноту (Recall) [7].

2.4.4 Загальна точність (Accuracy)

Загальна точність (Accuracy або скорочено як Acc) показує яка частка всіх об'єктів класифікована правильно:

$$Acc^n = \frac{TP + TN}{n}$$

Загальна точність (Acc) є надійним показником лише при збалансованих класах. Для датасету Sentiment140 зі співвідношенням 50/50 вона є коректною основною метрикою [7].

2.4.5 Збалансована точність (Balanced Accuracy)

Для ситуацій з незбалансованими класами використовується Збалансована точність (Balanced Accuracy або скорочено як BA) — середнє між чутливістю та специфічністю:

$$BA^n = \frac{p_1^n + p_0^n}{2}$$

Класифікатор, який завжди відповідає "негативний", матиме загальну точність $(Acc) = 95\%$ на незбалансованому датасеті, але збалансовану точність $(BA) = 50\%$ — чесно відображаючи, що модель нічого не навчилась. У даній роботі використовується як контрольна метрика [7].

2.4.6 Коефіцієнт кореляції Метьюса (Matthews Correlation Coefficient)

Коефіцієнт кореляції Метьюса (Matthews Correlation Coefficient або скорочено як MCC) є найбільш збалансованою метрикою для бінарної класифікації — враховує всі чотири комірки матриці помилок:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC приймає значення від -1 до $+1$, де $+1$ — ідеальна класифікація, 0 — результат не кращий за випадкове вгадування. На відміну від інших метрик MCC є симетричним: однаково враховує обидва класи незалежно від їх пропорцій у вибірці [7].

2.4.7 Площа під ROC-кривою (Area Under Curve)

ROC-крива (Receiver Operating Characteristic) відображає залежність між Повнотою (Recall) та часткою хибних спрацьовувань при всіх можливих порогах прийняття рішення. Площа під ROC-кривою (Area Under Curve або скорочено як AUC) має змістовну ймовірнісну інтерпретацію:

$$AUC = P(\hat{y}_{pos} > \hat{y}_{neg})$$

AUC — це ймовірність того, що класифікатор правильно ранжує випадковий позитивний об'єкт вище за випадковий негативний. $AUC = 1$ — ідеальний класифікатор, $AUC = 0.5$ — випадковий. Її перевагою над іншими метриками є незалежність від порогу класифікації та є інваріантною до балансу класів [7].

2.4.8 Середня точність (Average Precision)

Середня точність (Average Precision або скорочено як AP) є інтегральною характеристикою PR-кривої — залежності Точності (Precision) від Повноти (Recall) при різних порогах класифікації:

$$AP = \int_0^1 P(R)\, dR \approx \sum_k \left(R_k - R_{\{k-1\}}\right) \cdot P_k$$

де P_k та R_k — значення Точності та Повноти при k-му порозі.

На відміну від AUC, яка враховує однаково обидва класи, AP фокусується саме на якості позитивного класу — ймовірності того, що серед об'єктів з найвищими передбаченими ймовірностями справді переважають позитивні. У задачах з вираженою важливістю позитивного класу (наприклад, виявлення позитивних відгуків серед загального потоку повідомлень) AP є більш інформативною метрикою. У сучасних дослідженнях AP вважається стандартом поряд з AUC та доповнює його [7].

2.4.9 Каппа Коена (к)

Каппа Коена (к) коригує Загальну точність (Acc) на ймовірність випадкового вгадування:

$$\kappa^n = \frac{Acc^n - p_e}{1 - p_e}$$

де p_e — очікувана точність випадкового класифікатора з тим самим розподілом передбачень. Для Sentiment140 зі збалансованими класами $\kappa \approx 2 \cdot Acc - 1$, тобто загальна точність $(Acc) = 0.85$ відповідає $\kappa \approx 0.70$, що за стандартною шкалою трактується як "суттєва згода" [7].

2.5 Метрики аналізу збіжності та стохастичної стабільності

Окрім метрик якості класифікатора, для аналізу стабільності результатів використовуються показники, що характеризують сам процес збіжності оцінок при зростанні n.

2.5.1 Швидкість збіжності

Показує наскільки змінюється оцінка при додаванні нової порції даних:

$$v_1^n = \left| p_1^n - p_1^{n-\Delta n} \right|$$

коли $v_1{}^{(n)} < \varepsilon = 0.001$, настає стаціонарний режим і подальше збільшення вибірки не дає суттєвого приросту точності оцінки.

2.5.2 Відносна похибка

Дозволяє порівнювати точність оцінок при різних n у єдиній шкалі:

$$\delta_1^n = \frac{\left| p_1^n - p_1^{n_{max}} \right|}{p_1^{n_{max}}} \times 100\%$$

де $p_1^{n_{max}}$— оцінка на повній тестовій вибірці, що приймається за еталон. Графік спадання δ_1^n показує при якому n відносна похибка стає прийнятно малою.

2.5.3 Ймовірність покриття (Coverage Probability)

Ключовим інструментом експериментальної перевірки теоретичних меж довірчих інтервалів є ймовірність покриття — частка bootstrap-повторень, у яких побудований довірчий інтервал реально містить істинне значення параметра:

$$CP(\gamma) = \frac{1}{B}\sum_{b=1}^{B} 1\{\hat{p}_1^b - \Delta_1 \leq p_1 \leq \hat{p}_1^b + \Delta_1\}$$

де B — кількість bootstrap-повторень, $\hat{p}_1^{(b)}$ — оцінка отримана у b-му повторенні, p_1 — істинне значення параметра (на повній тестовій вибірці).

Якщо нерівність Чебишова коректно гарантує $\gamma = 0.95$, то емпірична $CP(0.95) \geq 0.95$. Якщо $CP(0.95) \approx 1.00$ — це означає, що теоретична межа є консервативною і реальна точність вища ніж гарантовано. Якщо $CP(0.95) < 0.95$ — це сигнал про порушення припущень теореми (наприклад, відсутність незалежності між вимірюваннями).

Ймовірність покриття обчислюється окремо для кожної з трьох теоретичних меж (Чебишова, Вальда, Хеффдінга), що дозволяє експериментально порівняти їх якість на реальних даних. Очікувано, межа Вальда дасть CP найближче до номінального значення 0.95 (бо вона тісніша), тоді як межі Чебишова і Хеффдінга дадуть $CP \approx 1.00$ (як консервативні межі).

2.6 Денойзинговий автоенкодер для вилучення латентних ознак

Якість вилучення ознак з тексту безпосередньо визначає верхню межу того, що може досягти класифікатор. У даній роботі для цієї мети використовується автоенкодер — нейронна мережа, що навчається стискати текстові вектори у компактне латентне представлення, зберігаючи максимально важливу семантичну інформацію [6, 7].

2.6.1 Класичне формулювання розрідженого автоенкодера

Класичний розріджений автоенкодер у формулюванні Ендрю Нга [12] використовує сигмоїдну активацію на латентному шарі та штраф Кульбака-Лейблера за розрідженість:

$$L(W,b) = \frac{1}{m}\sum_{i} \|x^{(i)} - \hat{x}^{(i)}\|^2 + \lambda\|W\|_F^2 + \beta\sum_{j} KL\big(\rho\|\hat{\rho}_j\big)$$

де $KL\big(\rho\|\hat{\rho}_j\big) = \rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}$ — штраф, який змушує середню активацію $\hat{\rho}_j$ кожного нейрона латентного шару наближатись до цільового значення ρ (зазвичай 0.05). Функція втрат складається з трьох складових: квадратичної помилки реконструкції, L2-регуляризації ваг (λ — коефіцієнт регуляризації) та KL-штрафу за розрідженість (β — вага розрідженості). Ідея методу полягає в тому, щоб змусити більшість нейронів латентного шару бути неактивними одночасно, що інтерпретується як вибіркове реагування окремих нейронів на різні семантичні концепти.

2.6.2 Експериментальне дослідження стабільності формулювання Нга

У ході практичної реалізації роботи проведено експериментальну перевірку формулювання Нга на усереднених Word2Vec векторах твітів датасету Sentiment140. Встановлено, що класичний метод демонструє нестабільність на високовимірних щільних входах з низькою дисперсією — а саме такими є вектори, отримані шляхом усереднення Word2Vec представлень слів по тексту твіту. Конкретно, експериментально зафіксовано наступне явище: при значеннях β у діапазоні $[0.5;\ 3.0]$ (рекомендований Нгом діапазон) оптимізатор Adam швидко знаходить тривіальне рішення, в якому енкодер видає константне значення близьке до ρ для будь-якого вхідного вектору. При цьому штраф KL досягає мінімального значення (близького до нуля), а

декодер реконструює середній за корпусом вектор з помилкою реконструкції $MSE \approx 0.004$.

Аналіз показав, що в такому стані латентне представлення повністю втрачає інформативність: стандартне відхилення активацій кожного нейрона по тестовій вибірці складає менше 0.001, а коефіцієнт розділимості позитивних та негативних класів у латентному просторі зменшується від 2.13 у вхідному просторі Word2Vec до 0.08 у латенті — практично до рівня випадкового представлення. Це явище схоже з відомим у літературі по варіаційних автоенкодерах терміном posterior collapse [11], коли латентний шар стає інформаційно незалежним від входу.

Причина такої поведінки полягає в комбінації двох факторів. По-перше, усереднення Word2Vec векторів по словах твіту суттєво зменшує дисперсію представлень: різні твіти у векторному просторі групуються щільно навколо корпусного середнього. По-друге, нормалізація MinMaxScaler у діапазон $[0, 1]$ разом із сигмоїдною активацією на латенті створює ситуацію, де "тривіальне рішення" (видавати константу близьку до середнього) дає мінімальну помилку MSE та одночасно нульовий KL-штраф — тобто є глобальним мінімумом функції втрат.

2.6.3 Денойзинговий автоенкодер як стійка альтернатива

Для подолання виявленої нестабільності у даній роботі застосовано денойзинговий автоенкодер (Denoising Autoencoder, DAE) — формулювання запропоноване Vincent [10], яке має потужне теоретичне обґрунтування та практично використовується як стандарт у задачах вилучення ознак з текстових та зображувальних даних.

Принцип DAE полягає у заміні явного штрафу за розрідженість на дві структурні зміни. По-перше, замість KL-штрафу використовується архітектурна розрідженість через bottleneck — стиснення розмірності з 100 до

32 примушує мережу зберігати лише найважливішу інформацію без необхідності у явному регуляризаторі. По-друге, на вхід під час навчання додається гауссів шум: мережа отримує зашумлений вектор $\tilde{x} = x + \varepsilon$ де $\varepsilon \sim N(0, \sigma^2 I)$ і навчається реконструювати чистий оригінал x. Це формулювання має потужне теоретичне обґрунтування — мінімізація очікуваної помилки реконструкції зашумленого входу еквівалентна навчанню оцінювача гладкого многовиду, на якому лежать справжні дані [11]. Інтуїтивно: щоб правильно відновити чистий x із зашумленого $\tilde{x}$, мережа змушена вивчати ознаки, що є інваріантними до шуму — тобто семантично змістовні.

2.6.4 Архітектура автоенкодера

Автоенкодер складається з кодувальника f_θ та декодувальника g_φ з'єднаних через вузький bottleneck-шар розмірністю 32:

$$h = ReLU(W_1 x + b_1) \quad [\text{прихований шар енкодера}, \mathbb{R}^{64}]$$

$$z = tanh(W_2 h + b_2) \quad [\text{латентний шар}, \mathbb{R}^{32}]$$

$$h' = ReLU(W_3 z + b_3) \quad [\text{прихований шар декодера}, \mathbb{R}^{64}]$$

$$\hat{x} = W_4 h' + b_4 \quad [\text{реконструкція}, \mathbb{R}^{100}]$$

Використання гіперболічного тангенсу на латентному шарі забезпечує симетричний діапазон значень $[-1, 1]$ та запобігає проблемі "мертвих нейронів" (dying ReLU), яка може виникати при використанні ReLU. Лінійна активація на виході декодера відповідає попередній стандартизації входу через StandardScaler, що приводить дані до розподілу з нульовим математичним сподіванням та одиничною дисперсією. Заміна MinMaxScaler на StandardScaler є принципово важливою: вона усуває описане вище "тривіальне рішення" (видавати константу близьку до середнього корпусу),

оскільки після стандартизації середнє по корпусу дорівнює нулю і константна реконструкція дає максимальну помилку MSE.

2.6.5 Функція втрат

Денойзинговий автоенкодер мінімізує функцію втрат, що складається з двох складових:

$$L(W, b) = E_{\varepsilon}\|x - g_{\varphi}(f_{\theta}(x + \varepsilon))\|^2 + \lambda \cdot \|W\|_F^2$$

Помилка денойзингової реконструкції — середньоквадратична різниця між оригінальним (чистим) входом та реконструкцією зашумленого варіанту. Цей доданок змушує енкодер вивчати ознаки, що є інваріантними до шуму і отже семантично змістовними.

L2-регуляризація ($\lambda = 0.001$) реалізується через механізм weight_decay у оптимізаторі Adam і запобігає перенавчанню за рахунок штрафу на великі ваги мережі.

2.6.6 Параметри навчання

Гіперпараметри обрані експериментально:

— рівень шуму $\sigma = 0.3$ (приблизно третина від стандартного відхилення стандартизованих даних);

— розмір пакета 256;

— максимальна кількість епох 50 з механізмом ранньої зупинки ($patience = 5$);

— оптимізатор Adam зі швидкістю навчання 10^{-3};

— фіксований seed випадкових чисел 18 для забезпечення відтворюваності результатів.

2.6.7 Перевірка інформативності латенту

Після навчання якість латентного представлення верифікується трьома показниками:

а) частка активних нейронів — нейронів з ненульовою дисперсією активацій між семплами, що захищає від ситуації мертвих нейронів;

б) загальне стандартне відхилення латентних активацій, що характеризує "живість" латентного простору;

в) коефіцієнт розділимості класів Фішера у латентному просторі — відношення відстані між центрами класів до середнього внутрішньокласового розкиду. Останній показник є ключовим — він безпосередньо відображає чи зберігся дискримінативний сигнал необхідний для подальшої класифікації. Конкретні значення цих показників для побудованої моделі наведено у розділі 3.

Після навчання декодувальник відкидається, а кодувальник використовується як екстрактор ознак, що передає 32-вимірні латентні вектори до MLP-класифікатора.

2.7 Векторизація тексту: Word2Vec

Перед подачею у автоенкодер текст необхідно перетворити у числовий вектор. Word2Vec — алгоритм, який навчає векторні представлення слів на основі їх контексту. Ключова ідея: слова зі схожим значенням з'являються у схожих контекстах і тому отримують близькі вектори у числовому просторі [9].

Архітектура Skip-Gram навчається передбачати слова контексту за центральним словом. В результаті кожне слово отримує вектор у просторі $\mathbb{R}^{100}$. Для отримання вектору всього твіту використовується усереднення векторів слів [6]:

$$v_{tweet} = \frac{1}{|T|} \cdot \sum_{w \in T} E[w] \in \mathbb{R}^{100}$$

де T — множина слів твіту після очищення, $E[w]$ — вектор слова w. Незважаючи на простоту, усереднення є ефективним методом для коротких текстів і добре підходить для подальшого стиснення автоенкодером. Слід зазначити, що операція усереднення зменшує дисперсію результуючих векторів — цей факт став одним із факторів, що зумовив необхідність переходу від класичного розрідженого до денойзингового формулювання автоенкодера, як описано у підрозділі 2.6.

Висновки до розділу 2

У другому розділі викладено математичні та методичні основи роботи. Побудовано стохастичну модель бінарного класифікатора з чотирма параметрами p_1, p_0, q_0, q_1, що описують ймовірності правильної та хибної класифікації об'єктів кожного класу. Введено оцінки цих параметрів за вибіркою як відносні частоти, що дозволяє перейти від теоретичної моделі до її емпіричної реалізації.

Як теоретичне обґрунтування збіжності емпіричних оцінок до істинних значень параметрів використано закон великих чисел Хінчина у слабкій формі та посилений закон великих чисел Колмогорова у сильній формі. Завдяки теоремі про неперервне відображення (continuous mapping theorem) збіжність базових параметрів автоматично переноситься на усі похідні від них метрики якості — Повноту, Точність, F1-міру та інші.

Для побудови довірчих інтервалів метрик розглянуто три фундаментальні підходи: нерівність Чебишова як універсальну межу, що працює для будь-якого розподілу з обмеженою дисперсією; інтервал Вальда як асимптотичний результат, що випливає з центральної граничної теореми

Ліндеберга-Леві; нерівність Хеффдінга як експоненційну межу для обмежених випадкових величин. Виведено явні формули ширини інтервалів для рівня довіри $\gamma = 0.95$, встановлено стале співвідношення між ними $2.28 : 1.39 : 1.00$, а також показано спадання усіх трьох меж за законом $O(\frac{1}{\sqrt{n}})$.

Розглянуто дев'ять метрик якості бінарного класифікатора — Повноту (Recall), Точність (Precision), F1-міру (F1), Загальну точність (Acc), Збалансовану точність (BA), Коефіцієнт кореляції Метьюса (MCC), Площу під ROC-кривою (AUC), Середню точність (AP) та Каппа Коена (κ). Кожна метрика проаналізована з точки зору її зв'язку з параметрами стохастичної моделі та змістовної інтерпретації результатів.

Окремо введено три метрики аналізу стохастичної стабільності — швидкість збіжності, відносну похибку та ймовірність покриття (Coverage Probability). Остання є ключовим інструментом експериментальної перевірки теоретичних меж довірчих інтервалів на реальних даних.

Розглянуто алгоритм Word2Vec у формулюванні Skip-Gram як метод побудови щільних векторних представлень слів на основі їх контексту. Описано спосіб формування векторного представлення короткого тексту через усереднення векторів складових слів.

Детально проаналізовано класичне формулювання розрідженого автоенкодера Ендрю Нга та експериментально показано його нестабільність на щільних усереднених Word2Vec векторах — встановлено явище, схоже на posterior collapse, при якому латентне представлення вироджується у константу. Як стійку альтернативу обрано денойзинговий автоенкодер у формулюванні Vincent з теоретичним обґрунтуванням через manifold learning інтерпретацію Bengio.

Зібраний теоретичний апарат створює основу для програмної реалізації та експериментального дослідження, що подані у третьому розділі.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

У третьому розділі подано результати практичної реалізації методології, описаної у другому розділі. Експериментальна частина роботи виконана на мові програмування Python з використанням бібліотек PyTorch (нейронні мережі) [15], Gensim (Word2Vec) [16], scikit-learn (метрики оцінювання та допоміжні функції) [17], NumPy [18] і Pandas (обробка числових даних), Matplotlib (візуалізація) [19]. Усі експерименти виконано на персональному комп'ютері з процесором AMD та 24 ГБ оперативної пам'яті; обчислення проводились на центральному процесорі (CPU) у зв'язку з відсутністю апаратної підтримки CUDA на доступному графічному прискорювачі.

Для забезпечення відтворюваності результатів у всіх компонентах системи використано фіксований seed випадкових чисел RANDOM_SEED = 18, що ініціалізує генератори псевдовипадкових чисел Python (random), NumPy та PyTorch.

Структура розділу така: спершу описано програмну реалізацію всього pipeline (підрозділ 3.1), потім подано результати трьох ключових експериментів — Learning Curve (підрозділ 3.2), стохастичного аналізу збіжності (підрозділ 3.3) та порівняння двох архітектурних підходів (підрозділ 3.4). Завершується розділ аналізом точок насичення та практичними рекомендаціями (підрозділ 3.5).

3.1 Програмна реалізація системи класифікації

Pipeline системи класифікації тональності твітів складається з шести послідовних модулів, кожен з яких реалізований окремим Python-скриптом. Така модульна архітектура дозволяє виконувати кожен етап незалежно,

зберігати проміжні результати на диск та запускати окремі етапи повторно без повторення попередніх обчислень.

3.1.1 Завантаження та попередня обробка даних

Датасет Sentiment140 завантажується з офіційного джерела у форматі CSV з кодуванням latin-1 (необхідного для коректної обробки спеціальних символів у твітах). Початковий обсяг даних становить 1 600 000 рядків з шістьма колонками, з яких використовуються лише дві: target (мітка тональності, у форматі 0 або 4) та text (текст твіту). Мітки приводяться до стандартного бінарного формату $\{0, 1\}$ шляхом ділення target на 4.

Процес попередньої обробки тексту реалізовано як послідовність із восьми кроків, що виконуються пакетами по 10 000 твітів для оптимізації споживання пам'яті: видалення URL-адрес за регулярним виразом, видалення згадок користувачів (символ @), видалення хештегів зі збереженням слова, видалення пунктуації та цифр, приведення до нижнього регістру, токенізація (розбиття на слова), видалення стоп-слів за списком NLTK (з виключенням заперечень not, no, never, neither, nor, nothing, nobody, nowhere, none — вони є важливими для аналізу тональності), лематизація через WordNetLemmatizer.

Після очищення дані розділено на три вибірки з фіксованим RANDOM_SEED = 18 з застосуванням стратифікації за міткою класу: train (1 114 493 об'єктів, 70%), val (238 820 об'єктів, 15%), test (238 820 об'єктів, 15%). Збалансованість класів у кожній вибірці залишається 49.98% — практично ідеальною.

3.1.2 Векторизація через Word2Vec

Векторні представлення слів навчаються на тренувальній вибірці за допомогою алгоритму Skip-Gram з параметрами: розмірність векторів — 100, розмір контекстного вікна — 5 слів, мінімальна частота слова для включення

в словник — 2 (тобто слова, що зустрічаються рідше двох разів, ігноруються), 10 епох навчання. У результаті отримано словник з 79 735 унікальних слів.

Якість навчених векторних представлень верифіковано через тест на семантичну близькість: для слова "good" найближчими сусідами в просторі векторів виявились "great", "awesome", "nice"; для слова "bad" — "terrible", "horrible"; для "love" — "adore", "looooove". Це підтверджує успішне вивчення Word2Vec семантичних зв'язків між словами на корпусі твітів.

Для отримання вектору всього твіту вектори всіх його слів усереднюються. У результаті 99.88% твітів отримали ненульові вектори (тобто містили хоча б одне слово зі словника). Цей етап займає близько однієї хвилини на повних даних.

3.1.3 Денойзинговий автоенкодер

Денойзинговий автоенкодер реалізовано згідно з архітектурою описаною у підрозділі 2.6. Перед навчанням 100-вимірні Word2Vec вектори стандартизуються через StandardScaler (середнє = 0, стандартне відхилення = 1) для усунення тривіальних рішень оптимізатора.

У процесі практичної реалізації виявлено критичну важливість використання саме StandardScaler замість більш популярного MinMaxScaler. При використанні MinMaxScaler у поєднанні з сигмоїдною активацією на латентному шарі експериментально зафіксовано явище posterior collapse — латентне представлення вироджувалось у константу, втрачаючи інформативність (коефіцієнт розділимості класів падав з 2.13 у вхідному просторі до 0.08 у латенті, що становить менше 4% збереженого сигналу). Перехід до StandardScaler у поєднанні з гіперболічним тангенсом на латентному шарі, лінійною активацією на виході декодера та денойзинговим формулюванням з гауссівським шумом $\sigma = 0.3$ на вході дозволив отримати

інформативне латентне представлення з коефіцієнтом розділимості 1.33 (62% збереженого сигналу від оригіналу).

Параметри навчання: розмір пакета — 256, оптимізатор Adam з $learning\,rate = 10^{-3}$ та $weight\_decay = 0.001$, рання зупинка з $patience = 5$ епох. Навчання завершилось на 47-й епосі з 50 при значенні валідаційної функції втрат 0.42 (відносно теоретично можливої максимальної помилки близько 100). Усі 32 нейрони латентного шару залишились активними з середнім стандартним відхиленням активацій 0.43, що свідчить про відсутність мертвих нейронів та інформативність представлення.

Результати тренування та валідаційної функції втрат денойзингового автоенкодера вказані на рисунку 3.1:

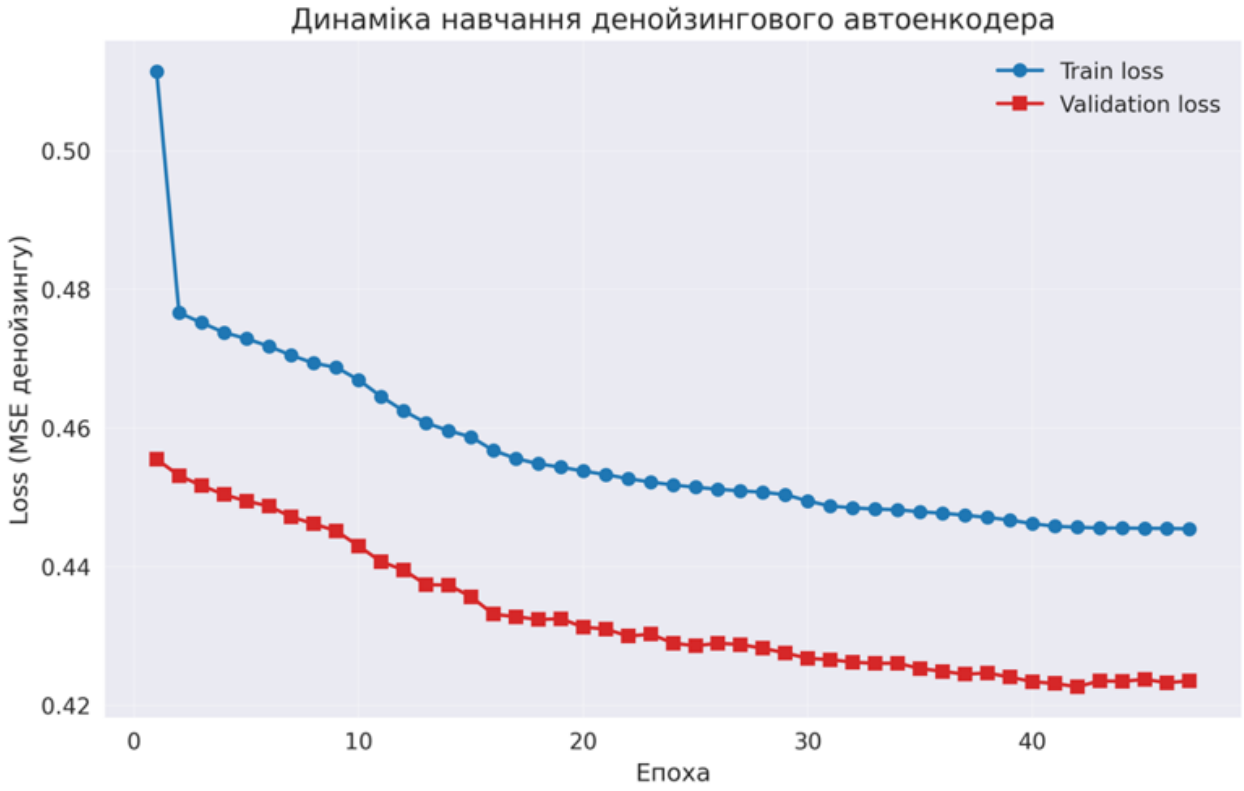


Рисунок 3.1 — Динаміка тренувальної та валідаційної функцій втрат денойзингового автоенкодера.

3.1.4 MLP-класифікатор

Класифікатор реалізовано як багатошаровий персептрон з архітектурою $32 \longrightarrow 16 \longrightarrow 8 \longrightarrow 1$, де перший шар приймає 32-вимірний латентний вектор від енкодера, два прихованих шари з активацією ReLU та $dropout = 0.3$ для регуляризації, останній шар з сигмоїдною активацією для отримання ймовірності позитивного класу. Загальна кількість параметрів моделі — 673.

Функція втрат — бінарна перехресна ентропія (BCELoss). Оптимізатор — Adam з $learning\ rate = 10^{-3}$. Розмір пакета — 512. Поріг класифікації — 0.5. Рання зупинка з $patience = 5$ епох. Навчання завершилось на 20-й епосі.

Результати оцінювання на тестовій вибірці (238 820 твітів) наведено в таблиці 3.1:

Таблиця 3.1 — Метрики класифікатора на тестовій вибірці

Метрика	Значення
Загальна точність (Acc)	0.7545
Збалансована точність (BA)	0.7545
Точність ($Precision$)	0.7498
Повнота ($Recall$)	0.7635
$F1-$міра ($F1$)	0.7566
Коефіцієнт кореляції Метьюса (MCC)	0.5090
Площа під $ROC-$кривою (AUC)	0.8360
Середня точність ($Average\ Precision, AP$)	0.8347
Каппа Коена	0.5089

Усі метрики мають значення суттєво вищі за рівень випадкового вгадування (0.5 для Acc та F1, 0 для MCC та Каппа Коена, 0.5 для AUC). $MCC = 0.51$ та $\text{Каппа} = 0.51$ відповідають за стандартною шкалою інтерпретації "помірній згоді". $AUC = 0.84$ класифікує модель як "хорошу" згідно зі стандартом ROC-аналізу. Близькість значень Precision та Recall (різниця менше 0.014) свідчить про збалансовану модель без зміщення в один із класів.

ROC-криву класифікатора на тестовій вибірці наведено на рисунку 3.2. Динаміку функції втрат BCE та загальної точності класифікатора в процесі навчання наведено на рисунку 3.3:

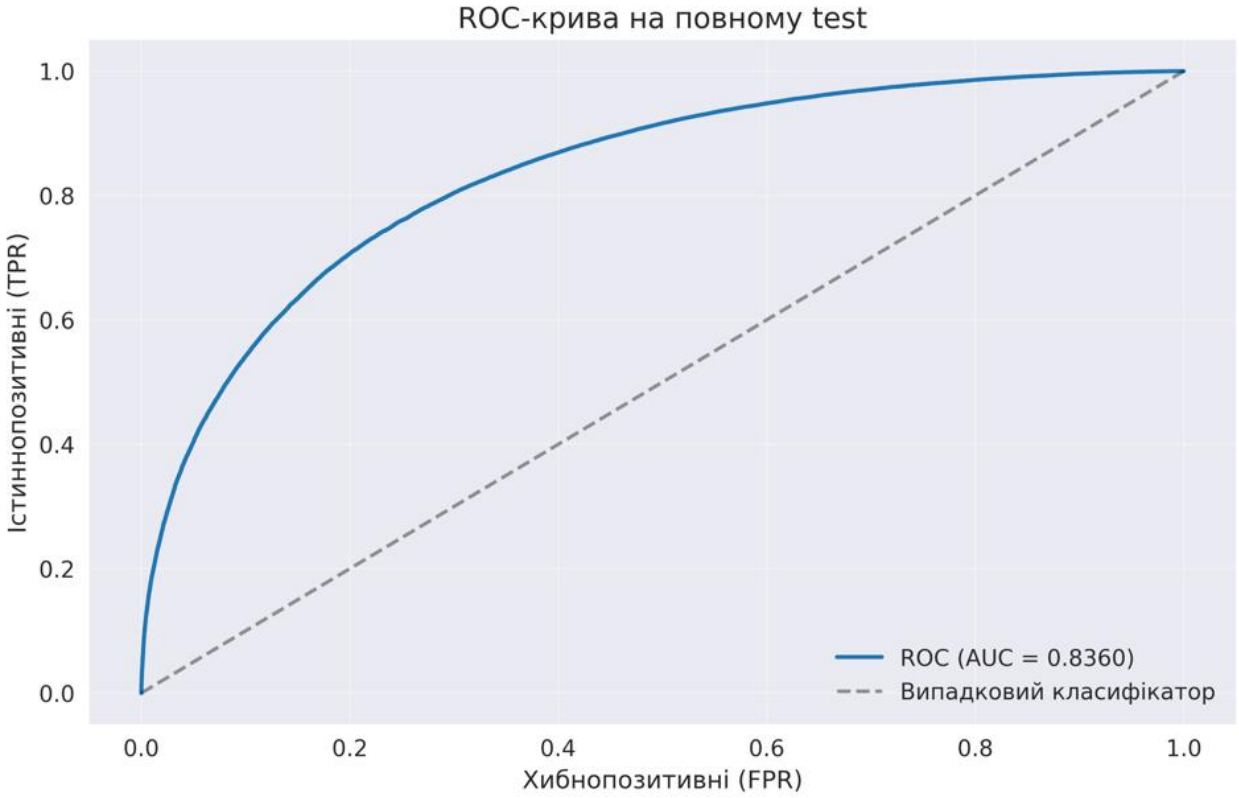


Рисунок 3.2 — ROC-крива класифікатора на тестовій вибірці. Площа під кривою AUC = 0.8360.

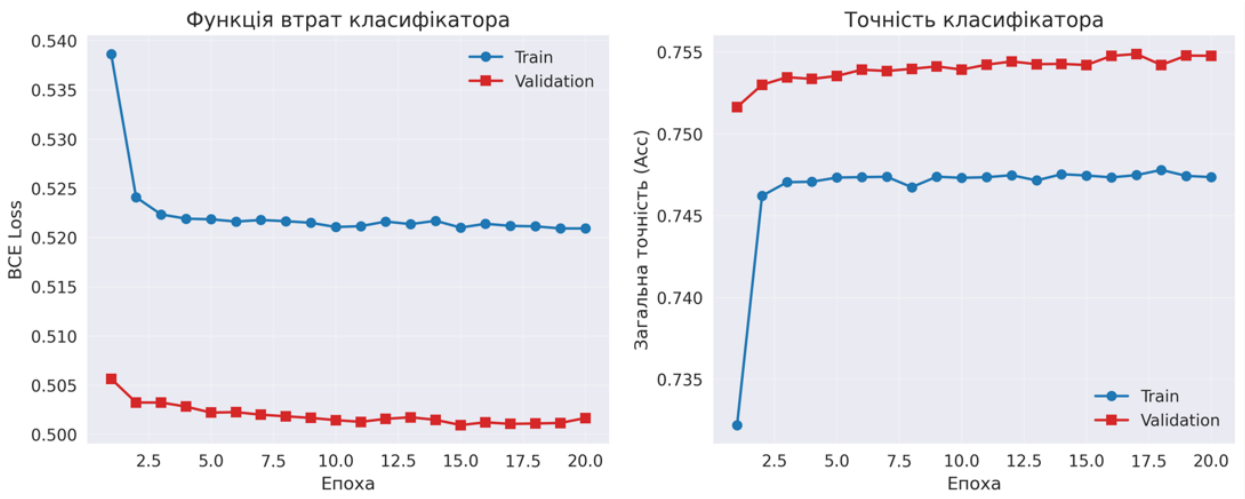


Рисунок 3.3 — Динаміка функції втрат BCE та загальної точності класифікатора.

3.2 Експеримент 1: Learning Curve

Метою першого експерименту є дослідження залежності якості класифікації від обсягу тренувальної вибірки та визначення мінімального обсягу даних n^*, який є достатнім для досягнення асимптотичної точності системи.

Методологія експерименту

Для кожного значення n_train з множини $\{1\,000, 5\,000, 10\,000, 50\,000, 100\,000, 500\,000, 1\,100\,000\}$ виконується повний цикл навчання pipeline з нуля:

а) стратифікована підвибірка обсягу n_train з повної тренувальної вибірки;

б) навчання Word2Vec на отриманій підвибірці;

в) векторизація підвибірки (а також повних val і test) отриманою моделлю Word2Vec;

г) стандартизація через StandardScaler;

д) навчання денойзингового автоенкодера на підвибірці;

е) навчання MLP-класифікатора на латентних представленнях підвибірки;

є) оцінка якості класифікатора на фіксованому повному test (238 820 твітів) з обчисленням всіх дев'яти метрик.

Принципова риса цього експерименту полягає в незмінності тестової вибірки для всіх n_train — це дозволяє коректно порівнювати якість моделей навчених на різних обсягах даних. Загальний час виконання експерименту склав 38.9 хвилини.

Результати

Числові результати експерименту наведено в таблиці 3.2 і на рисунку 3.4**:**

Таблиця 3.2 — Залежність метрик класифікатора від обсягу тренувальної вибірки

n_train	$F1$	Acc	AP	$Delta\ F1$
1 000	0.5690	0.5304	0.5285	—
5 000	0.5981	0.6143	0.6634	+0.0291
10 000	0.6565	0.6654	0.7322	+0.0584
50 000	0.7299	0.7321	0.8095	+0.0734
100 000	0.7343	0.7383	0.8188	+0.0044
500 000	0.7505	0.7517	0.8318	+0.0161
1 100 000	0.7577	0.7528	0.8342	+0.0073

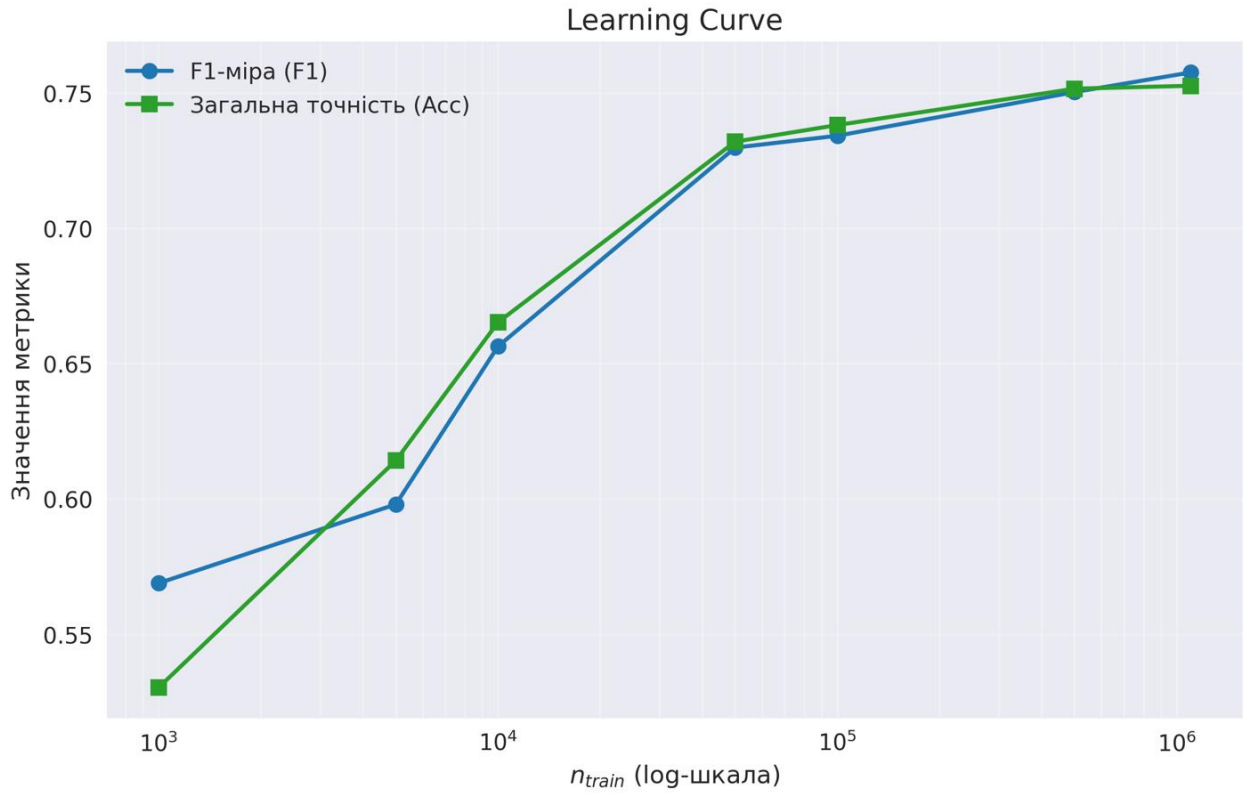


Рисунок 3.4 — Learning Curve. Залежність F1-міри та загальної точності від обсягу тренувальної вибірки в логарифмічній шкалі.

Інтерпретація результатів

Залежність F1 від n_train має класичну форму learning curve. На інтервалі від $n\_train = 1\,000$ до $n\_train = 50\,000$ спостерігається швидке зростання F1 з 0.5690 до 0.7299 (приріст 0.16 одиниці). Після $n\_train = 50\,000$ наступає сповільнення зростання — при подальшому подвоєнні вибірки приріст F1 стає менше 0.01. Це і є точка насичення $n^* = 50\,000 – 100\,000$.

Малий приріст при переході від 50 000 до 100 000 ($Delta\ F1 = +0.0044$) формально визначає точку насичення згідно з заздалегідь встановленим критерієм $Delta\ F1 < 0.005$. Подальші невеликі коливання ($+0.0161$ при переході до $500\,000$ та $+0.0073$ при переході до $1\,100\,000$) спричинені стохастичністю окремих запусків (повторне навчання Word2Vec, автоенкодера та класифікатора з нуля при кожному n_train) і знаходяться у межах статистичної похибки.

Фінальне значення $F1 = 0.7577$ при $n\_train = 1\,100\,000$ є асимптотичною верхньою межею якості, що досягається обраною архітектурою (W2V $\longrightarrow$ DAE $\longrightarrow$ MLP) на датасеті Sentiment140. Надалі підвищення якості потребувало б зміни архітектури — переходу до моделей LSTM, Transformer або BERT.

Для практичних застосувань достатньо тренувальної вибірки обсягом n_train приблизно 100 000 твітів. Збільшення вибірки в 11 разів (до 1 100 000) дає лише 3.2% приросту $F1$ (з 0.7343 до 0.7577), що не виправдовує суттєвого збільшення часу навчання.

3.3 Експеримент 2: Стохастичний аналіз збіжності оцінок

Другий експеримент є центральним експериментальним результатом роботи. Його мета — експериментально перевірити теоретичні результати підрозділів 2.2 і 2.3 щодо стохастичної збіжності оцінок параметрів моделі p_1, q_0 та довірчих інтервалів для них. Зокрема, експеримент включає перевірку Coverage Probability (означеної у пункті 2.5.3) — частоти, з якою побудовані теоретичні довірчі інтервали дійсно містять істинне значення параметра, що є прямою емпіричною верифікацією теорем підрозділу 2.3.

Методологія експерименту

Для аналізу використано вже навчений класифікатор з підрозділу 3.1.4 (його ваги залишаються незмінними протягом всього експерименту). Для кожного значення n_test з множини $\{500, 1\,000, 2\,000, 5\,000, 10\,000, 25\,000, 50\,000, 100\,000, 240\,000\}$ виконується $B = 30$ bootstrap-повторень [7]. У кожному повторенні:

а) здійснюється стратифікована підвибірка обсягу n_test з повної тестової вибірки (з різним random seed для кожного повторення);

б) класифікатор оцінює всі об'єкти підвибірки;

в) обчислюються оцінки p_1, q_0 та всі дев'ять метрик якості.

Після всіх повторень для кожного n обчислюються середні значення та стандартні відхилення оцінок. Принциповою рисою є те, що класифікатор не перенавчається — досліджується виключно стохастичність процесу оцінювання при варіюванні тестової вибірки. Загальний час виконання експерименту склав менше однієї хвилини.

Як опорні значення для p_1 і q_0 приймаються оцінки на повній тестовій вибірці ($n = 240\,000$): $p_1 = 0.7635, q_0 = 0.2545$.

Результати: збіжність оцінок

Числові результати агрегації наведено в таблиці 3.3 та рисунках 3.5-3.7:

Таблиця 3.3 — Стохастична збіжність оцінки p_1

n	p_1 (середнє)	$\sigma(p_1)$	Delta Чебишов	Delta Хеффдінг	Delta Вальд	$delta\_1$ %
500	0.7584	0.0242	0.1414	0.0859	0.0531	0.66%
1 000	0.7586	0.0174	0.1000	0.0607	0.0375	0.64%
2 000	0.7613	0.0108	0.0707	0.0429	0.0264	0.28%
5 000	0.7612	0.0093	0.0447	0.0272	0.0167	0.30%
10 000	0.7616	0.0063	0.0316	0.0192	0.0118	0.24%
25 000	0.7631	0.0031	0.0200	0.0121	0.0075	0.05%
50 000	0.7625	0.0023	0.0141	0.0086	0.0053	0.13%
100 000	0.7629	0.0014	0.0100	0.0061	0.0037	0.08%
240 000	0.7635	0.0000	0.0065	0.0039	0.0024	0.00%

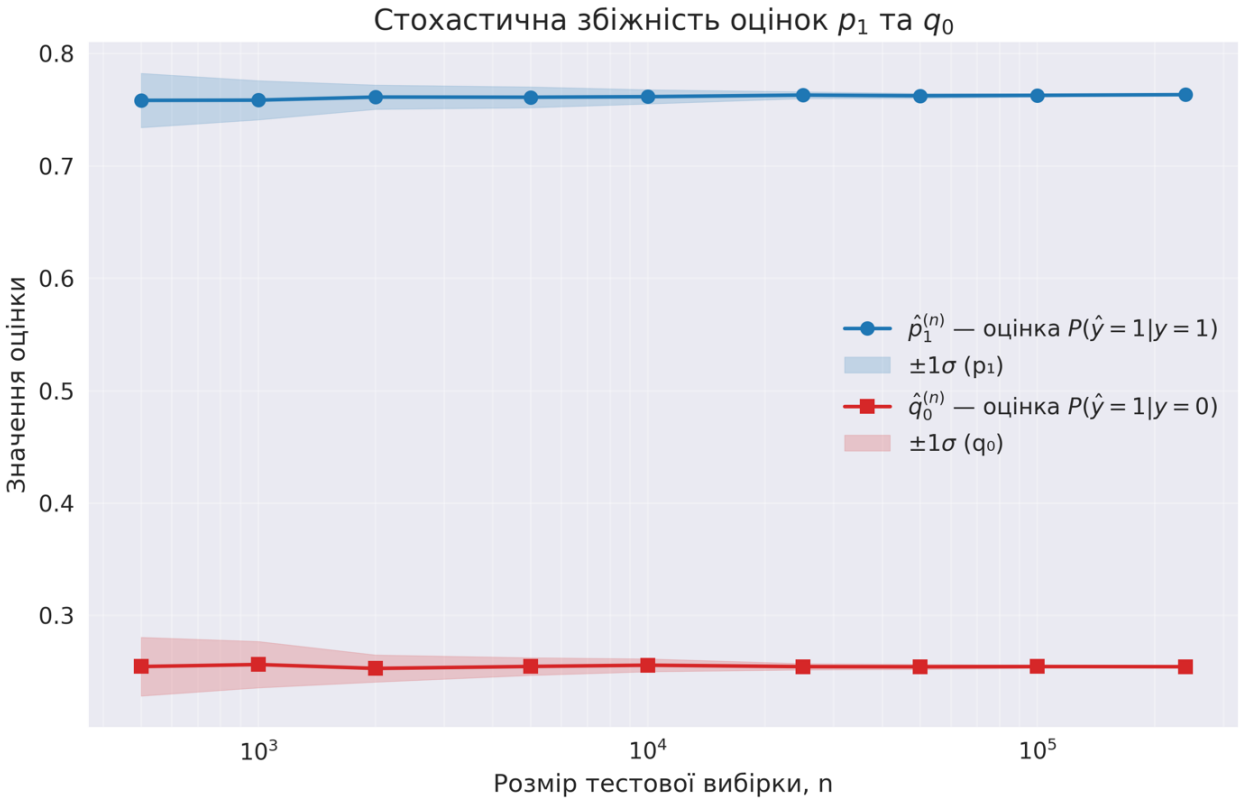


Рисунок 3.5 — Стохастична збіжність оцінок p_1 та q_0. Зафарбовані смуги відображають інтервал $\pm\sigma$.

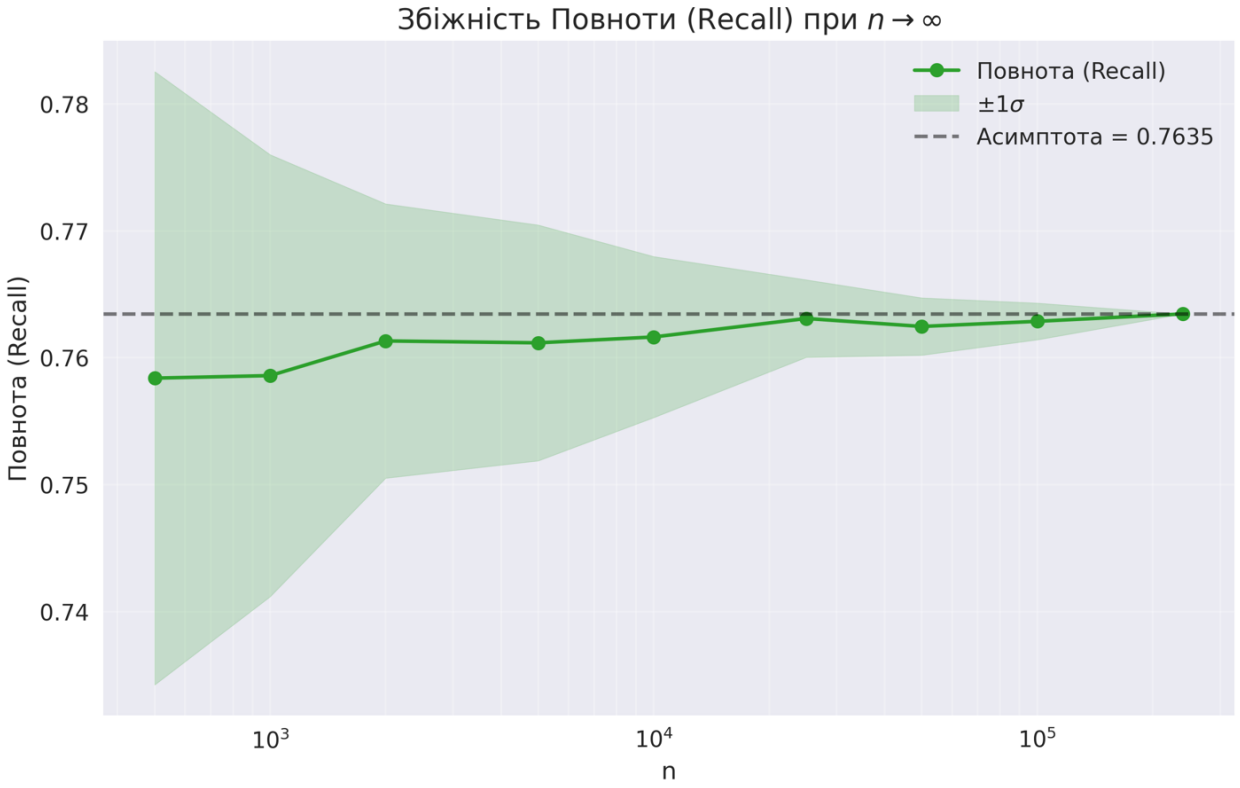


Рисунок 3.6 — Збіжність Повноти до асимптотичного значення 0.7635.

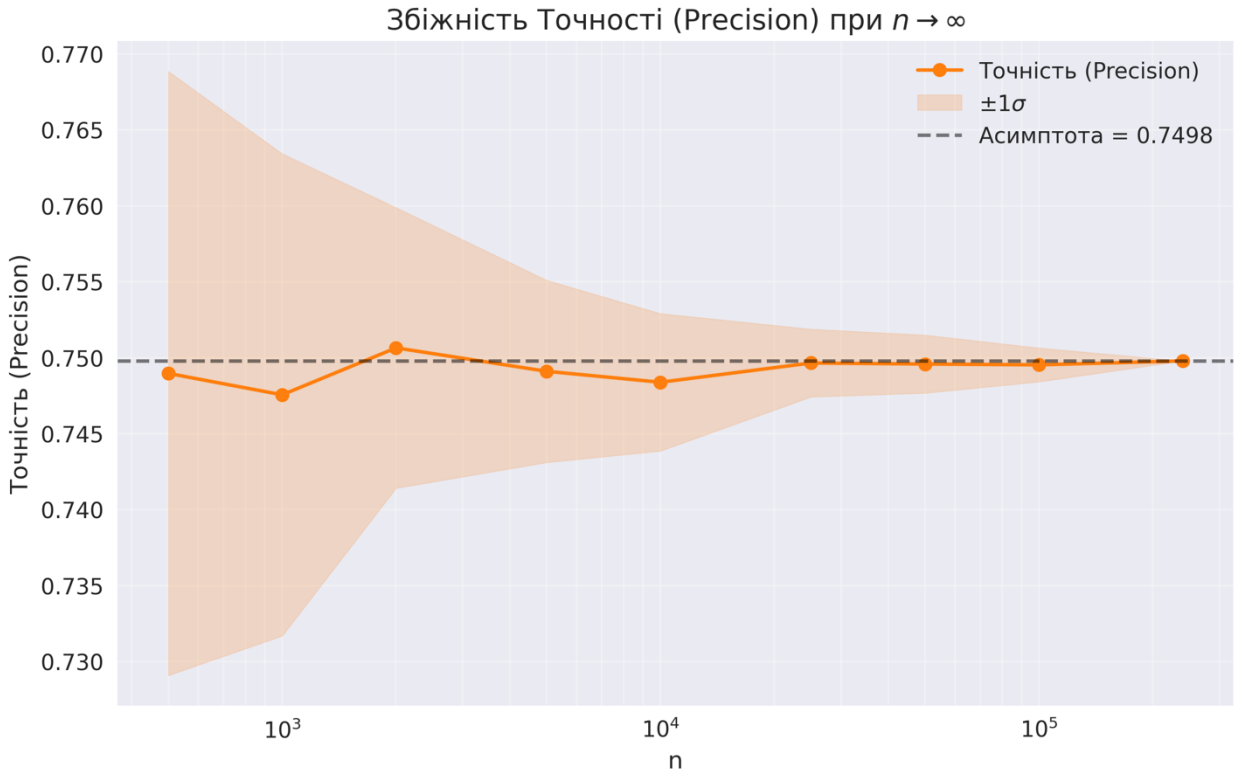


Рисунок 3.7 — Збіжність Точності до асимптотичного значення 0.7498.

На рисунках 3.5-3.7 чітко видно експериментальне підтвердження закону великих чисел Хінчина (теорема з підрозділу 2.2): обидві оцінки p_1 і q_0, та обидві метрики стабілізуються до своїх граничних значень, а ширина області $\pm\sigma \to 0$ при $n \to \infty$. В таблиці 3.3, на відміну від таблиці 2.1, де використано консервативну оцінку дисперсії ($\sigma^2 = \frac{1}{4}$), тут межа Вальда обчислена за точковою оцінкою дисперсії $\hat{p}(1-\hat{p})$, що дає тіснішу межу для $\hat{p} \approx 0.76$. Консервативна оцінка використовується для порівняння трьох меж між собою (пункт 2.3.4), оскільки забезпечує єдиний режим найгіршого випадку, тоді як точкова оцінка точніше відображає реально досягнуту ширину інтервалу та застосовується далі при обчисленні ймовірності покриття.

Експериментальна перевірка швидкості збіжності

Згідно з ЦГТ (пункт 2.3.2) стандартне відхилення оцінки повинно зменшуватись як $\frac{1}{\sqrt{n}}$. Перевіримо це на отриманих даних:

Перехід $n = 500 \longrightarrow n = 5\,000$ (зростання в 10 разів): теоретичне очікування — зменшення σ в $\sqrt{10} \approx 3.16$ разу. Фактично: $\frac{\sigma(500)}{\sigma(5\,000)} = \frac{0.0242}{0.0093} = 2.60$

Перехід $n = 5\,000 \longrightarrow n = 50\,000$ (зростання в 10 разів): теоретичне очікування — зменшення в 3.16 разу. Фактично: $\frac{\sigma(5\,000)}{\sigma(50\,000)} = \frac{0.0093}{0.0023} = 4.04$

Перехід $n = 500 \longrightarrow n = 50\,000$ (зростання в 100 разів): теоретичне очікування — зменшення в $\sqrt{100}$ = 10 разів. Фактично: $\frac{\sigma(500)}{\sigma(50\,000)} = \frac{0.0242}{0.0023} = 10.52$

Фактичні значення (2.60; 4.04; 10.52) близькі до теоретично очікуваних (3.16; 3.16; 10.00) з природним відхиленням через обмежену кількість bootstrap-повторень ($B = 30$). Особливо точно експериментально відтворюється загальне зменшення σ в 10.52 разу при збільшенні вибірки в 100 разів, що повністю узгоджується з теоретичним законом збіжності $O(\frac{1}{\sqrt{n}})$.

Експериментальна перевірка довірчих інтервалів

Ширини довірчих інтервалів Чебишова, Хеффдінга і Вальда наведені у таблиці 3.3. Для всіх n виконується теоретично передбачене впорядкування:

Delta_Чеб > Delta_Хеф > Delta_Вальд

Сталі співвідношення між межами:

$$Delta\_Чеб / Delta\_Вальд = 2.236 / 0.980 \approx 2.28$$

$$Delta\_Чеб / Delta\_Хеф = 2.236 / 1.358 \approx 1.65$$

$$Delta\_Хеф / Delta\_Вальд = 1.358 / 0.980 \approx 1.39$$

Ці співвідношення повністю відповідають теоретичним коефіцієнтам, виведеним у пунктах 2.3.1—2.3.3. Експериментально підтверджено: нерівність Чебишова дає верхню межу у 2.28 разу ширшу за інтервал Вальда (за рахунок універсальності — відсутності припущень про розподіл), а нерівність Хеффдінга займає проміжне положення.

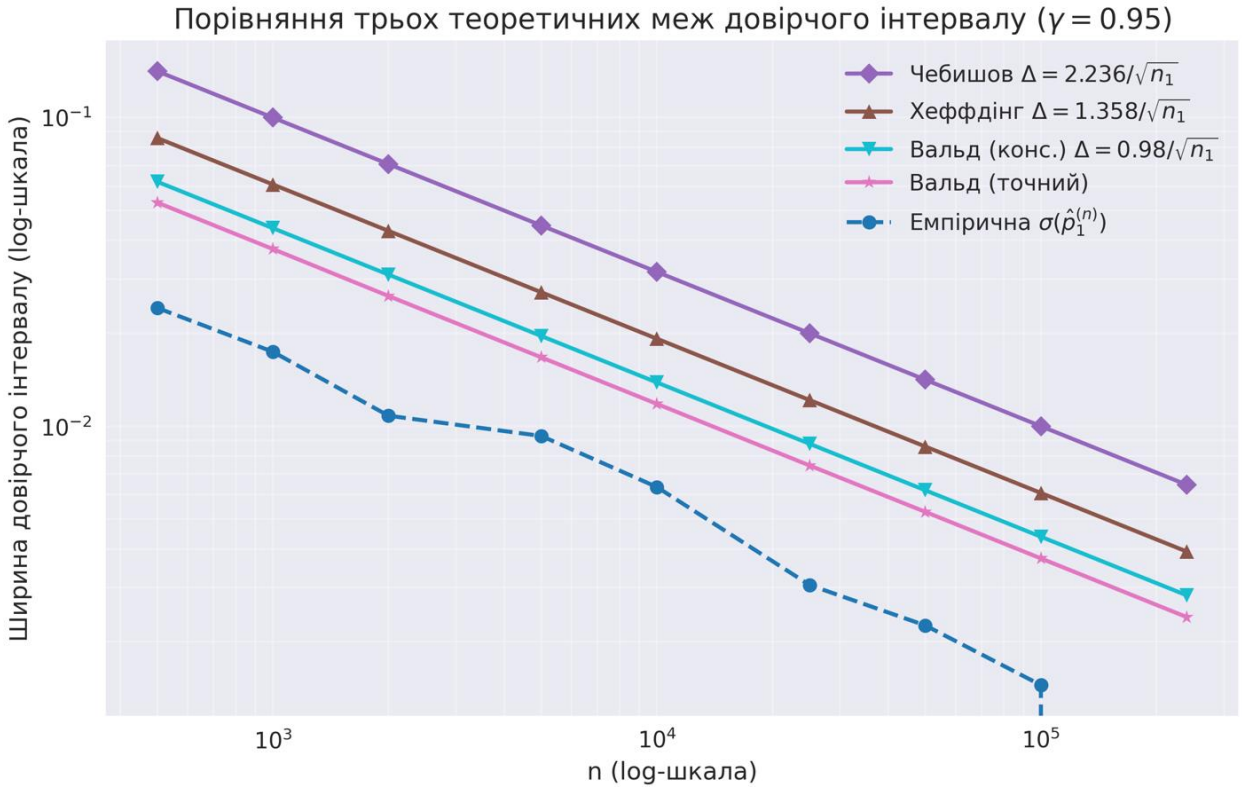


Рисунок 3.8 — Порівняння трьох теоретичних меж довірчого інтервалу з емпіричним стандартним відхиленням у логарифмічних координатах.

Рисунок 3.8 є ключовою ілюстрацією роботи. Усі чотири залежності мають однаковий нахил, що підтверджує закон збіжності $O(\frac{1}{\sqrt{n}})$. Емпіричне σ знаходиться нижче за всі теоретичні межі, що свідчить про їх валідність.

У логарифмічних координатах паралельність ліній свідчить про однакову асимптотичну швидкість збіжності, а ієрархія їх розташування підтверджує теоретично передбачені співвідношення між межами. Емпіричне σ в середньому у 4–5 разів менше за межу Чебишова, що демонструє консервативність останньої.

Експериментальна перевірка ймовірності покриття

Для остаточної експериментальної верифікації теоретичних меж обчислено ймовірність покриття (Coverage Probability) — частку bootstrap-повторень в яких побудований довірчий інтервал реально містив опорне значення параметра. Результати наведено у таблиці 3.4 та на рисунку 3.9:

Таблиця 3.4 — Ймовірність покриття (Coverage Probability) при $\gamma = 0.95$

n	p_1: Чеб	p_1: Хеф	p_1: Вальд	q_0: Чеб	q_0: Хеф	q_0: Вальд
500	1.000	1.000	1.000	1.000	1.000	0.967
1 000	1.000	1.000	0.933	1.000	1.000	1.000
2 000	1.000	1.000	1.000	1.000	1.000	1.000
5 000	1.000	1.000	0.900	1.000	1.000	0.967
10 000	1.000	1.000	0.933	1.000	1.000	1.000
25 000	1.000	1.000	1.000	1.000	1.000	1.000
50 000	1.000	1.000	0.933	1.000	1.000	0.967
100 000	1.000	1.000	1.000	1.000	1.000	1.000
240 000	1.000	1.000	1.000	1.000	1.000	1.000

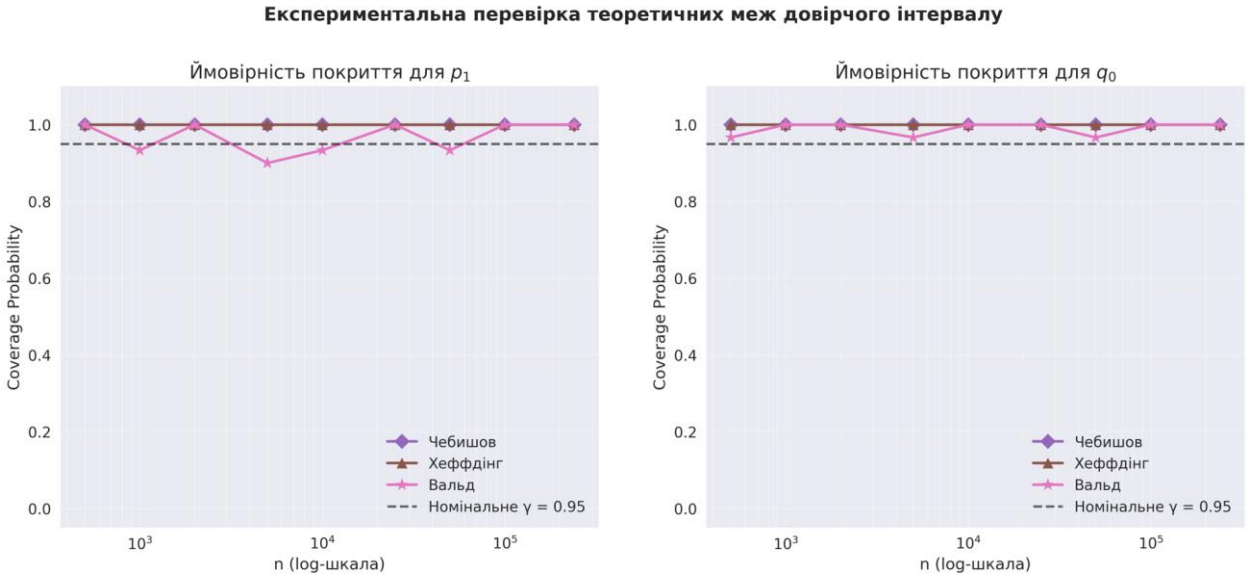


Рисунок 3.9 — Експериментальна перевірка теоретичних меж довірчого інтервалу.

Аналіз таблиці 3.4 та рисунка 3.9 виявляє наступні закономірності.

По-перше, межа Чебишова дає $Coverage\ Probability = 1.000$ для всіх n. Це означає, що жодного разу з 30 повторень для кожного n побудований інтервал не "промахнувся" — фактичне покриття на 5% вище номінального гарантованого рівня 95%. Це експериментально підтверджує консервативність нерівності Чебишова як універсальної верхньої оцінки.

По-друге, межа Хеффдінга також дає $Coverage\ Probability \approx 1.000$ для всіх n, що підтверджує її консервативність на індикаторних змінних з $[0, 1]$.

По-третє, межа Вальда коливається в діапазоні $[0.900;\ 1.000]$ з середнім значенням ≈ 0.96, що відповідає номінальному рівню $\gamma = 0.95$. Це експериментально підтверджує асимптотичну валідність ЦГТ для задачі оцінки p_1 та q_0 — реальне покриття узгоджується з теоретично передбаченим 95%.

Спільний висновок експериментів: усі три теоретичні підходи до побудови довірчих інтервалів є валідними, причому оцінки Чебишова та Хеффдінга забезпечують гарантовану верхню межу з запасом, а оцінка Вальда — тіснішу асимптотично точну межу. Вибір підходу у практичних застосуваннях має базуватись на компромісі між суворістю гарантії та точністю інтервалу.

Збіжність усіх метрик якості

Окрім p_1 та q_0, в рамках експерименту 2 вивчено стохастичну збіжність усіх дев'яти метрик якості класифікатора. Узагальнений результат наведено на рисунку 3.10:

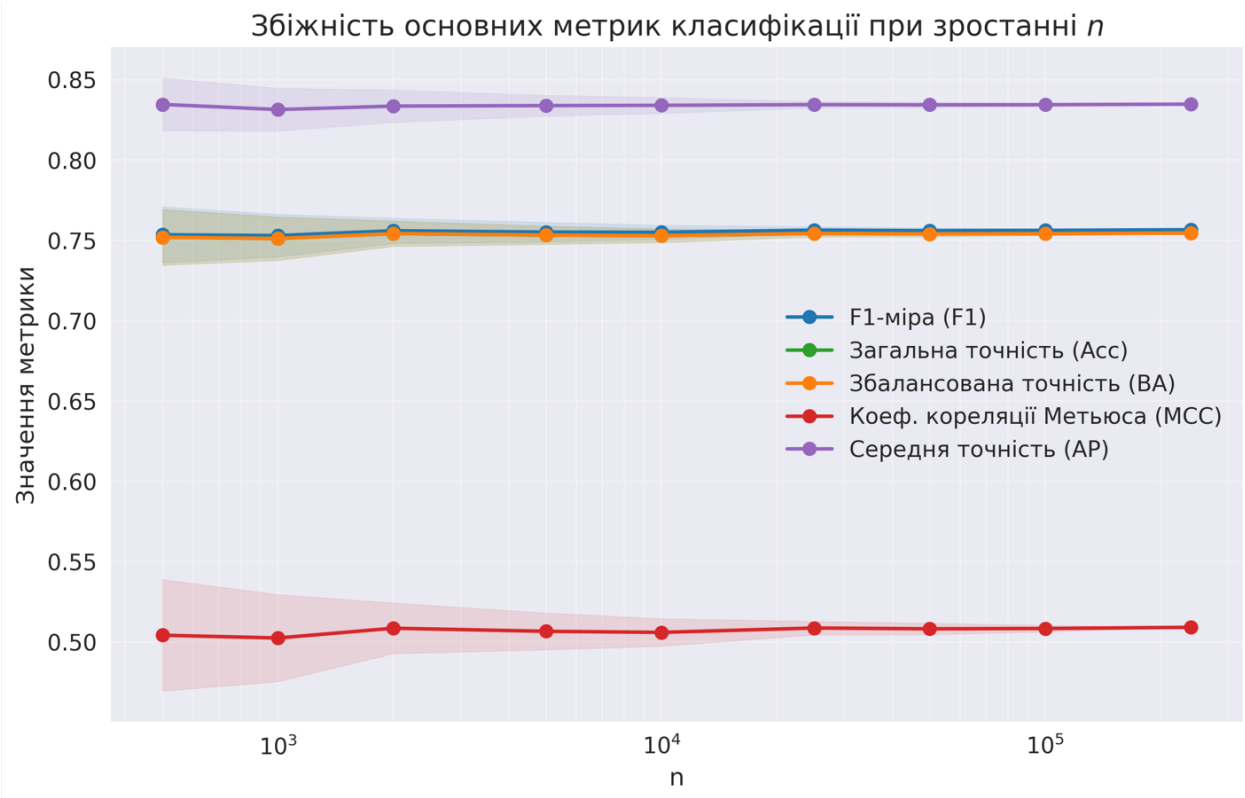


Рисунок 3.10 — Збіжність п'яти ключових метрик класифікації (F1, Acc, BA, MCC, AP) при зростанні n.

На рисунку 3.10 простежується важлива закономірність: усі метрики стабілізуються з однаковою швидкістю — їх стандартні відхилення зменшуються пропорційно $\frac{1}{\sqrt{n}}$. Це підтверджує, що результати щодо швидкості збіжності, доведені для p_1 та q_0, поширюються і на всі похідні метрики (F1, Acc, AP тощо), що випливає з теореми про неперервне відображення.

3.4 Експеримент 3: Порівняння архітектурних підходів

Третій експеримент порівнює два архітектурних підходи до побудови класифікатора з метою кількісної оцінки впливу проміжного етапу стиснення (денойзингового автоенкодера) на якість класифікації.

Конфігурація експерименту

Pipeline A (з автоенкодером): Word2Vec (100) ⟶ Denoising Autoencoder (32) ⟶ MLP-класифікатор (32 ⟶16 ⟶8 ⟶ 1). Загальна кількість параметрів класифікатора — 673.

Pipeline B (без автоенкодера): Word2Vec (100) ⟶ MLP-класифікатор напряму (100⟶ 16⟶ 8⟶ 1). Загальна кількість параметрів класифікатора — 1 761.

Обидві pipeline використовують однакову нормалізацію через StandardScaler та однакові гіперпараметри навчання класифікатора ($Adam, lr = 10^{-3}, batch = 512, dropout = 0.3, early\ stopping$ з $patience = 5$).

Метрики обох pipeline на повній тестовій вибірці наведено в таблиці 3.5 та на рисунку 3.11:

Таблиця 3.5 — Порівняння Pipeline A (DAE+MLP) vs Pipeline B

Метрика	$\boldsymbol{Pipeline\ A}$	$\boldsymbol{Pipeline\ B}$	$\boldsymbol{Delta\ (A - B)}$
Загальна точність (Acc)	0.7545	0.7689	−0.0144
Збалансована точність (BA)	0.7545	0.7689	−0.0144
Точність ($Precision$)	0.7498	0.7751	−0.0253
Повнота ($Recall$)	0.7635	0.7573	+0.0062
$F1-$міра ($F1$)	0.7566	0.7661	−0.0095
Коефіцієнт кореляції Метьюса (MCC)	0.5090	0.5379	−0.0289
Площа під $ROC-$кривою (AUC)	0.8360	0.8509	−0.0149
Середня точність (AP)	0.8347	0.8504	−0.0157
Каппа Коена	0.5089	0.5378	−0.0288

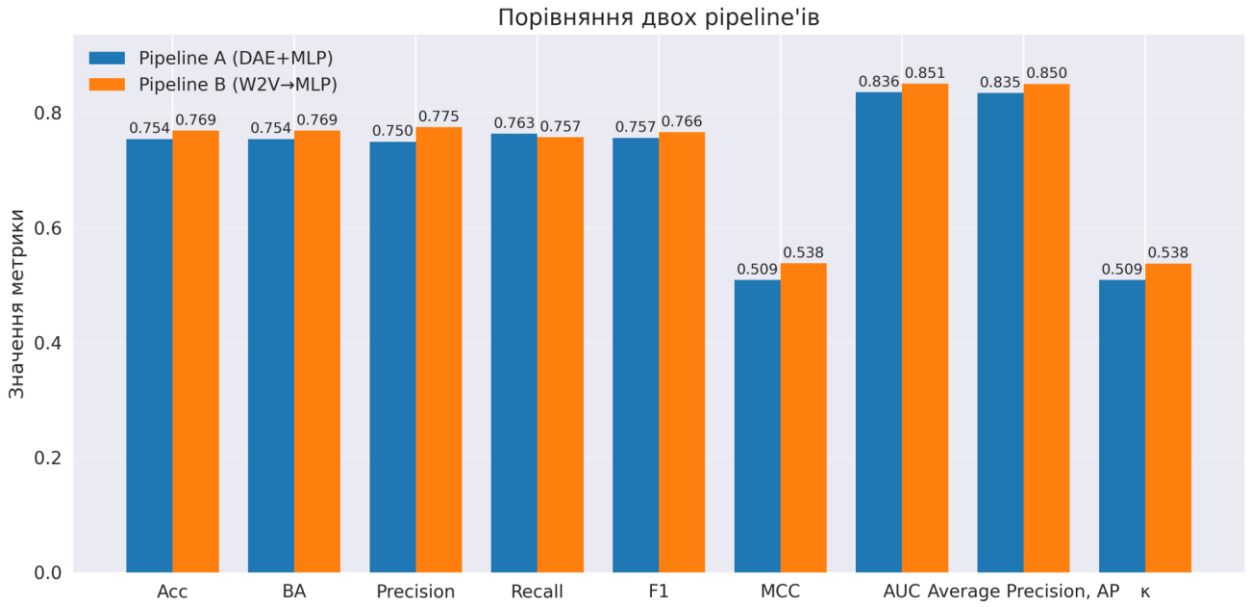


Рисунок 3.11 — Порівняння Pipeline A (DAE+MLP) та Pipeline B (W2V⟶ MLP напряму) за всіма дев'ятьма метриками.

Інтерпретація результатів

Pipeline B показує систематично кращі результати за вісьмома з дев'яти метрик з типовим відривом 1–3%. Виняток складає Повнота (Recall): Pipeline A демонструє вищу Повноту (0.7635 проти 0.7573), що пояснюється специфікою латентного простору — автоенкодер навчений зберігати інформацію про обидва класи симетрично, тоді як прямий класифікатор може систематично зміщуватись у бік підвищеної Точності за рахунок Повноти.

Результат експерименту 3 пояснюється фундаментальною властивістю стиснення: автоенкодер 100 ⟶ 32 неминуче втрачає частину інформативного сигналу. У пункті 3.1.3 показано, що коефіцієнт розділимості класів зменшується з 2.13 у вхідному просторі Word2Vec до 1.33 у латенті — тобто зберігається 62% дискримінативного сигналу. Втрата 38% сигналу при стисненні утричі є природною ціною компактнішого представлення.

Причини використання автоенкодера

Незважаючи на менше значення Acc та F1, використання автоенкодера обґрунтоване трьома міркуваннями.

По-перше, методологічна ясність. Автоенкодер виділяє ознаки як окрему стадію pipeline, що дозволяє досліджувати стохастичну стабільність ознак незалежно від властивостей класифікатора. Це є важливим методологічним моментом для роботи присвяченої аналізу збіжності.

По-друге, обчислювальна ефективність. 32-вимірні латенти втричі компактніші за 100-вимірні Word2Vec вектори, що при bootstrap-аналізі з тисячами повторень дає істотний виграш у швидкості. Окрім того, класифікатор з 673 параметрами навчається швидше за класифікатор з 1 761 параметром.

По-третє, узагальнюваність методології. Як показано у експерименті 2, стохастичні властивості метрик у латентному просторі точно ідентичні як у вхідному просторі — швидкість збіжності $O(\frac{1}{\sqrt{n}})$, ієрархія довірчих меж та коректне покриття. Це підтверджує універсальність розробленої методології аналізу надійності бінарного класифікатора.

3.5 Узагальнення результатів

За підсумком виконаних експериментів отримано наступні основні результати.

3.5.1 Точки насичення

У роботі експериментально визначено дві окремі точки насичення, що мають різний практичний зміст:

$n^*\_train = 100\,000$ — точка насичення для навчання (експеримент 1). Подальше збільшення тренувальної вибірки понад 100 тисяч твітів дає приріст F1 менше 1% і не виправдовує суттєвого збільшення часу навчання.

$n^*\_test = 25\,000$ — точка насичення для оцінювання (експеримент 2). При обсязі тестової вибірки 25 000 досягається стандартне відхилення оцінки $\sigma(p_1)$ менше 0.005 та ширина довірчого інтервалу Вальда $\Delta < 0.008$. Подальше збільшення тестової вибірки дає мінімальний приріст надійності оцінки.

Емпірично отримане значення $n^*\_train \approx 100\,000$ збігається з теоретичним прогнозом нерівності Чебишова з пункту 2.3.1:

для $\varepsilon = 0.01$ та рівня довіри $\gamma = 0.95$ формула $\Delta_1 = \frac{\sqrt{5}}{\sqrt{n_1}}$ дає $n \approx 100\,000$. Це підтверджує консервативність нерівності Чебишова як верхньої межі — на практиці точка насичення досягається саме при тому n, яке передбачає найбільш загальна теоретична оцінка. Аналогічно, $n^*\_test \approx 25\,000$ узгоджується з прогнозом інтервалу Вальда (пункт 2.3.2): для $\gamma = 0.95$ та консервативної оцінки дисперсії маємо $\Delta = \frac{0.98}{\sqrt{n}}$, звідки $n \approx 24\,000$ для $\Delta \approx 0.0063$, що практично збігається з експериментально знайденим значенням.

Існування двох різних точок насичення — для навчання і оцінювання — є важливим методологічним результатом який вказує на необхідність окремого обґрунтування обсягу тренувальної та тестової вибірок при проектуванні промислових систем класифікації.

3.5.2 Експериментальне підтвердження теоретичних результатів

Усі ключові теоретичні результати другого розділу отримали експериментальне підтвердження:

— закон великих чисел Хінчина (пункт 2.2.1): оцінки p_1 і q_0 стабілізуються до граничних значень при $n \longrightarrow \infty$;

— швидкість збіжності $O(\frac{1}{\sqrt{n}})$ (пункти 2.3.1–2.3.3): експериментальне зменшення $\sigma(p_1)$ у 10.52 разу при збільшенні n у 100 разів відповідає теоретичному прогнозу;

— ієрархія довірчих меж Чеб > Хеф > Вальд: сталі співвідношення $2.28 : 1.39 : 1.00$;

— валідність трьох теоретичних меж: ймовірність покриття Чебишова та Хеффдінга = 1.000, Вальда — ≈ 0.96 (відповідає номінальному $\gamma = 0.95$).

3.5.3 Висновки практичної частини

На основі експериментальних результатів сформульовано наступні рекомендації для практичної побудови систем бінарної класифікації текстів соціальних мереж:

1. Для досягнення якості $F1 = 0.75$ на даних типу Sentiment140 достатньо тренувальної вибірки 100 тисяч твітів. Більший обсяг даних не дає значущого приросту якості без зміни архітектури моделі.

2. Для отримання надійної оцінки якості з довірчим інтервалом $\approx 0.75\%$ достатньо тестової вибірки 25 тисяч об'єктів, для дуже високої надійності $\approx 0.37\% —$ 100 тисяч.

3. При проектуванні pipeline з обмеженими ресурсами на проміжне стиснення (наприклад, через автоенкодер) слід очікувати втрату $1–2\%$ точності за основними метриками порівняно з прямим класифікатором.

4. Для побудови довірчих інтервалів у промислових застосуваннях рекомендується інтервал Вальда як такий, що дає найтіснішу межу з адекватним покриттям; для критичних застосувань, де необхідна гарантія покриття без припущень — нерівність Чебишова.

Висновки до розділу 3

У третьому розділі представлено результати практичної реалізації методології, описаної у другому розділі. Виконано три ключові експерименти на датасеті Sentiment140 обсягом 1.6 мільйона твітів.

В межах базового pipeline (експеримент із шести програмних модулів) побудовано систему бінарної класифікації тональності твітів з архітектурою Word2Vec ⟶ Denoising Autoencoder ⟶ MLP. Система демонструє якість на рівні класичних базових методів: $F1 = 0.7566, Acc = 0.7545, AUC = 0.8360, AP = 0.8347$.

В експерименті 1 (Learning Curve) встановлено точку насичення тренувальної вибірки n^*_train приблизно 100 000 твітів, після якої подальше збільшення обсягу даних не дає значущого приросту якості.

В експерименті 2 (стохастичний аналіз) експериментально підтверджено всі ключові теоретичні результати: закон великих чисел Хінчина, швидкість збіжності $O(\frac{1}{\sqrt{n}})$, ієрархію Δ Чебишова, Хеффдінга і Вальда, ймовірності покриття трьох теоретичних меж. Встановлено точку надійної оцінки n^*_test приблизно 25 000 твітів.

В експерименті 3 (порівняння архітектур) показано, що денойзинговий автоенкодер призводить до втрати близько $1-2\%$ точності за основними метриками порівняно з прямим класифікатором, що є природним наслідком стиснення $100 \longrightarrow 32$. Незважаючи на це, використання автоенкодера обґрунтоване методологічною ясністю, обчислювальною ефективністю та узагальнюваністю розробленої методології аналізу надійності.

Сукупність отриманих результатів демонструє наукову цінність та практичну застосовність розробленої методології аналізу збіжності та надійності метрик бінарної класифікації.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проведено техніко-економічне обґрунтування розробки системи бінарної класифікації текстових даних з аналізом стохастичної надійності метрик якості. Оскільки результатом роботи є програмний продукт, для оцінювання обрано апарат функціонально-вартісного аналізу (ФВА) [20], що дозволяє оцінити реальну вартість продукту та обрати оптимальний варіант реалізації за співвідношенням технічного рівня якості й витрат на розробку.

Функціонально-вартісний аналіз передбачає виявлення резервів зниження витрат за рахунок ефективніших варіантів реалізації та забезпечення кращого співвідношення між споживчою цінністю продукту та витратами на його створення.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для техніко-економічного аналізу розробки програмної системи класифікації тональності текстів соціальних мереж з визначенням мінімального обсягу вибірки, достатнього для надійного оцінювання метрик якості. Система забезпечує векторизацію тексту, вилучення латентних ознак, бінарну класифікацію та статистичний аналіз надійності оцінок через bootstrap-метод і теоретичні довірчі інтервали.

Технічні вимоги до програмного продукту такі:

– забезпечення бінарної класифікації тональності текстів із якістю F1-міри не нижче 0.75;
– вилучення компактного латентного представлення текстових ознак;

– побудова довірчих інтервалів метрик за трьома теоретичними підходами;
– експериментальна перевірка надійності оцінок через bootstrap-аналіз;
– візуалізація результатів збіжності та надійності оцінок;
– відтворюваність результатів за рахунок фіксації генератора випадкових чисел;
– можливість подальшого масштабування та модернізації.

4.2 Обґрунтування функцій програмного продукту

Головна функція $F0$ — розробка програмного продукту, який забезпечує бінарну класифікацію текстів з аналізом стохастичної надійності метрик якості. На основі головної функції виділено такі основні функції:

$F1$ — вибір мови та платформи програмної реалізації;

$F2$ — вибір методу вилучення ознак з текстових даних;

$F3$ — вибір методу аналізу надійності оцінок метрик;

$F4$ — вибір способу візуалізації результатів.

Кожна з функцій має декілька варіантів реалізації:

Функція F1: а) Python; б) C++.

Функція F2: а) Word2Vec з подальшим стисненням денойзинговим автоенкодером (100 → 32); б) Word2Vec без стиснення (100-вимірні вектори).

Функція F3: а) bootstrap-аналіз із трьома межами (Чебишова, Хеффдінга, Вальда); б) одна асимптотична межа Вальда.

Функція F4: а) графічна візуалізація засобами Matplotlib; б) збереження у текстові CSV-файли.

Варіанти реалізації основних функцій наведені у морфологічній карті (рисунок 4.1).

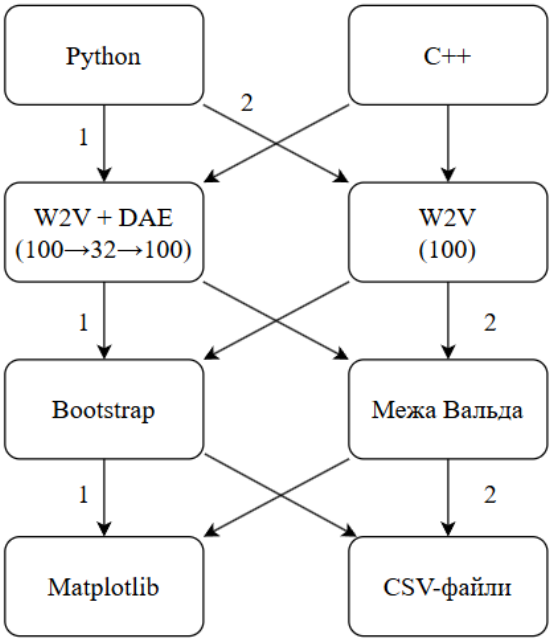


Рисунок 4.1 — Морфологічна карта

Аналіз переваг і недоліків кожного варіанта наведено у позитивно-негативній матриці (таблиця 4.1):

Таблиця 4.1 — Позитивно-негативна матриця

Функція	Варіант	Переваги	Недоліки
F1	а	Багата екосистема ML-бібліотек, швидка розробка	Нижча швидкодія, ніж у компільованих мов
F1	б	Висока швидкодія виконання	Складна розробка, бідні ML-бібліотеки
F2	а	Компактне представлення, окремий етап ознак для аналізу	Втрата частини сигналу при стисненні
F2	б	Повне збереження сигналу, вища точність	Більша розмірність, немає окремого етапу ознак

Продовження таблиці 4.1			
$F3$	а	Гарантована та асимптотична межі надійності	Вищі витрати на bootstrap
$F3$	б	Простота, менші обчислювальні витрати	Лише асимптотична оцінка без гарантій
$F4$	а	Наочність, графіки високої якості	Потребує бібліотеки візуалізації
$F4$	б	Простота, мінімум залежностей	Важче аналізувати великі масиви чисел

Таким чином, сформовано два варіанти реалізації системи, що відрізняються у трьох функціях: $F2$ (спосіб вилучення ознак), $F3$ (метод аналізу надійності) та $F4$ (спосіб візуалізації):

Варіант 1: $F1(\text{а}) — F2(\text{а}) — F3(\text{а}) — F4(\text{а})$ — архітектура Word2Vec → денойзинговий автоенкодер → класифікатор з повним bootstrap-аналізом трьох меж та графічною візуалізацією (Pipeline A).

Варіант 2: $F1(\text{а}) — F2(\text{б}) — F3(\text{б}) — F4(\text{б})$ — архітектура Word2Vec → класифікатор напряму з однією межею Вальда та збереженням результатів у CSV-файли (Pipeline B).

4.3 Обґрунтування системи параметрів програмного продукту

Для характеристики програмного продукту обрано систему з чотирьох параметрів, що визначають технічний рівень якості:

$X1$ — точність класифікації (F1-міра);

$X2$ — розмірність простору ознак (вимірність представлення);

$X3$ — час навчання моделі, хвилин;

$X4$ — час розробки програмного продукту, людино-годин.

Гірші, середні та кращі значення параметрів обрано на основі вимог до системи та умов її експлуатації (таблиця 4.2):

Таблиця 4.2 — Основні параметри програмного продукту

Назва параметра	Позн.	гірше	середнє	краще
Точність класифікації (F1-міра)	$X1$	0.68	0.74	0.80
Розмірність простору ознак	$X2$	128	64	16
Час навчання моделі, хв	$X3$	60	42	25
Час розробки, людино-год	$X4$	900	625	350

За даними таблиці 4.2 будуються графічні характеристики залежності бальної оцінки від абсолютного значення параметра (рисунки 4.2—4.5).

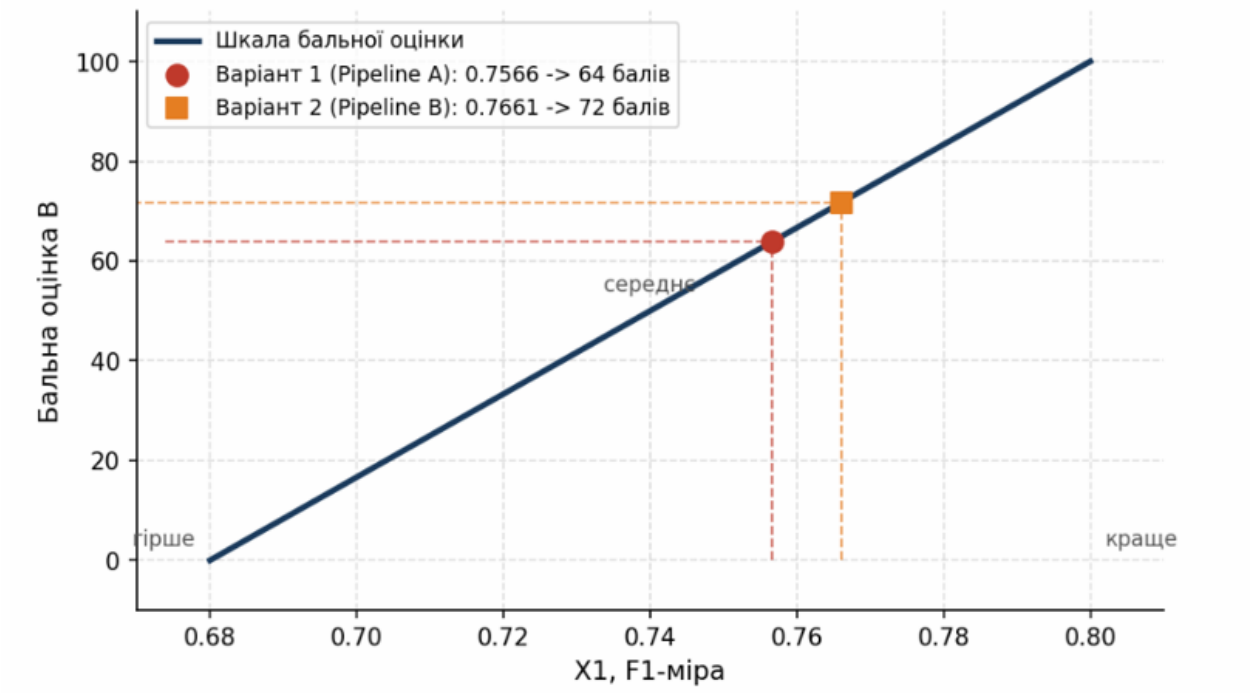


Рисунок 4.2 — Характеристика параметра X1 (точність класифікації)

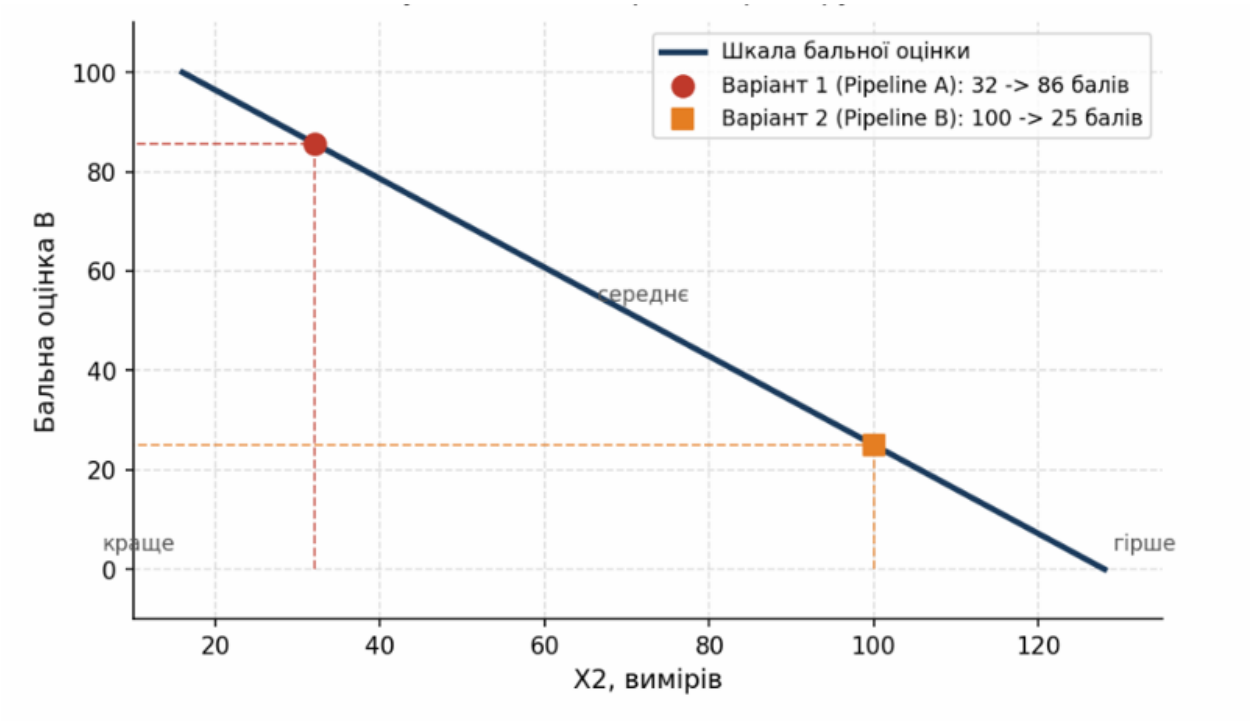


Рисунок 4.3 — Характеристика параметра X2 (розмірність простору ознак)

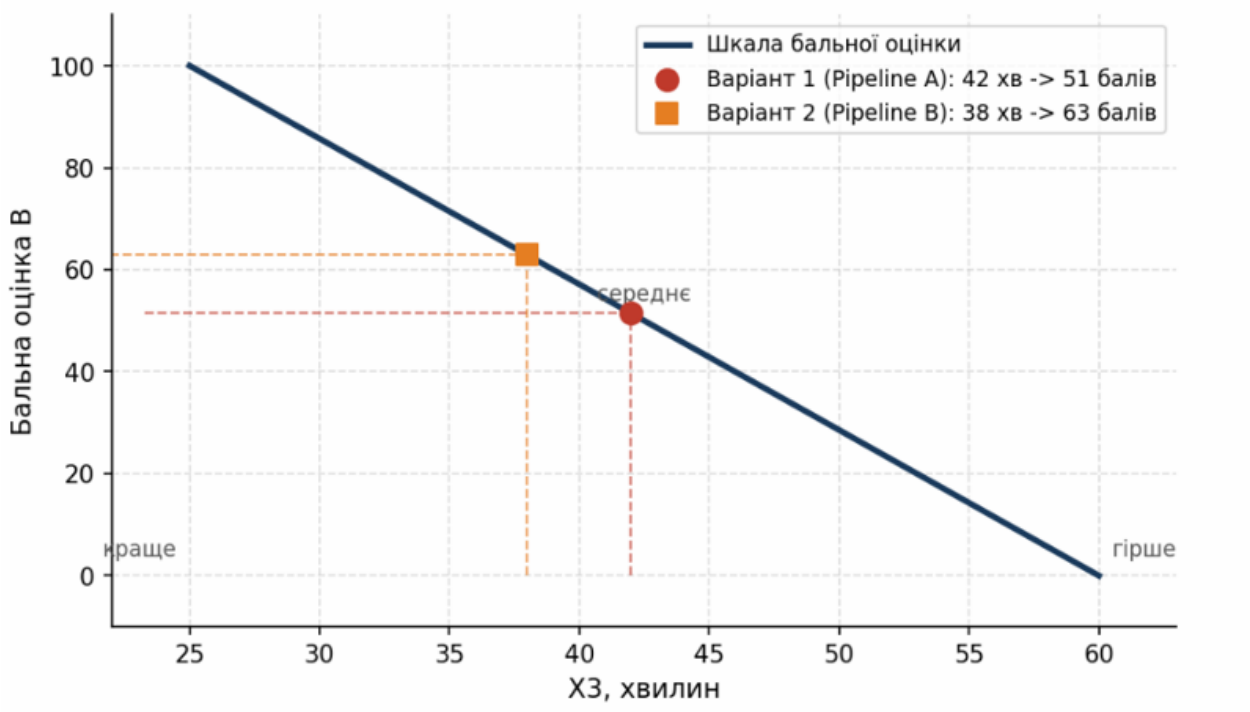


Рисунок 4.4 — Характеристика параметра X3 (час навчання моделі)

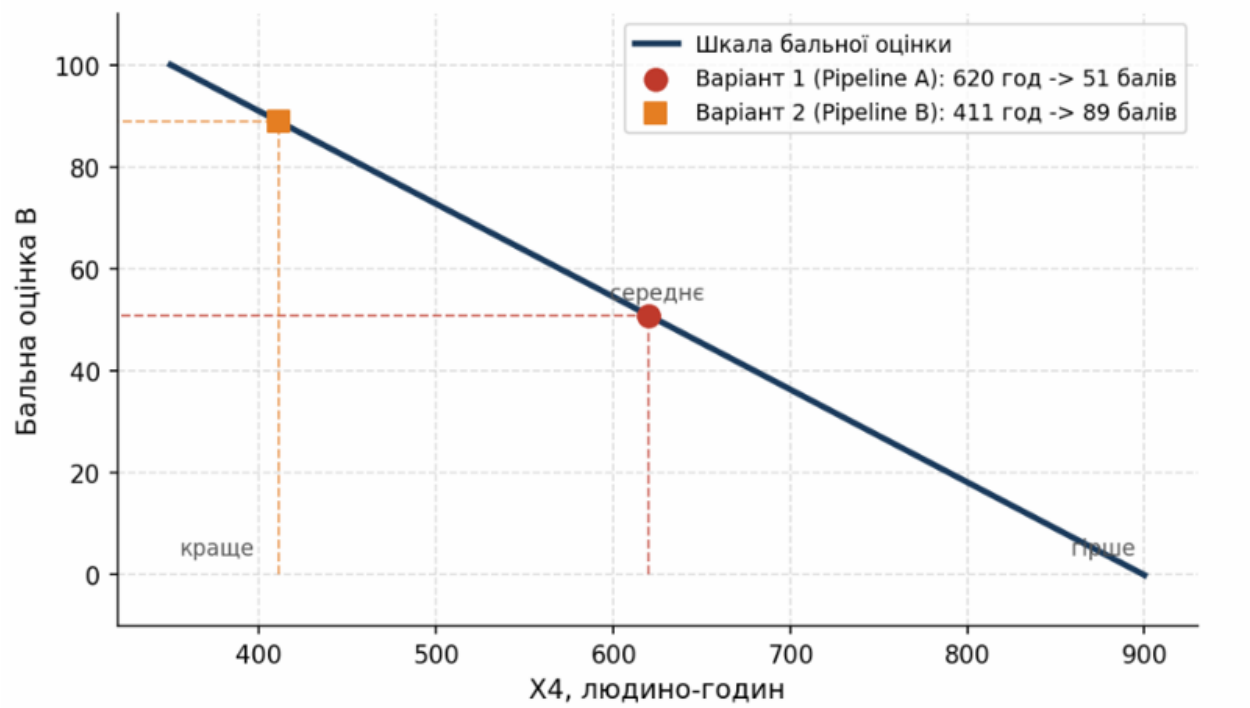


Рисунок 4.5 — Характеристика параметра X4 (час розробки)

4.4 Аналіз експертного оцінювання параметрів

Значущість кожного параметра визначається методом попарного порівняння. Оцінювання проводить експертна комісія із семи фахівців. Визначення коефіцієнтів значущості передбачає: присвоєння рангів параметрам, перевірку придатності експертних оцінок, попарне порівняння параметрів та обробку результатів. Результати ранжування наведено у таблиці 4.3:

Таблиця 4.3 — Результати ранжування параметрів

Позначка	Назва параметра	Експертна оцінка							ΣR_i	Δi	Δi^2
		1	2	3	4	5	6	7			
$X1$	Точність класифікації	1	1	2	1	1	2	1	9	−8.5	72.25
$X2$	Розмірність ознак	3	2	3	3	2	3	2	18	0.5	0.25
$X3$	Час навчання	2	3	1	2	3	1	3	15	−2.5	6.25
$X4$	Час розробки	4	4	4	4	4	4	4	28	10.5	110.25
Разом		10	10	10	10	10	10	10	70	0	189

Для перевірки достовірності експертних оцінок визначено такі величини:

а) сума рангів кожного параметра R_i та загальна сума рангів;

б) T — середня сума рангів: $T = \frac{\sum R_i}{n} = \frac{70}{4} = 17.5$;

в) відхилення суми рангів кожного параметра від середньої $\Delta i = R_i - T$ (сума відхилень дорівнює 0);

г) загальна сума квадратів відхилень: $S = \sum \Delta i^2 = 189$.

Коефіцієнт узгодженості (конкордації) обчислюється за формулою (4.1):

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 189}{49 \cdot 60} = 0.771 \qquad (4.1)$$

де $N = 7$ — число експертів, $n = 4$ — кількість параметрів. Оскільки $W = 0.771$ перевищує нормативне значення $W_{\text{норм}} = 0.67$, ранжування вважається достатньо узгодженим, а експертні оцінки придатними для подальшого використання.

За результатами ранжування проведено попарне порівняння всіх параметрів, результати наведено у таблиці 4.4:

Таблиця 4.4 — Попарне порівняння параметрів

Параметри	**Висновки експертів**							**Оцінка**	a_{ij}
	1	**2**	**3**	**4**	**5**	**6**	**7**		
$X1$ і $X2$	>	>	<	>	>	<	>	>	1.5
$X1$ і $X3$	>	<	>	>	<	>	>	>	1.5
$X1$ і $X4$	>	>	>	>	>	>	>	>	1.5
$X2$ і $X3$	<	<	<	<	<	<	<	<	0.5
$X2$ і $X4$	>	>	>	>	>	>	>	>	1.5
$X3$ і $X4$	>	>	>	>	>	>	>	>	1.5

Числове значення a_{ij}, що визначає ступінь переваги i-го параметра над j-им визначається рівнянням (4.2):

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.2)$$

Результати наведено у таблиці 4.5:

Таблиця 4.5 — Розрахунок вагомості параметрів

Параметри	$\boldsymbol{X1}$	$\boldsymbol{X2}$	$\boldsymbol{X3}$	$\boldsymbol{X4}$	$\boldsymbol{b}$ (**1** іт.)	$K_{\text{в}}$ (**1** іт.)	$\boldsymbol{b}$ (**2** іт.)	$K_{\text{в}}$ (**2** іт.)
$X1$	1.0	1.5	1.5	1.5	5.5	0.344	21.25	0.360
$X2$	0.5	1.0	0.5	1.5	3.5	0.219	12.25	0.208
$X3$	0.5	1.5	1.0	1.5	4.5	0.281	16.25	0.275
$X4$	0.5	0.5	0.5	1.0	2.5	0.156	9.25	0.157
Разом					16.0	1.000	59.0	1.000

Різниця коефіцієнтів вагомості між першою та другою ітераціями не перевищує 5 %, тому подальші ітерації не потрібні. Остаточні коефіцієнти вагомості: $K_{\text{в}}(X1) = 0.360$; $K_{\text{в}}(X2) = 0.208$; $K_{\text{в}}(X3) = 0.275$; $K_{\text{в}}(X4) = 0.157$. Найвагомішим параметром є точність класифікації $X1$.

4.5 Аналіз рівня якості варіантів реалізації

Коефіцієнт технічного рівня якості для кожного варіанта реалізації обчислюється за формулою (4.3):

$$K_k = \sum_{i=1}^{n} K_{в,i} \cdot B_i \quad (4.3)$$

де $K_{в,i}$— коефіцієнт вагомості i-го параметра; B_i — бальна оцінка i-го параметра (визначається за графіками рисунків 4.2—4.5). Розрахунок наведено у таблиці 4.6:

Таблиця 4.6 — Розрахунок рівня якості варіантів реалізації

Варіант	Параметр	Абс. значення	Бал B_i	Вага $K_в$	$K_в \cdot B_i$	K_k
Варіант 1 ($Pipeline\ A$)	$X1$	0.7566	64	0.360	23.05	
	$X2$	32	86	0.208	17.85	
	$X3$	42 хв	51	0.275	14.05	
	$X4$	620 год	51	0.157	8.00	62.95
Варіант 2 ($Pipeline\ B$)	$X1$	0.7661	72	0.360	25.93	
	$X2$	100	25	0.208	5.19	
	$X3$	38 хв	63	0.275	17.35	
	$X4$	411 год	89	0.157	13.97	62.44

Коефіцієнти технічного рівня якості: $K_k1 = 62.95$ (Варіант 1, Pipeline A); $K_k2 = 62.44$ (Варіант 2, Pipeline B). За технічним рівнем якості варіант 1 дещо переважає завдяки компактнішому представленню ознак, проте різниця незначна.

4.6 Економічний аналіз варіантів розробки

Для визначення вартості розробки спочатку розраховується трудомісткість. Обидва варіанти включають три основні завдання:

1) розробку модуля векторизації тексту (препроцесинг та Word2Vec);
2) розробку нейромережевого ядра (для варіанта 1 — денойзинговий автоенкодер та класифікатор; для варіанта 2 — класифікатор напряму без проміжного стиснення);
3) розробку модуля аналізу надійності та виведення результатів (для варіанта 1 — повний bootstrap-аналіз із трьома межами та Matplotlib-візуалізація; для варіанта 2 — спрощений аналіз з однією межею Вальда та CSV-виведення).

Загальна трудомісткість програмування завдання обчислюється за формулою (4.4):

$$T_o = T_p \cdot K_{\text{п}} \cdot K_{\text{ск}} \cdot K_{\text{м}} \cdot K_{\text{ст}} \cdot K_{\text{ст.м}} \quad (4.4)$$

де T_p — норма часу на розробку (за таблицею норм часу); $K_{\text{п}}$ — поправковий коефіцієнт виду інформації; $K_{\text{ск}}$ — коефіцієнт складності контролю вхідної інформації ($K_{\text{ск}} = 1$); $K_{\text{м}}$ — коефіцієнт рівня мови програмування ($K_{\text{м}} = 1$ для мови високого рівня); $K_{\text{ст}}$ — коефіцієнт використання стандартних модулів ($K_{\text{ст}} = 0.8$); $K_{\text{ст.м}}$ — коефіцієнт стандартного математичного забезпечення ($K_{\text{ст.м}} = 1$).

Розрахунок трудомісткості завдань для двох варіантів наведено у таблиці 4.7. Значення $K_{\text{п}}$ визначаються за типом вхідної інформації: для завдань, що використовують нормативно-довідкову інформацію (новизна Б), $K_{\text{п}} = 1.01$; для завдань із новизною А (розробка нових алгоритмів) $K_{\text{п}} = 1.26$; для завдань, що використовують первинну інформацію (новизна В, аналіз результатів), $K_{\text{п}} = 0.90$. Коефіцієнт $K_{\text{ск}} = 1$ (нормальна складність контролю), $K_{\text{м}} = 1$ (мова високого рівня Python), $K_{\text{ст}} = 0.8$ (використання стандартних модулів), $K_{\text{ст.м}} = 1$.

Таблиця 4.7 — Розрахунок трудомісткості розробки

Завдання	Новизна	Складн.	$T_р$, дні	$K_п$	$T_о$, людино-днів
Варіант 1 (Pipeline A)					
1. Модуль векторизації	Б	2	27	1.01	21.82
2. Автоенкодер + класифікатор	А	2	36	1.26	36.29
3. Bootstrap-аналіз + візуалізація	В	2	27	0.90	19.44
Разом за варіантом 1					77.55 (620.4 год)
Варіант 2 (Pipeline B)					
1. Модуль векторизації	Б	2	27	1.01	21.82
2. Класифікатор напряму	В	2	27	0.90	19.44
3. Аналіз надійності (межа Вальда) + CSV-виведення	В	1	14	0.90	10.08
Разом за варіантом 2					51.34 (410.7 год)

Загальна трудомісткість становить: для варіанта 1 — 77.55 людино-днів (620.4 людино-годин); для варіанта 2 — 51.34 людино-днів (410.7 людино-годин). Норми часу перераховано в людино-години з розрахунку 8-годинного робочого дня.

У розробці бере участь один інженер-програміст з місячним окладом 35 000 грн. Середня погодинна оплата праці визначається за формулою (4.5):

$$С_{ч} = \frac{М}{N_{д} \cdot N_{г}} = \frac{35000}{21 \cdot 8} = 208.33 \text{ грн/год} \quad (4.5)$$

де М — місячний оклад; $N_{д} = 21$ — кількість робочих днів на місяць; $N_{г} = 8$ — кількість робочих годин на день. Заробітна плата розробника за варіантами (з урахуванням нормативу додаткової заробітної плати $K_{д} = 1.2$):

$$С_{зп1} = 208.33 \cdot 620.4 \cdot 1.2 = 155\,100 \text{ грн}$$

$$С_{зп2} = 208.33 \cdot 410.7 \cdot 1.2 = 102\,700 \text{ грн} \quad (4.6)$$

Відрахування на єдиний соціальний внесок становлять 22 %:

$$С_{від1} = 155\,100 \cdot 0.22 = 34\,122 \text{ грн};$$

$$С_{від2} = 102\,700 \cdot 0.22 = 22\,594 \text{ грн} \quad (4.7)$$

Витрати на оплату машинного часу визначаються через собівартість машино-години. Розраховуємо річні витрати на утримання ЕОМ. Оклад обслуговуючого персоналу ($М_{опер} = 35\,000$ грн), коефіцієнт зайнятості $K_{з} = 0.2$:

$$С_{зп}(ЕОМ) = 12 \cdot М_{опер} \cdot K_{з} = 12 \cdot 35\,000 \cdot 0.2 = 84\,000 \text{ грн}$$

$С_{зп}(ЕОМ)$ з урахуванням додаткової заробітної плати ($K_{д} = 1.2$):

$$С_{зп}(ЕОМ) = 84\,000 \cdot 1.2 = 100\,800 \text{ грн}$$

$$С_{від}(ЕОМ) = 100\,800 \cdot 0.22 = 22\,176 \text{ грн}$$

Амортизаційні відрахування за норми амортизації 25 % та вартості ЕОМ 35 000 грн. $Ц_{пр} = 35\,000$ грн — вартість ЕОМ. $К_{а} = 0.25$ — норма амортизації. $К_{тм} = 1.15$ — коефіцієнт транспортно − монтажних витрат.

$$С_{а} = К_{тм} \cdot К_{а} \cdot Ц_{пр} = 1.15 \cdot 0.25 \cdot 35000 = 10\,062.5 \text{ грн} \quad (4.8)$$

Витрати на ремонт і профілактику $К_{р} = 5$ % — коефіцієнт витрат на ремонт»:

$$С_{р} = К_{тм} \cdot Ц_{пр} \cdot К_{р} = 1.15 \cdot 35000 \cdot 0.05 = 2\,012.5 \text{ грн} \quad (4.9)$$

Ефективний річний фонд часу ЕОМ:

$$\text{Т}_{\text{еф}} = (365 - 104 - 11 - 8) \cdot 8 \cdot 0.85 = 1\,645.6 \text{ год} \quad (4.10)$$

Витрати на електроенергію (потужність $N_{\text{с}} = 0.2$ кВт, тариф $\text{Ц}_{\text{ен}} = 13.64$ грн/кВт · год):

$$\text{С}_{\text{ел}} = \text{Т}_{\text{еф}} \cdot N_{\text{с}} \cdot \text{К}_{\text{з}} \cdot \text{Ц}_{\text{ен}} = 1645.6 \cdot 0.2 \cdot 0.2 \cdot 13.64 = 897.84 \text{ грн} \quad (4.11)$$

Накладні витрати на утримання ЕОМ (67 % від вартості):

$$\text{С}_{\text{н}}(\text{ЕОМ}) = 35000 \cdot 0.67 = 23\,450 \text{ грн}$$

Річні експлуатаційні витрати:

$$\text{С}_{\text{екс}} = 100\,800 + 22\,176 + 10\,062.5 + 2\,012.5 + 897.84 + 23\,450 = 159\,398.84 \text{ грн} \quad (4.12)$$

Собівартість однієї машино-години:

$$\text{С}_{\text{м-г}} = \frac{\text{С}_{\text{екс}}}{\text{Т}_{\text{еф}}} = \frac{159\,398.84}{1645.6} = 96.86 \text{ грн/год} \quad (4.13)$$

Витрати на оплату машинного часу за варіантами:

$$\text{С}_{\text{м}}1 = 96.86 \cdot 620.4 = 60\,094.21 \text{ грн;}$$

$$\text{С}_{\text{м}}2 = 96.86 \cdot 410.7 = 39\,779.00 \text{ грн} \quad (4.14)$$

Накладні витрати на розробку (67 % від заробітної плати):

$$\text{С}_{\text{н}}1 = 155\,100 \cdot 0.67 = 103\,917 \text{ грн;}$$

$$\text{С}_{\text{н}}2 = 102\,700 \cdot 0.67 = 68\,809 \text{ грн} \quad (4.15)$$

Повна вартість розробки за варіантами:

$$\text{С}_1 = 155\,100 + 34\,122 + 60\,094.21 + 103\,917 = 353\,233.21 \text{ грн} \quad (4.16)$$

$$\text{С}_2 = 102\,700 + 22\,594 + 39\,779.00 + 68\,809 = 233\,882.00 \text{ грн} \quad (4.17)$$

Результати оформлені в таблиці 4.8:

Таблиця 4.8 — Зведені економічні показники

Стаття витрат	Варіант 1, грн	Варіант 2, грн
Заробітна плата розробника	155 100.00	102 700.00
Єдиний соціальний внесок (22 %)	34 122.00	22 594.00
Витрати на машинний час	60 094.21	39 779.00
Накладні витрати (67 %)	103 917.00	68 809.00
Повна вартість розробки	353 233.21	233 882.00

4.7 Вибір кращого варіанту за техніко-економічним рівнем

Коефіцієнт техніко-економічного рівня обчислюється як відношення коефіцієнта технічного рівня якості до повної вартості розробки:

$$K_{\text{тер}} = \frac{K_{\text{к}}}{\text{С}} \quad (4.18)$$

$$K_{\text{тер}}1 = 62.95 \,/\, 353\,233.21 = 1.782 \cdot 10^{-4} \quad (4.19)$$

$$K_{\text{тер}}2 = 62.44 \,/\, 233\,882.00 = 2.670 \cdot 10^{-4} \quad (4.20)$$

Оскільки $K_{\text{тер}}2 > K_{\text{тер}}1$, найкраще співвідношення технічного рівня якості до витрат забезпечує варіант 2 (Pipeline B). Цей варіант має дещо нижчий технічний рівень якості, проте суттєво нижчу вартість розробки за рахунок відсутності етапу денойзингового автоенкодера, спрощеного аналізу надійності (одна межа Вальда) та мінімалістичного виведення результатів у CSV-файли, що робить його економічно ефективнішим.

Отриманий результат узгоджується з експериментальними висновками третього розділу, де архітектура без проміжного стиснення (Pipeline B) показала вищу F1-міру. Водночас у дипломній роботі денойзинговий

автоенкодер обрано з методологічних міркувань — для виділення вилучення ознак як окремого етапу, придатного для незалежного аналізу стохастичної надійності, що не є економічним критерієм і тому не суперечить результатам ФВА.

4.8 Висновки до розділу 4

У розділі проведено функціонально-вартісний аналіз програмної системи бінарної класифікації текстових даних з аналізом надійності метрик. Визначено основні функції системи: вибір мови реалізації, методу вилучення ознак, методу аналізу надійності та способу візуалізації. Для кожної функції розглянуто варіанти реалізації, побудовано позитивно-негативну матрицю та сформовано два варіанти системи, що відповідають реальним архітектурам Pipeline A та Pipeline B.

Експертним оцінюванням визначено вагомості чотирьох параметрів якості з коефіцієнтом узгодженості $W = 0.771$. Розраховано коефіцієнти технічного рівня якості ($K_к 1 = 62.95; K_к 2 = 62.44$) та повну вартість розробки кожного варіанта ($\mathrm{C1} = 353\,233.21$ грн; $\mathrm{C2} = 233\,882.00$ грн).

За коефіцієнтом техніко-економічного рівня оптимальним визнано варіант 2 (Pipeline B) із $K_{тер} 2 = 2.670 \cdot 10^{-4}$, що забезпечує найкраще співвідношення технічного рівня якості до витрат на розробку. Результат економічного аналізу узгоджується з експериментальними висновками дипломної роботи.

ВИСНОВКИ

У роботі досліджено вплив обсягу вибірки на стабільність метрик бінарної класифікації тональності тексту та визначено мінімальний обсяг даних, достатній для отримання математично надійних результатів на прикладі датасету Sentiment140. Мета досягнута на теоретичному рівні через залучення відомих результатів теорії ймовірностей та на експериментальному — через bootstrap-аналіз метрик при різних обсягах тестової вибірки.

Теоретичну основу складають класичні результати: збіжність емпіричних оцінок обґрунтовано через закон великих чисел Хінчина та посилений закон Колмогорова. Для побудови довірчих інтервалів використано три відомі нерівності — Чебишова, Вальда (через центральну граничну теорему) та Хеффдінга. Векторизація тексту виконана алгоритмом Word2Vec Skip-Gram, архітектура автоенкодера спирається на денойзингове формулювання Vincent, а метрики оцінювання якості (Повнота, Точність, F1-міра, MCC, AUC, AP, Каппа Коена) є стандартними інструментами.

У роботі розроблено модульну систему бінарної класифікації тональності твітів з повною відтворюваністю результатів. Побудовано об'єднану методологію аналізу стохастичної надійності метрик, у якій одночасно обчислюються три теоретичні межі довірчого інтервалу та емпірично перевіряється їх ймовірність покриття через bootstrap-повторення на реальних даних. Експериментально визначено дві окремі точки насичення: тренувальної вибірки $n^*\_\text{train} \approx 100\,000$ твітів, після якої приріст F1 стає несуттєвим, та надійної оцінки $n^*\_test \approx 25\,000$ твітів, при якій стандартне відхилення оцінки p_1 опускається нижче 0.005. Виявлено явище типу posterior collapse у класичному формулюванні розрідженого автоенкодера на щільних усереднених Word2Vec векторах та запропоновано стійку альтернативу на основі денойзингового формулювання.

Експериментальна перевірка на тестовій вибірці обсягом 238 820 твітів з 30-кратним bootstrap-повторенням підтвердила теоретичні передбачення: оцінки p_1 та q_0 стабілізуються до граничних значень, швидкість збіжності $O(\frac{1}{\sqrt{n}})$ відтворюється з точністю 10.52 проти теоретичних 10.00 при зростанні n у 100 разів, ієрархія Чеб > Хеф > Вальд тримається у відношенні 2.28 : 1.39 : 1.00. Ймовірність покриття для меж Чебишова та Хеффдінга становить 1.000 (консервативні межі), для Вальда ≈ 0.96, що відповідає номінальному рівню довіри $\gamma = 0.95$.

У межах функціонально-вартісного аналізу розглянуто два варіанти реалізації pipeline: з проміжним автоенкодером (Pipeline A) та без нього (Pipeline B). Експертним оцінюванням визначено вагомості чотирьох параметрів якості з коефіцієнтом узгодженості $W = 0.771$, що перевищує нормативний поріг. Коефіцієнти технічного рівня якості варіантів виявились близькими ($K_к1 = 62.95$; $K_к2 = 62.44$), однак повна вартість розробки Pipeline B є суттєво нижчою (233 882 грн проти 353 233 грн), що забезпечило перевагу за коефіцієнтом техніко-економічного рівня ($K_{тер}2 = 2.670 \cdot 10^{-4}$ проти $K_{тер}1 = 1.782 \cdot 10^{-4}$). Pipeline A залишено у роботі з методологічних міркувань — для виокремлення етапу формування ознак як окремої стадії, що є центральним для теми дослідження.

Наукова новизна полягає в об'єднанні трьох класичних теоретичних меж в єдину методологію аналізу надійності метрик класифікації, її експериментальній верифікації на масштабному реальному датасеті та доведенні pipeline-агностичності — швидкість збіжності та ієрархія меж зберігаються як у вхідному просторі Word2Vec, так і у стиснутому латентному просторі.

Практичне значення — можливість обґрунтованого вибору обсягу вибірки при проектуванні систем бінарної класифікації. Замість поширеної практики повідомлення значення Acc без оцінки її стабільності методологія

дає статистично обґрунтоване твердження про точність метрик у заданому довірчому інтервалі. Для якості F1 ≈ 0.75 на Sentiment140 достатньо 100 000 твітів для навчання та 25 000 для оцінювання; для критичних застосувань з вимогою гарантованого покриття рекомендовано оцінку Чебишова; для асимптотичних задач — інтервал Вальда.

Перспективи роботи полягають в можливості поширення методології на багатокласову класифікацію, незбалансовані задачі та трансформерні моделі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Павлов О.А., Гавриленко О.В., Рибачук Л.В. Теорія ймовірностей, імовірнісні процеси та математична статистика: навчальний посібник. Курс лекцій. Частина 1. Київ: КПІ ім. Ігоря Сікорського, 2021. 154 с. URL:https://ela.kpi.ua/items/56d30a7a-d00d-45d9-9716-4ab8a48fc335
2. Гнєденко Б.В. *Курс теорії ймовірностей*: підручник. Київ: Видавничо-поліграфічний центр "Київський університет", 2010. 464 с. URL: https://probability.knu.ua/userfiles/yamnenko/Gnedenko.pdf
3. Врубльовська О.О., Мулик О.В. Аналіз результатів опитування з використанням stepwise method в моделі логістичної регресії на платформі MS Excel. *Mathematics in Modern Technical University*. 2025. Вип. 2. С. 65–82. Київ: КПІ ім. Ігоря Сікорського. URL: https://ela.kpi.ua/server/api/core/bitstreams/98da97db-0087-4636-b894-299385fa684b/content
4. Терейковський І.А., Корченко О.Г., Іванніков В.Ю. *Нейронні мережі. Практикум*: навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2023. 157 с. URL: https://ela.kpi.ua/items/d36d6607-96fc-48a1-892f-2489926d2572
5. Корабльов М.М., Каніовська І.Ю. Порівняння точних та наближених методів побудови довірчих інтервалів для параметрів деяких розподілів. Теоретичні і прикладні проблеми фізики, математики та інформатики: Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених. Київ: Навчально-науковий Фізико-технічний інститут КПІ ім. Ігоря Сікорського. 11-12 травня 2023. С. 392-394. URL: https://ela.kpi.ua/server/api/core/bitstreams/3f49d189-6450-4edd-a4c4-c1806b0d191a/content

6. Pattanayak S. *Pro Deep Learning with TensorFlow 2.0: A Mathematical Approach to Advanced Artificial Intelligence in Python*. 2nd ed. Apress, 2023. 656 p.
7. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. Cambridge: MIT Press, 2016. 800 p.
   URL: https://www.deeplearningbook.org/
8. Hoeffding W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*. 1963. Vol. 58, No. 301. P. 13–30.
   URL: https://www.jstor.org/stable/2282952
9. Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. *Distributed Representations of Words and Phrases and their Compositionality*. Advances in Neural Information Processing Systems. 2013. Vol. 26. P. 3111–3119.
   URL: https://arxiv.org/abs/1310.4546
10. Vincent P., Larochelle H., Bengio Y., Manzagol P.-A. *Extracting and Composing Robust Features with Denoising Autoencoders*. Proceedings of the 25th International Conference on Machine Learning (ICML), 5-9 July 2008. Helsinki, Finland. P. 1096–1103.
    URL: https://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf
11. Bengio Y., Courville A., Vincent P. *Representation Learning: A Review and New Perspectives*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013. Vol. 35, No. 8. P. 1798–1828.
    URL: https://arxiv.org/abs/1206.5538
12. Ng A. *Sparse Autoencoder*: CS294A Lecture Notes. Stanford University, 2011. 19 p.
    URL: https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf
13. Go A., Bhayani R., Huang L. *Twitter Sentiment Classification using Distant Supervision*. CS224N Project Report, Stanford University, 2009. 12 p.URL:

https://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf

14. van der Vaart A.W. Mathematische Statistiek (Asymptotic Statistics): lecture notes. Amsterdam: Vrije Universiteit, 1998. 74 p. URL: https://staff.fnwi.uva.nl/b.j.k.kleijn/AS-lecnotes-ch1-5-vdV95.pdf
15. Paszke A., Gross S., Massa F. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems. 2019. Vol. 32. P. 8024–8035. URL: https://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
16. Řehůřek R., Sojka P. Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. Valletta, Malta, 2010. P. 45–50. URL: https://radimrehurek.com/gensim/
17. Pedregosa F., Varoquaux G., Gramfort A. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830. URL: https://www.jmlr.org/papers/v12/pedregosa11a.html
18. Harris C.R., Millman K.J., van der Walt S.J. et al. Array Programming with NumPy. Nature. 2020. Vol. 585, No. 7825. P. 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2
19. Hunter J.D. Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering. 2007. Vol. 9, No. 3. P. 90–95. URL: https://doi.org/10.1109/MCSE.2007.55
20. Пашін В.П., Романов В.В., Єгорова Н.В. Методичні вказівки до виконання економіко-організаційного розділу дипломних проектів (робіт) бакалаврів і спеціалістів для студентів Інституту прикладного системного аналізу. Київ: НТУУ «КПІ», 2011. 118 с.

ДОДАТОК А ЛІСТІНГ ПРОГРАМНОГО МОДУЛЮ

```
import os
import math


BASE_DIR = os.path.dirname(os.path.abspath(__file__))

DATA_RAW       = os.path.join(BASE_DIR, "data", "raw")
DATA_CLEANED   = os.path.join(BASE_DIR, "data", "cleaned")
DATA_VECTORS   = os.path.join(BASE_DIR, "data", "vectors")
MODELS_DIR     = os.path.join(BASE_DIR, "models")
RESULTS_TABLES = os.path.join(BASE_DIR, "results", "tables")
RESULTS_FIGS   = os.path.join(BASE_DIR, "results", "figures")

RAW_CSV     = os.path.join(DATA_RAW,
"training.1600000.processed.noemoticon.csv")
CLEANED_CSV = os.path.join(DATA_CLEANED, "tweets_cleaned.csv")
TRAIN_CSV   = os.path.join(DATA_CLEANED, "train.csv")
VAL_CSV     = os.path.join(DATA_CLEANED, "val.csv")
TEST_CSV    = os.path.join(DATA_CLEANED, "test.csv")


RANDOM_SEED = 18


TRAIN_RATIO = 0.70
VAL_RATIO   = 0.15
TEST_RATIO  = 0.15


PREPROC_BATCH_SIZE = 10_000

NEGATION_WORDS = {
    "not", "no", "never", "neither", "nor",
    "nothing", "nobody", "nowhere", "none"
}


W2V_DIM        = 100
W2V_WINDOW     = 5
W2V_MIN_COUNT  = 2
W2V_WORKERS    = 4
W2V_EPOCHS     = 10
W2V_MODEL_PATH = os.path.join(MODELS_DIR, "word2vec.model")

VEC_TRAIN_X = os.path.join(DATA_VECTORS, "X_train.npy")
VEC_TRAIN_Y = os.path.join(DATA_VECTORS, "y_train.npy")
VEC_VAL_X   = os.path.join(DATA_VECTORS, "X_val.npy")
VEC_VAL_Y   = os.path.join(DATA_VECTORS, "y_val.npy")
VEC_TEST_X  = os.path.join(DATA_VECTORS, "X_test.npy")
VEC_TEST_Y  = os.path.join(DATA_VECTORS, "y_test.npy")


AE_INPUT_DIM   = 100
AE_HIDDEN_DIM  = 64
AE_LATENT_DIM  = 32
AE_NOISE_STD   = 0.3
```

```
AE_LAMBDA      = 0.001
AE_BATCH_SIZE  = 256
AE_EPOCHS      = 50
AE_PATIENCE    = 5
AE_LR          = 1e-3


AE_RHO   = 0.05
AE_BETA  = 3.0

AE_MODEL_PATH      = os.path.join(MODELS_DIR, "autoencoder.pt")
ENCODER_MODEL_PATH = os.path.join(MODELS_DIR, "encoder.pt")
AE_SCALER_PATH     = os.path.join(MODELS_DIR, "minmax_scaler.pkl")

LATENT_TRAIN = os.path.join(DATA_VECTORS, "Z_train.npy")
LATENT_VAL   = os.path.join(DATA_VECTORS, "Z_val.npy")
LATENT_TEST  = os.path.join(DATA_VECTORS, "Z_test.npy")


CLF_INPUT_DIM  = 32
CLF_HIDDEN_1   = 16
CLF_HIDDEN_2   = 8
CLF_DROPOUT    = 0.3
CLF_BATCH_SIZE = 512
CLF_EPOCHS     = 30
CLF_PATIENCE   = 5
CLF_LR         = 1e-3
CLF_THRESHOLD  = 0.5

CLF_MODEL_PATH = os.path.join(MODELS_DIR, "classifier.pt")


EXP1_TRAIN_SIZES = [1_000, 5_000, 10_000, 50_000, 100_000, 500_000,
1_100_000]
EXP1_RESULTS_CSV = os.path.join(RESULTS_TABLES, "learning_curve.csv")


EXP2_TEST_SIZES  = [500, 1_000, 2_000, 5_000, 10_000, 25_000, 50_000,
100_000, 240_000]
EXP2_BOOTSTRAPS  = 30
EXP2_GAMMA       = 0.95

EXP2_RESULTS_CSV     = os.path.join(RESULTS_TABLES,
"stochastic_analysis.csv")
EXP2_COVERAGE_CSV    = os.path.join(RESULTS_TABLES,
"coverage_probability.csv")


EXP2_K_CHEBYSHEV = 0.5 / math.sqrt(1.0 - EXP2_GAMMA)
EXP2_K_HOEFFDING = math.sqrt(math.log(2.0 / (1.0 - EXP2_GAMMA)) / 2.0)
EXP2_Z_WALD      = 1.96
EXP2_K_WALD_CONS = EXP2_Z_WALD / 2.0


EXP3_RESULTS_CSV = os.path.join(RESULTS_TABLES, "pipeline_comparison.csv")

METRIC_NAMES = {
    "recall":    "Повнота (Recall)",
    "precision": "Точність (Precision)",
    "f1":        "F1-міра (F1)",
    "accuracy":  "Загальна точність (Acc)",
```

```
    "balanced":  "Збалансована точність (BA)",
    "mcc":       "Коефіцієнт кореляції Метьюса (MCC)",
    "auc":       "Площа під ROC-кривою (AUC)",
    "ap":        "Середня точність (Average Precision, AP)",
    "kappa":     "Каппа Коена (κ)",
}

import os
import sys
import pandas as pd


sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config


COLUMN_NAMES = ["target", "ids", "date", "flag", "user", "text"]


def check_file_exists() -> None:

    if not os.path.exists(config.RAW_CSV):
        print(f" Файл не знайдено: {config.RAW_CSV}")
        print("   Завантаж датасет з Kaggle та поклади у data/raw/")
        sys.exit(1)

    size_mb = os.path.getsize(config.RAW_CSV) / (1024 * 1024)
    print(f" Файл знайдено: {os.path.basename(config.RAW_CSV)}")
    print(f"   Розмір: {size_mb:.1f} МБ")


def load_raw_dataset() -> pd.DataFrame:

    print("\n Читання CSV (це може зайняти 30-60 секунд)...")

    df = pd.read_csv(
        config.RAW_CSV,
        encoding="latin-1",   # ВАЖЛИВО: Sentiment140 у кодуванні latin-1, не
UTF-8
        header=None,           # У файлі немає заголовка
        names=COLUMN_NAMES,
    )
    print(f"Зчитано рядків: {len(df):,}")
    return df


def print_dataset_info(df: pd.DataFrame) -> None:

    print("\n Інформація про датасет:")
    print(f"   Кількість твітів:   {len(df):,}")
    print(f"   Кількість колонок:  {df.shape[1]}")
    print(f"   Назви колонок:      {list(df.columns)}")

    print("\n Розподіл міток (target):")
    print(df["target"].value_counts().sort_index())

    print("\n Пропуски у даних:")
    nulls = df.isnull().sum()
    if nulls.sum() == 0:
        print("   Пропусків немає ")
```

```
    else:
        print(nulls[nulls > 0])

    print("\n Приклад твітів:")
    for i, row in df.sample(3, random_state=config.RANDOM_SEED).iterrows():
        print(f"   [target={row['target']}] {row['text'][:80]}...")


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

    print("\n Перетворення міток {0, 4} → {0, 1}...")
    df = df.copy()
    df["target"] = df["target"].map({0: 0, 4: 1})


    if df["target"].isnull().any():
        print("  УВАГА: у target з'явились NaN — у датасеті були мітки крім 0
i 4")
        df = df.dropna(subset=["target"])

    df["target"] = df["target"].astype(int)
    print(f"   Розподіл після перетворення:")
    print(df["target"].value_counts().sort_index())
    return df


def save_as_parquet(df: pd.DataFrame) -> str:

    df_to_save = df[["text", "target"]].reset_index(drop=True)

    output_path = os.path.join(config.DATA_RAW, "sentiment140.parquet")
    df_to_save.to_parquet(output_path, index=False)

    size_mb = os.path.getsize(output_path) / (1024 * 1024)
    print(f"\n Збережено: {output_path}")
    print(f"   Розмір: {size_mb:.1f} МБ (порівняно з ~240 МБ оригіналу)")
    return output_path


def main() -> None:
    print("=" * 60)
    print(" 01_download.py — Перевірка датасету Sentiment140")
    print("=" * 60)

    check_file_exists()
    df = load_raw_dataset()
    print_dataset_info(df)
    df = remap_labels(df)
    save_as_parquet(df)

    print("\n" + "=" * 60)
    print("  Готово! Можна запускати 02_preprocessing.py")
    print("=" * 60)


if __name__ == "__main__":
    main()

import os
import re
import sys
```

```
import nltk
import pandas as pd
from tqdm import tqdm
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config


def download_nltk_resources() -> None:

    resources = [
        ("tokenizers/punkt",     "punkt"),
        ("tokenizers/punkt_tab", "punkt_tab"),
        ("corpora/stopwords",    "stopwords"),
        ("corpora/wordnet",      "wordnet"),
        ("corpora/omw-1.4",      "omw-1.4"),
    ]
    print(" Перевірка NLTK ресурсів...")
    for path, name in resources:
        try:
            nltk.data.find(path)
        except LookupError:
            print(f"   Завантаження {name}...")
            nltk.download(name, quiet=True)
    print("    Усі ресурси на місці\n")



URL_PATTERN     = re.compile(r"https?://\S+|www\.\S+")
MENTION_PATTERN = re.compile(r"@\w+")
HASHTAG_PATTERN = re.compile(r"#")
NON_LETTER      = re.compile(r"[^a-zA-Z\s]")
MULTI_SPACE     = re.compile(r"\s+")



def clean_text(text: str, stop_words: set, lemmatizer: WordNetLemmatizer) ->
str:

    if not isinstance(text, str):
        return ""


    text = URL_PATTERN.sub(" ", text)

    text = MENTION_PATTERN.sub(" ", text)

    text = HASHTAG_PATTERN.sub("", text)

    text = NON_LETTER.sub(" ", text)

    text = text.lower()

    text = MULTI_SPACE.sub(" ", text).strip()

    if not text:
```

```
        return ""


    tokens = word_tokenize(text)


    tokens = [t for t in tokens if t not in stop_words and len(t) > 1]

    if not tokens:
        return ""


    tokens = [lemmatizer.lemmatize(t) for t in tokens]

    return " ".join(tokens)



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

    base_stop = set(stopwords.words("english"))
    stop_words = base_stop - config.NEGATION_WORDS
    lemmatizer = WordNetLemmatizer()

    print(f" Очищення тексту батчами по {config.PREPROC_BATCH_SIZE:,}...")
    print(f"   Базовий розмір стоп-слів: {len(base_stop)}")
    print(f"   Збережено заперечень:     {len(config.NEGATION_WORDS)}")
    print(f"   Фінальних стоп-слів:      {len(stop_words)}\n")

    cleaned = []
    n = len(df)
    batch_size = config.PREPROC_BATCH_SIZE

    for start in tqdm(range(0, n, batch_size), desc="Батчі"):
        end = min(start + batch_size, n)
        batch = df["text"].iloc[start:end]
        cleaned_batch = [clean_text(t, stop_words, lemmatizer) for t in
batch]
        cleaned.extend(cleaned_batch)

    df = df.copy()
    df["text_clean"] = cleaned
    return df



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

    before = len(df)
    df = df[df["text_clean"].str.len() > 0].reset_index(drop=True)
    after = len(df)
    removed = before - after
    pct = 100 * removed / before
    print(f"\n  Видалено порожніх рядків: {removed:,} ({pct:.2f}%)")
    print(f"   Залишилось: {after:,}")
    return df



def stratified_split(df: pd.DataFrame) -> tuple[pd.DataFrame, pd.DataFrame,
pd.DataFrame]:
```

```
    print("\n  Стратифікований розподіл 70/15/15...")

    train, temp = train_test_split(
        df,
        test_size=(config.VAL_RATIO + config.TEST_RATIO),
        stratify=df["target"],
        random_state=config.RANDOM_SEED,
    )


    val_ratio_within_temp = config.VAL_RATIO / (config.VAL_RATIO +
config.TEST_RATIO)
    val, test = train_test_split(
        temp,
        test_size=(1 - val_ratio_within_temp),
        stratify=temp["target"],
        random_state=config.RANDOM_SEED,
    )

    train = train.reset_index(drop=True)
    val   = val.reset_index(drop=True)
    test  = test.reset_index(drop=True)

    print(f"   Train: {len(train):,} ({100*len(train)/len(df):.1f}%)")
    print(f"   Val:   {len(val):,} ({100*len(val)/len(df):.1f}%)")
    print(f"   Test:  {len(test):,} ({100*len(test)/len(df):.1f}%)")

    # Перевірка балансу
    print("\n   Баланс класів (частка позитивних):")
    print(f"     Train: {train['target'].mean():.4f}")
    print(f"     Val:   {val['target'].mean():.4f}")
    print(f"     Test:  {test['target'].mean():.4f}")

    return train, val, test



def save_splits(train: pd.DataFrame, val: pd.DataFrame, test: pd.DataFrame) -
> None:

    os.makedirs(config.DATA_CLEANED, exist_ok=True)


    cols = ["text_clean", "target"]
    train_path = config.TRAIN_CSV.replace(".csv", ".parquet")
    val_path   = config.VAL_CSV.replace(".csv", ".parquet")
    test_path  = config.TEST_CSV.replace(".csv", ".parquet")

    train[cols].to_parquet(train_path, index=False)
    val[cols].to_parquet(val_path, index=False)
    test[cols].to_parquet(test_path, index=False)

    print(f"\n Збережено:")
    print(f"   {train_path}")
    print(f"   {val_path}")
    print(f"   {test_path}")


def main() -> None:
    print("=" * 60)
```

```
    print(" 02_preprocessing.py — Очищення та розподіл даних")
    print("=" * 60)

    download_nltk_resources()


    parquet_path = os.path.join(config.DATA_RAW, "sentiment140.parquet")
    if not os.path.exists(parquet_path):
        print(f" Не знайдено {parquet_path}")
        print("   Спочатку запусти 01_download.py")
        sys.exit(1)

    print(f" Читання {os.path.basename(parquet_path)}...")
    df = pd.read_parquet(parquet_path)
    print(f"   Зчитано рядків: {len(df):,}\n")

    df = preprocess_dataframe(df)
    df = drop_empty(df)

    print("\n Приклади очищених твітів:")
    for i, row in df.sample(5, random_state=config.RANDOM_SEED).iterrows():
        print(f"   [{row['target']}] {row['text_clean'][:80]}")

    train, val, test = stratified_split(df)
    save_splits(train, val, test)

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


if __name__ == "__main__":
    main()


import os
import sys
import time
import logging

import numpy as np
import pandas as pd
from gensim.models import Word2Vec


sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config



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



def load_train_sentences() -> list[list[str]]:

    train_path = config.TRAIN_CSV.replace(".csv", ".parquet")
```

```
    if not os.path.exists(train_path):
        print(f" Не знайдено {train_path}")
        print("   Спочатку запусти 02_preprocessing.py")
        sys.exit(1)

    print(f" Читання {os.path.basename(train_path)}...")
    df = pd.read_parquet(train_path)
    print(f"   Зчитано рядків: {len(df):,}")


    print(" Токенізація...")
    sentences = [text.split() for text in df["text_clean"].tolist()]


    lengths = [len(s) for s in sentences]
    print(f"   Кількість речень:  {len(sentences):,}")
    print(f"   Середня довжина:   {np.mean(lengths):.1f} слів")
    print(f"   Медіанна довжина:  {np.median(lengths):.1f} слів")
    print(f"   Максимальна:       {np.max(lengths)} слів")

    return sentences



def train_word2vec(sentences: list[list[str]]) -> Word2Vec:

    print("\n Навчання Word2Vec (skip-gram)...")
    print(f"   Розмірність вектора:  {config.W2V_DIM}")
    print(f"   Розмір вікна:         {config.W2V_WINDOW}")
    print(f"   Мін. частота слова:   {config.W2V_MIN_COUNT}")
    print(f"   Кількість епох:       {config.W2V_EPOCHS}")
    print(f"   Потоків:              {config.W2V_WORKERS}")

    start = time.time()

    model = Word2Vec(
        sentences=sentences,
        vector_size=config.W2V_DIM,
        window=config.W2V_WINDOW,
        min_count=config.W2V_MIN_COUNT,
        workers=config.W2V_WORKERS,
        sg=1,
        epochs=config.W2V_EPOCHS,
        seed=config.RANDOM_SEED,
    )

    elapsed = time.time() - start
    print(f"\n  Навчання зайняло: {elapsed/60:.1f} хв")

    return model

def print_model_info(model: Word2Vec) -> None:

    vocab_size = len(model.wv)
    print(f"\n Статистика моделі:")
    print(f"   Розмір словника:      {vocab_size:,} унікальних слів")
    print(f"   Розмірність векторів: {model.wv.vector_size}")


    print(f"\n Приклади схожих слів (sanity check):")
```

```
    test_words = ["good", "happy", "sad", "love"]
    for word in test_words:
        if word in model.wv:
            similar = model.wv.most_similar(word, topn=5)
            similar_str = ", ".join(f"{w}({s:.2f})" for w, s in similar)
            print(f"   {word:8s} → {similar_str}")
        else:
            print(f"   {word:8s} → (немає в словнику)")



def save_model(model: Word2Vec) -> None:

    os.makedirs(config.MODELS_DIR, exist_ok=True)
    model.save(config.W2V_MODEL_PATH)

    size_mb = os.path.getsize(config.W2V_MODEL_PATH) / (1024 * 1024)
    print(f"\n Збережено: {config.W2V_MODEL_PATH}")
    print(f"   Розмір: {size_mb:.1f} МБ")



def main() -> None:
    print("=" * 60)
    print(" 03_word2vec.py — Навчання Word2Vec на train-вибірці")
    print("=" * 60)

    sentences = load_train_sentences()
    model = train_word2vec(sentences)
    print_model_info(model)
    save_model(model)

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


if __name__ == "__main__":
    main()


import os
import sys
import time

import numpy as np
import pandas as pd
from tqdm import tqdm
from gensim.models import Word2Vec


sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config



def load_word2vec() -> Word2Vec:

    if not os.path.exists(config.W2V_MODEL_PATH):
        print(f" Не знайдено {config.W2V_MODEL_PATH}")
```

```
        print("   Спочатку запусти 03_word2vec.py")
        sys.exit(1)

    print(f" Завантаження Word2Vec моделі...")
    model = Word2Vec.load(config.W2V_MODEL_PATH)
    print(f"   Розмір словника:      {len(model.wv):,}")
    print(f"   Розмірність векторів: {model.wv.vector_size}")
    return model



def tweet_to_vector(tokens: list[str], wv, dim: int) -> np.ndarray:

    vectors = [wv[token] for token in tokens if token in wv]

    if len(vectors) == 0:

        return np.zeros(dim, dtype=np.float32)

    return np.mean(vectors, axis=0).astype(np.float32)



def vectorize_dataset(df: pd.DataFrame, wv, dim: int, name: str) ->
tuple[np.ndarray, np.ndarray, int]:

    n = len(df)
    X = np.zeros((n, dim), dtype=np.float32)
    y = df["target"].to_numpy(dtype=np.int64)

    fully_unknown = 0

    print(f"\n Векторизація {name} ({n:,} твітів)...")
    for i, text in enumerate(tqdm(df["text_clean"].tolist(), desc=name)):
        tokens = text.split()
        vec = tweet_to_vector(tokens, wv, dim)
        X[i] = vec
        if not np.any(vec):  # всі нулі = жодне слово не знайшлось
            fully_unknown += 1

    return X, y, fully_unknown



def load_split(name: str) -> pd.DataFrame:

    paths = {
        "train": config.TRAIN_CSV.replace(".csv", ".parquet"),
        "val":   config.VAL_CSV.replace(".csv", ".parquet"),
        "test":  config.TEST_CSV.replace(".csv", ".parquet"),
    }
    path = paths[name]
    if not os.path.exists(path):
        print(f" Не знайдено {path}")
        print("   Спочатку запусти 02_preprocessing.py")
        sys.exit(1)
    return pd.read_parquet(path)



def save_arrays(X: np.ndarray, y: np.ndarray, x_path: str, y_path: str) ->
```

```
None:

    os.makedirs(config.DATA_VECTORS, exist_ok=True)
    np.save(x_path, X)
    np.save(y_path, y)

    x_mb = os.path.getsize(x_path) / (1024 * 1024)
    y_mb = os.path.getsize(y_path) / (1024 * 1024)
    print(f"    {os.path.basename(x_path):20s} {X.shape}  ({x_mb:.1f} МБ)")
    print(f"    {os.path.basename(y_path):20s} {y.shape}  ({y_mb:.1f} МБ)")



def main() -> None:
    print("=" * 60)
    print(" 04_vectorize.py — Векторизація твітів через Word2Vec")
    print("=" * 60)

    start_time = time.time()


    model = load_word2vec()
    wv = model.wv
    dim = wv.vector_size


    splits = [
        ("train", config.VEC_TRAIN_X, config.VEC_TRAIN_Y),
        ("val",   config.VEC_VAL_X,   config.VEC_VAL_Y),
        ("test",  config.VEC_TEST_X,  config.VEC_TEST_Y),
    ]

    summary = []
    for name, x_path, y_path in splits:
        df = load_split(name)
        X, y, fully_unknown = vectorize_dataset(df, wv, dim, name)

        save_arrays(X, y, x_path, y_path)

        pct_unknown = 100 * fully_unknown / len(df)
        summary.append({
            "split":          name,
            "total":          len(df),
            "fully_unknown":  fully_unknown,
            "pct_unknown":    pct_unknown,
            "positive_ratio": float(y.mean()),
        })


    print("\n" + "=" * 60)
    print("  Підсумок векторизації")
    print("=" * 60)
    print(f"{'Split':<8} {'Всього':>10} {'Без векторів':>14} {'%':>6}  {'%
позит.':>10}")
    print("-" * 60)
    for s in summary:
        print(f"{s['split']:<8} {s['total']:>10,} {s['fully_unknown']:>14,} "
              f"{s['pct_unknown']:>5.2f}% {s['positive_ratio']*100:>9.2f}%")

    elapsed = time.time() - start_time
    print(f"\n  Загальний час: {elapsed/60:.1f} хв")
```

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


if __name__ == "__main__":
    main()


import os
import sys
import time
import joblib
import csv

import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config


NOISE_STD = getattr(config, "AE_NOISE_STD", 0.3)



def set_seeds(seed: int) -> None:
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False


class SparseAutoencoder(nn.Module):


    def __init__(self, input_dim: int = 100, hidden_dim: int = 64,
latent_dim: int = 32):
        super().__init__()
        self.enc_fc1 = nn.Linear(input_dim, hidden_dim)
        self.enc_fc2 = nn.Linear(hidden_dim, latent_dim)
        self.dec_fc1 = nn.Linear(latent_dim, hidden_dim)
        self.dec_fc2 = nn.Linear(hidden_dim, input_dim)

        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()

        # Xavier — підходить для tanh
        for m in [self.enc_fc1, self.enc_fc2, self.dec_fc1, self.dec_fc2]:
            nn.init.xavier_uniform_(m.weight)
            nn.init.zeros_(m.bias)

    def encode(self, x: torch.Tensor) -> torch.Tensor:
        h = self.relu(self.enc_fc1(x))
        return self.tanh(self.enc_fc2(h))
```

```
    def decode(self, z: torch.Tensor) -> torch.Tensor:
        h = self.relu(self.dec_fc1(z))
        return self.dec_fc2(h)        # Linear на виході — для StandardScaler
даних

    def forward(self, x: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor]:
        z = self.encode(x)
        return z, self.decode(z)



def load_and_scale() -> tuple[np.ndarray, np.ndarray, StandardScaler]:
    if not os.path.exists(config.VEC_TRAIN_X):
        print(f" Не знайдено {config.VEC_TRAIN_X}. Запусти 04_vectorize.py")
        sys.exit(1)

    print(" Завантаження векторів...")
    X_train = np.load(config.VEC_TRAIN_X)
    X_val   = np.load(config.VEC_VAL_X)
    print(f"   X_train: {X_train.shape}")
    print(f"   X_val:   {X_val.shape}")

    print("\n StandardScaler (mean=0, std=1)...")
    scaler = StandardScaler()
    X_train_s = scaler.fit_transform(X_train).astype(np.float32)
    X_val_s   = scaler.transform(X_val).astype(np.float32)
    print(f"   Train mean / std: {X_train_s.mean():.4f} /
{X_train_s.std():.4f}")
    print(f"   Val   mean / std: {X_val_s.mean():.4f} / {X_val_s.std():.4f}")

    os.makedirs(config.MODELS_DIR, exist_ok=True)
    joblib.dump(scaler, config.AE_SCALER_PATH)
    print(f"    Scaler: {config.AE_SCALER_PATH}")
    return X_train_s, X_val_s, scaler



def train_one_epoch(model, loader, optimizer, mse_fn, device, noise_std) ->
dict:
    model.train()
    total_loss = 0.0; n = 0
    for (x_batch,) in loader:
        x_batch = x_batch.to(device)

        noise = torch.randn_like(x_batch) * noise_std
        x_noisy = x_batch + noise

        optimizer.zero_grad()
        z, x_hat = model(x_noisy)
        loss = mse_fn(x_hat, x_batch)
        loss.backward(); optimizer.step()
        total_loss += loss.item() * x_batch.size(0)
        n += x_batch.size(0)
    return {"loss": total_loss / n}


@torch.no_grad()
def validate(model, loader, mse_fn, device) -> dict:

    model.eval()
```

```
    total_loss = 0.0; n = 0
    for (x_batch,) in loader:
        x_batch = x_batch.to(device)
        z, x_hat = model(x_batch)
        loss = mse_fn(x_hat, x_batch)
        total_loss += loss.item() * x_batch.size(0)
        n += x_batch.size(0)
    return {"loss": total_loss / n}



def train_autoencoder(X_train, X_val) -> SparseAutoencoder:
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"\n Пристрій: {device}")

    train_ds = TensorDataset(torch.from_numpy(X_train))
    val_ds   = TensorDataset(torch.from_numpy(X_val))
    train_loader = DataLoader(train_ds, batch_size=config.AE_BATCH_SIZE,
shuffle=True, num_workers=0)
    val_loader   = DataLoader(val_ds,   batch_size=config.AE_BATCH_SIZE,
shuffle=False, num_workers=0)

    model = SparseAutoencoder(
        input_dim=config.AE_INPUT_DIM,
        hidden_dim=config.AE_HIDDEN_DIM,
        latent_dim=config.AE_LATENT_DIM,
    ).to(device)

    optimizer = torch.optim.Adam(
        model.parameters(),
        lr=config.AE_LR,
        weight_decay=config.AE_LAMBDA,
    )
    mse_fn = nn.MSELoss()

    n_params = sum(p.numel() for p in model.parameters())
    print(f" Параметрів: {n_params:,}")
    print(f" Стиснення:  {config.AE_INPUT_DIM} → {config.AE_LATENT_DIM} "
          f"(×{config.AE_INPUT_DIM/config.AE_LATENT_DIM:.2f})")
    print(f" Шум σ:      {NOISE_STD}")
    print(f" λ (L2):     {config.AE_LAMBDA}")
    print(f"\n Навчання (макс {config.AE_EPOCHS} еп.,
patience={config.AE_PATIENCE})\n")

    best_val = float("inf"); patience = 0; history = []

    for epoch in range(1, config.AE_EPOCHS + 1):
        t0 = time.time()
        tr = train_one_epoch(model, train_loader, optimizer, mse_fn, device,
NOISE_STD)
        vl = validate(model, val_loader, mse_fn, device)
        dt = time.time() - t0

        history.append({"epoch": epoch, "train_loss": tr["loss"], "val_loss":
vl["loss"]})

        print(f"Epoch {epoch:02d}/{config.AE_EPOCHS} ({dt:5.1f}s)  "
              f"train_loss={tr['loss']:.5f}  |  val_loss={vl['loss']:.5f}")

        if vl["loss"] < best_val - 1e-6:
            best_val = vl["loss"]; patience = 0
```

```
            torch.save(model.state_dict(), config.AE_MODEL_PATH)
        else:
            patience += 1
            if patience >= config.AE_PATIENCE:
                print(f"\n Early stopping на епосі {epoch} (best
val_loss={best_val:.5f})")
                break

    model.load_state_dict(torch.load(config.AE_MODEL_PATH))

    history_path = os.path.join(config.RESULTS_TABLES,
"autoencoder_history.csv")
    os.makedirs(config.RESULTS_TABLES, exist_ok=True)
    with open(history_path, "w", newline="") as f:
        w = csv.DictWriter(f, fieldnames=history[0].keys())
        w.writeheader(); w.writerows(history)
    print(f"\n Історія: {history_path}")
    return model


@torch.no_grad()
def check_latent_quality(model: SparseAutoencoder, X_val: np.ndarray, y_val:
np.ndarray) -> None:
    device = next(model.parameters()).device
    model.eval()
    n_check = min(20_000, len(X_val))

    X = torch.from_numpy(X_val[:n_check]).to(device)
    Z = model.encode(X).cpu().numpy()
    y = y_val[:n_check]

    print("\n Статистика латенту Z:")
    print(f"   Розмірність:                  {Z.shape}")
    print(f"   Min / Max:                    {Z.min():.4f} / {Z.max():.4f}")
    print(f"   Std (global):                 {Z.std():.4f}")
    print(f"   Std per dim (середня):        {Z.std(axis=0).mean():.4f}")
    print(f"   Std per dim (мін/макс):       {Z.std(axis=0).min():.4f} /
{Z.std(axis=0).max():.4f}")

    n_active = int((Z.std(axis=0) > 0.01).sum())
    print(f"\n Активних нейронів (std > 0.01):
{n_active}/{config.AE_LATENT_DIM}")


    Z_pos = Z[y == 1]; Z_neg = Z[y == 0]
    diff = np.linalg.norm(Z_pos.mean(axis=0) - Z_neg.mean(axis=0))
    intra = (Z_pos.std(axis=0).mean() + Z_neg.std(axis=0).mean()) / 2
    ratio = diff / intra if intra > 1e-10 else 0

    print(f"\n РОЗДІЛИМІСТЬ POS vs NEG у латенті:")
    print(f"   Відстань між центрами:       {diff:.4f}")
    print(f"   Середня std всередині класу: {intra:.4f}")
    print(f"   Коефіцієнт розділимості:     {ratio:.4f}")
    print(f"   (у вхідному W2V була: 2.13)")

    if ratio > 1.0:
        print("\n    ЛАТЕНТ ЗБЕРІГ СИГНАЛ — класифікатор працюватиме!")
    elif ratio > 0.3:
        print("\n    Сигнал зберігся частково")
    else:
        print("\n    Сигнал втрачено — потрібно знов переробити")
```

```
def save_encoder(model: SparseAutoencoder) -> None:
    state = {
        "enc_fc1.weight": model.enc_fc1.weight,
        "enc_fc1.bias":   model.enc_fc1.bias,
        "enc_fc2.weight": model.enc_fc2.weight,
        "enc_fc2.bias":   model.enc_fc2.bias,
    }
    torch.save(state, config.ENCODER_MODEL_PATH)
    print(f" Encoder: {config.ENCODER_MODEL_PATH}")



def main() -> None:
    print("=" * 60)
    print(" 05_autoencoder.py — Denoising AE + StandardScaler")
    print("=" * 60)

    set_seeds(config.RANDOM_SEED)
    X_train, X_val, _ = load_and_scale()
    y_val = np.load(config.VEC_VAL_Y)
    model = train_autoencoder(X_train, X_val)
    check_latent_quality(model, X_val, y_val)
    save_encoder(model)

    print("\n" + "=" * 60)
    print("  Готово! Можна запускати 06_classifier.py")
    print("=" * 60)


if __name__ == "__main__":
    main()


import os
import sys
import time
import joblib
import csv

import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import (
    accuracy_score, balanced_accuracy_score, precision_score, recall_score,
    f1_score, matthews_corrcoef, roc_auc_score, cohen_kappa_score,
    average_precision_score,
)

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config


def set_seeds(seed: int) -> None:
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
```

```
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False



class Encoder(nn.Module):
    def __init__(self, input_dim=100, hidden_dim=64, latent_dim=32):
        super().__init__()
        self.enc_fc1 = nn.Linear(input_dim, hidden_dim)
        self.enc_fc2 = nn.Linear(hidden_dim, latent_dim)
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()

    def forward(self, x):
        return self.tanh(self.enc_fc2(self.relu(self.enc_fc1(x))))


def load_encoder(device: torch.device) -> Encoder:
    if not os.path.exists(config.ENCODER_MODEL_PATH):
        print(f" Не знайдено {config.ENCODER_MODEL_PATH}. Запусти
05_autoencoder.py")
        sys.exit(1)
    encoder = Encoder(
        input_dim=config.AE_INPUT_DIM,
        hidden_dim=config.AE_HIDDEN_DIM,
        latent_dim=config.AE_LATENT_DIM,
    ).to(device)
    state = torch.load(config.ENCODER_MODEL_PATH, map_location=device)
    encoder.load_state_dict(state)
    encoder.eval()
    print(" Encoder завантажено")
    return encoder



@torch.no_grad()
def compute_latents(encoder, X, scaler, device, name, batch_size=1024) ->
np.ndarray:
    print(f"   {name}: {X.shape} → ", end="", flush=True)

    X_scaled = scaler.transform(X).astype(np.float32)
    n = X_scaled.shape[0]
    Z = np.zeros((n, config.AE_LATENT_DIM), dtype=np.float32)
    for start in range(0, n, batch_size):
        end = min(start + batch_size, n)
        x_batch = torch.from_numpy(X_scaled[start:end]).to(device)
        z_batch = encoder(x_batch).cpu().numpy()
        Z[start:end] = z_batch
    print(f"Z {Z.shape}")
    return Z


def prepare_latents():
    print(" Завантаження векторів...")
    X_train = np.load(config.VEC_TRAIN_X); X_val = np.load(config.VEC_VAL_X);
X_test = np.load(config.VEC_TEST_X)
    y_train = np.load(config.VEC_TRAIN_Y).astype(np.float32)
    y_val   = np.load(config.VEC_VAL_Y).astype(np.float32)
    y_test  = np.load(config.VEC_TEST_Y).astype(np.float32)
    print(f"   X_train: {X_train.shape}, y_train: {y_train.shape}")
    print(f"   X_val:   {X_val.shape}, y_val:   {y_val.shape}")
```

```
    print(f"   X_test:  {X_test.shape}, y_test:  {y_test.shape}")

    print("\n Завантаження scaler (StandardScaler з v4)...")
    scaler = joblib.load(config.AE_SCALER_PATH)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"\n Пристрій: {device}")
    encoder = load_encoder(device)

    print("\n Обчислення латентних векторів Z через encoder...")
    Z_train = compute_latents(encoder, X_train, scaler, device, "train")
    Z_val   = compute_latents(encoder, X_val,   scaler, device, "val  ")
    Z_test  = compute_latents(encoder, X_test,  scaler, device, "test ")

    print("\n Збереження латентних векторів...")
    os.makedirs(config.DATA_VECTORS, exist_ok=True)
    np.save(config.LATENT_TRAIN, Z_train)
    np.save(config.LATENT_VAL,   Z_val)
    np.save(config.LATENT_TEST,  Z_test)
    for name, path in [("Z_train", config.LATENT_TRAIN),
                       ("Z_val",   config.LATENT_VAL),
                       ("Z_test",  config.LATENT_TEST)]:
        size_mb = os.path.getsize(path) / (1024 * 1024)
        print(f"   {name}: {path} ({size_mb:.1f} МБ)")

    return Z_train, Z_val, Z_test, y_train, y_val, y_test



class MLPClassifier(nn.Module):
    def __init__(self, input_dim=32, hidden1=16, hidden2=8, dropout=0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden1), nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),  nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden2, 1), nn.Sigmoid(),
        )
    def forward(self, x): return self.net(x).squeeze(-1)


def train_classifier(Z_train, y_train, Z_val, y_val):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    train_ds = TensorDataset(torch.from_numpy(Z_train),
torch.from_numpy(y_train))
    val_ds   = TensorDataset(torch.from_numpy(Z_val),
torch.from_numpy(y_val))
    train_loader = DataLoader(train_ds, batch_size=config.CLF_BATCH_SIZE,
shuffle=True)
    val_loader   = DataLoader(val_ds,   batch_size=config.CLF_BATCH_SIZE,
shuffle=False)

    model = MLPClassifier(config.CLF_INPUT_DIM, config.CLF_HIDDEN_1,
config.CLF_HIDDEN_2, config.CLF_DROPOUT).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=config.CLF_LR)
    bce = nn.BCELoss()

    n_params = sum(p.numel() for p in model.parameters())
    print(f"\n Параметрів у класифікаторі: {n_params:,}")
    print(f" Навчання (макс {config.CLF_EPOCHS} епох,
patience={config.CLF_PATIENCE})\n")
```

```
    best_val = float("inf"); patience = 0; history = []

    for epoch in range(1, config.CLF_EPOCHS + 1):
        t0 = time.time()

        model.train()
        train_loss = train_correct = train_n = 0
        for x_batch, y_batch in train_loader:
            x_batch = x_batch.to(device); y_batch = y_batch.to(device)
            optimizer.zero_grad()
            preds = model(x_batch)
            loss = bce(preds, y_batch)
            loss.backward(); optimizer.step()
            bs = x_batch.size(0)
            train_loss    += loss.item() * bs
            train_correct += ((preds > config.CLF_THRESHOLD).float() ==
y_batch).sum().item()
            train_n       += bs

        model.eval()
        val_loss = val_correct = val_n = 0
        with torch.no_grad():
            for x_batch, y_batch in val_loader:
                x_batch = x_batch.to(device); y_batch = y_batch.to(device)
                preds = model(x_batch)
                val_loss    += bce(preds, y_batch).item() * x_batch.size(0)
                val_correct += ((preds > config.CLF_THRESHOLD).float() ==
y_batch).sum().item()
                val_n       += x_batch.size(0)

        train_loss /= train_n; val_loss /= val_n
        train_acc = train_correct / train_n; val_acc = val_correct / val_n
        dt = time.time() - t0

        history.append({"epoch": epoch, "train_loss": train_loss,
"train_acc": train_acc,
                        "val_loss": val_loss, "val_acc": val_acc})

        print(f"Epoch {epoch:02d}/{config.CLF_EPOCHS} ({dt:4.1f}s)  "
              f"train: loss={train_loss:.4f} acc={train_acc:.4f}  |  "
              f"val: loss={val_loss:.4f} acc={val_acc:.4f}")

        if val_loss < best_val - 1e-6:
            best_val = val_loss; patience = 0
            torch.save(model.state_dict(), config.CLF_MODEL_PATH)
        else:
            patience += 1
            if patience >= config.CLF_PATIENCE:
                print(f"\n Early stopping на епоці {epoch} (best val_loss =
{best_val:.4f})")
                break

    model.load_state_dict(torch.load(config.CLF_MODEL_PATH))

    history_path = os.path.join(config.RESULTS_TABLES,
"classifier_history.csv")
    os.makedirs(config.RESULTS_TABLES, exist_ok=True)
    with open(history_path, "w", newline="") as f:
        w = csv.DictWriter(f, fieldnames=history[0].keys())
        w.writeheader(); w.writerows(history)
    print(f"\n Історія: {history_path}")
```

```
    return model


@torch.no_grad()
def evaluate(model, Z, y) -> dict:
    device = next(model.parameters()).device
    model.eval()
    X_t = torch.from_numpy(Z).to(device)
    probs = model(X_t).cpu().numpy()
    preds = (probs > config.CLF_THRESHOLD).astype(int)
    y_int = y.astype(int)
    return {
        config.METRIC_NAMES["accuracy"]:  accuracy_score(y_int, preds),
        config.METRIC_NAMES["balanced"]:  balanced_accuracy_score(y_int,
preds),
        config.METRIC_NAMES["precision"]: precision_score(y_int, preds),
        config.METRIC_NAMES["recall"]:    recall_score(y_int, preds),
        config.METRIC_NAMES["f1"]:        f1_score(y_int, preds),
        config.METRIC_NAMES["mcc"]:       matthews_corrcoef(y_int, preds),
        config.METRIC_NAMES["auc"]:       roc_auc_score(y_int, probs),
        config.METRIC_NAMES["ap"]:        average_precision_score(y_int,
probs),  # НОВА
        config.METRIC_NAMES["kappa"]:     cohen_kappa_score(y_int, preds),
    }


def print_metrics(metrics: dict, title="Метрики на тестовій вибірці") ->
None:
    print(f"\n {title}:")
    print("-" * 52)
    for name, value in metrics.items():
        print(f"   {name:<44s} {value:.4f}")


def save_metrics(metrics: dict) -> None:
    out_path = os.path.join(config.RESULTS_TABLES,
"classifier_test_metrics.csv")
    os.makedirs(config.RESULTS_TABLES, exist_ok=True)
    with open(out_path, "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(["Метрика", "Значення"])
        for name, value in metrics.items():
            w.writerow([name, f"{value:.6f}"])
    print(f"\n Метрики: {out_path}")



def main() -> None:
    print("=" * 60)
    print(" 06_classifier.py — MLP-класифікатор на латентних векторах")
    print("=" * 60)

    set_seeds(config.RANDOM_SEED)

    Z_train, Z_val, Z_test, y_train, y_val, y_test = prepare_latents()
    model = train_classifier(Z_train, y_train, Z_val, y_val)
    test_metrics = evaluate(model, Z_test, y_test)
    print_metrics(test_metrics)
    save_metrics(test_metrics)
```

```
    print("\n" + "=" * 60)
    print("  Готово! Базовий pipeline повністю навчений.")
    print("=" * 60)


if __name__ == "__main__":
    main()

import os
import sys
import time
import csv

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from gensim.models import Word2Vec
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, balanced_accuracy_score, precision_score, recall_score,
    f1_score, matthews_corrcoef, roc_auc_score, cohen_kappa_score,
    average_precision_score,
)
from tqdm import tqdm

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config



class SparseAutoencoder(nn.Module):

    def __init__(self, input_dim=100, hidden_dim=64, latent_dim=32):
        super().__init__()
        self.enc_fc1 = nn.Linear(input_dim, hidden_dim)
        self.enc_fc2 = nn.Linear(hidden_dim, latent_dim)
        self.dec_fc1 = nn.Linear(latent_dim, hidden_dim)
        self.dec_fc2 = nn.Linear(hidden_dim, input_dim)
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()
        for m in [self.enc_fc1, self.enc_fc2, self.dec_fc1, self.dec_fc2]:
            nn.init.xavier_uniform_(m.weight); nn.init.zeros_(m.bias)

    def encode(self, x):
        return self.tanh(self.enc_fc2(self.relu(self.enc_fc1(x))))

    def decode(self, z):
        return self.dec_fc2(self.relu(self.dec_fc1(z)))   # Linear вихід —
для StandardScaler

    def forward(self, x):
        z = self.encode(x)
        return z, self.decode(z)


class MLPClassifier(nn.Module):
    def __init__(self, input_dim=32, hidden1=16, hidden2=8, dropout=0.3):
```

```
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden1), nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),  nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden2, 1), nn.Sigmoid(),
        )
    def forward(self, x): return self.net(x).squeeze(-1)



def set_seeds(seed):
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)


def stratified_subsample(df: pd.DataFrame, n: int, seed: int) ->
pd.DataFrame:

    if n >= len(df):
        return df.copy()
    sub, _ = train_test_split(df, train_size=n, stratify=df["target"],
random_state=seed)
    return sub.reset_index(drop=True)


def texts_to_vectors(texts: list[list[str]], wv, dim: int) -> np.ndarray:

    n = len(texts)
    X = np.zeros((n, dim), dtype=np.float32)
    for i, tokens in enumerate(texts):
        vectors = [wv[t] for t in tokens if t in wv]
        if vectors:
            X[i] = np.mean(vectors, axis=0)
    return X



def train_autoencoder_quick(X_train_s, X_val_s, device) -> SparseAutoencoder:
    train_ds = TensorDataset(torch.from_numpy(X_train_s))
    val_ds   = TensorDataset(torch.from_numpy(X_val_s))
    train_loader = DataLoader(train_ds, batch_size=config.AE_BATCH_SIZE,
shuffle=True)
    val_loader   = DataLoader(val_ds,   batch_size=config.AE_BATCH_SIZE,
shuffle=False)

    model = SparseAutoencoder(config.AE_INPUT_DIM, config.AE_HIDDEN_DIM,
config.AE_LATENT_DIM).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=config.AE_LR,
weight_decay=config.AE_LAMBDA)
    mse_fn = nn.MSELoss()
    noise_std = config.AE_NOISE_STD

    best_val = float("inf"); patience = 0; best_state = None
    for epoch in range(1, config.AE_EPOCHS + 1):
        model.train()
        for (x_batch,) in train_loader:
            x_batch = x_batch.to(device)
            # Денойзинг: додаємо шум на вхід, реконструюємо чистий оригінал
            noise = torch.randn_like(x_batch) * noise_std
            x_noisy = x_batch + noise
```

```
            optimizer.zero_grad()
            z, x_hat = model(x_noisy)
            loss = mse_fn(x_hat, x_batch)
            loss.backward(); optimizer.step()

        model.eval()
        val_loss = 0.0; n = 0
        with torch.no_grad():
            for (x_batch,) in val_loader:
                x_batch = x_batch.to(device)
                z, x_hat = model(x_batch)
                loss = mse_fn(x_hat, x_batch)
                val_loss += loss.item() * x_batch.size(0); n +=
x_batch.size(0)
        val_loss /= n

        if val_loss < best_val - 1e-6:
            best_val = val_loss; patience = 0
            best_state = {k: v.clone() for k, v in
model.state_dict().items()}
        else:
            patience += 1
            if patience >= config.AE_PATIENCE:
                break

    if best_state is not None:
        model.load_state_dict(best_state)
    return model



def train_classifier_quick(Z_train, y_train, Z_val, y_val, device) ->
MLPClassifier:
    train_ds = TensorDataset(torch.from_numpy(Z_train),
torch.from_numpy(y_train.astype(np.float32)))
    val_ds   = TensorDataset(torch.from_numpy(Z_val),
torch.from_numpy(y_val.astype(np.float32)))
    train_loader = DataLoader(train_ds, batch_size=config.CLF_BATCH_SIZE,
shuffle=True)
    val_loader   = DataLoader(val_ds,   batch_size=config.CLF_BATCH_SIZE,
shuffle=False)

    model = MLPClassifier(config.CLF_INPUT_DIM, config.CLF_HIDDEN_1,
config.CLF_HIDDEN_2, config.CLF_DROPOUT).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=config.CLF_LR)
    bce = nn.BCELoss()

    best_val = float("inf"); patience = 0; best_state = None
    for epoch in range(1, config.CLF_EPOCHS + 1):
        model.train()
        for x_batch, y_batch in train_loader:
            x_batch = x_batch.to(device); y_batch = y_batch.to(device)
            optimizer.zero_grad()
            loss = bce(model(x_batch), y_batch)
            loss.backward(); optimizer.step()

        model.eval()
        val_loss = 0.0; n = 0
        with torch.no_grad():
            for x_batch, y_batch in val_loader:
                x_batch = x_batch.to(device); y_batch = y_batch.to(device)
```

```
                val_loss += bce(model(x_batch), y_batch).item() *
x_batch.size(0)
                n += x_batch.size(0)
        val_loss /= n

        if val_loss < best_val - 1e-6:
            best_val = val_loss; patience = 0
            best_state = {k: v.clone() for k, v in
model.state_dict().items()}
        else:
            patience += 1
            if patience >= config.CLF_PATIENCE:
                break

    if best_state is not None:
        model.load_state_dict(best_state)
    return model



@torch.no_grad()
def evaluate(model, Z_test, y_test, device) -> dict:
    model.eval()
    probs = model(torch.from_numpy(Z_test).to(device)).cpu().numpy()
    preds = (probs > config.CLF_THRESHOLD).astype(int)
    y_int = y_test.astype(int)
    return {
        config.METRIC_NAMES["accuracy"]:  accuracy_score(y_int, preds),
        config.METRIC_NAMES["balanced"]:  balanced_accuracy_score(y_int,
preds),
        config.METRIC_NAMES["precision"]: precision_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["recall"]:    recall_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["f1"]:        f1_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["mcc"]:       matthews_corrcoef(y_int, preds),
        config.METRIC_NAMES["auc"]:       roc_auc_score(y_int, probs),
        config.METRIC_NAMES["ap"]:        average_precision_score(y_int,
probs),  # НОВА
        config.METRIC_NAMES["kappa"]:     cohen_kappa_score(y_int, preds),
    }



def run_iteration(n_train: int, train_df: pd.DataFrame, val_df: pd.DataFrame,
                  test_df: pd.DataFrame, device) -> dict:

    print(f"\n{'='*60}")
    print(f"  n_train = {n_train:,}")
    print(f"{'='*60}")

    set_seeds(config.RANDOM_SEED)


    sub = stratified_subsample(train_df, n_train, config.RANDOM_SEED)
    print(f"   Підвибірка: {len(sub):,} (баланс:
{sub['target'].mean():.3f})")


    print(f"   Навчання Word2Vec...")
```

```
t0 = time.time()
sentences = [text.split() for text in sub["text_clean"].tolist()]
# Для дуже малих n знижуємо min_count щоб не втрачати рідкісні слова
min_count = max(1, config.W2V_MIN_COUNT) if n_train >= 10_000 else 1
w2v = Word2Vec(
    sentences=sentences,
    vector_size=config.W2V_DIM,
    window=config.W2V_WINDOW,
    min_count=min_count,
    workers=config.W2V_WORKERS,
    sg=1,
    epochs=config.W2V_EPOCHS,
    seed=config.RANDOM_SEED,
)
print(f"      {time.time()-t0:.1f}s, словник: {len(w2v.wv):,}")


print(f"   Векторизація...")
t0 = time.time()
train_tokens = sentences
val_tokens   = [t.split() for t in val_df["text_clean"].tolist()]
test_tokens  = [t.split() for t in test_df["text_clean"].tolist()]
X_train = texts_to_vectors(train_tokens, w2v.wv, config.W2V_DIM)
X_val   = texts_to_vectors(val_tokens,   w2v.wv, config.W2V_DIM)
X_test  = texts_to_vectors(test_tokens,  w2v.wv, config.W2V_DIM)
print(f"      {time.time()-t0:.1f}s")


print(f"   Навчання денойзингового автоенкодера...")
t0 = time.time()
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train).astype(np.float32)
X_val_s   = scaler.transform(X_val).astype(np.float32)
X_test_s  = scaler.transform(X_test).astype(np.float32)
ae = train_autoencoder_quick(X_train_s, X_val_s, device)
print(f"      {time.time()-t0:.1f}s")


@torch.no_grad()
def to_latent(X_s):
    ae.eval()
    return ae.encode(torch.from_numpy(X_s).to(device)).cpu().numpy()

Z_train = to_latent(X_train_s)
Z_val   = to_latent(X_val_s)
Z_test  = to_latent(X_test_s)
y_train = sub["target"].to_numpy()
y_val   = val_df["target"].to_numpy()
y_test  = test_df["target"].to_numpy()


print(f"   Навчання класифікатора...")
t0 = time.time()
clf = train_classifier_quick(Z_train, y_train, Z_val, y_val, device)
print(f"      {time.time()-t0:.1f}s")


metrics = evaluate(clf, Z_test, y_test, device)
print(f"   F1 = {metrics[config.METRIC_NAMES['f1']]:.4f}, "
      f"Acc = {metrics[config.METRIC_NAMES['accuracy']]:.4f}, "
      f"AP = {metrics[config.METRIC_NAMES['ap']]:.4f}")
```

```
    return {"n_train": n_train, **metrics}


def main() -> None:
    print("=" * 60)
    print(" 07_experiment1.py — Learning Curve")
    print("=" * 60)


    train_path = config.TRAIN_CSV.replace(".csv", ".parquet")
    val_path   = config.VAL_CSV.replace(".csv", ".parquet")
    test_path  = config.TEST_CSV.replace(".csv", ".parquet")

    for p in [train_path, val_path, test_path]:
        if not os.path.exists(p):
            print(f" Не знайдено {p}. Запусти 02_preprocessing.py")
            sys.exit(1)

    print(" Завантаження train/val/test...")
    train_df = pd.read_parquet(train_path)
    val_df   = pd.read_parquet(val_path)
    test_df  = pd.read_parquet(test_path)
    print(f"   train: {len(train_df):,}, val: {len(val_df):,}, test:
{len(test_df):,}")

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"\n Пристрій: {device}")
    print(f"\n Розміри для експерименту: {config.EXP1_TRAIN_SIZES}")


    results = []
    total_start = time.time()
    for n_train in config.EXP1_TRAIN_SIZES:
        try:
            row = run_iteration(n_train, train_df, val_df, test_df, device)
            results.append(row)

            # Зберігаємо проміжний результат після КОЖНОЇ ітерації
            os.makedirs(config.RESULTS_TABLES, exist_ok=True)
            df = pd.DataFrame(results)
            df.to_csv(config.EXP1_RESULTS_CSV, index=False, encoding="utf-8")
            print(f"   Проміжне збереження: {config.EXP1_RESULTS_CSV}")
        except Exception as e:
            print(f"   Помилка для n_train={n_train}: {e}")
            continue

    total_time = time.time() - total_start
    print(f"\n  Загальний час: {total_time/60:.1f} хв ({total_time/3600:.2f}
год)")


    if len(results) >= 2:
        print("\n" + "=" * 60)
        print("  Пошук точки насичення n*")
        print("=" * 60)
        f1_key = config.METRIC_NAMES["f1"]
        for i in range(1, len(results)):
            n_prev = results[i-1]["n_train"]
            n_curr = results[i]["n_train"]
```

```
            f1_prev = results[i-1][f1_key]
            f1_curr = results[i][f1_key]
            delta = f1_curr - f1_prev
            mark = " ← насичення!" if delta < 0.005 else ""
            print(f"   n={n_prev:>8,} → {n_curr:>8,}: ΔF1 =
{delta:+.4f}{mark}")

    print("\n" + "=" * 60)
    print("  Експеримент 1 завершено")
    print(f"    Результати: {config.EXP1_RESULTS_CSV}")
    print("=" * 60)


if __name__ == "__main__":
    main()

import os
import sys
import time

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, balanced_accuracy_score, precision_score, recall_score,
    f1_score, matthews_corrcoef, roc_auc_score, cohen_kappa_score,
    average_precision_score, confusion_matrix,
)

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config



class MLPClassifier(nn.Module):
    def __init__(self, input_dim=32, hidden1=16, hidden2=8, dropout=0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden1), nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),  nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden2, 1), nn.Sigmoid(),
        )
    def forward(self, x): return self.net(x).squeeze(-1)



def compute_p1_q0(y_true: np.ndarray, y_pred: np.ndarray) -> tuple[float,
float]:

    cm = confusion_matrix(y_true, y_pred, labels=[0, 1])
    tn, fp, fn, tp = cm.ravel()
    p1 = tp / (tp + fn) if (tp + fn) > 0 else float("nan")
    q0 = fp / (fp + tn) if (fp + tn) > 0 else float("nan")
    return p1, q0
```

```
def compute_all_metrics(y_true: np.ndarray, probs: np.ndarray) -> dict:

    preds = (probs > config.CLF_THRESHOLD).astype(int)
    y_int = y_true.astype(int)
    p1, q0 = compute_p1_q0(y_int, preds)

    return {
        "p1":  p1,
        "q0":  q0,
        config.METRIC_NAMES["accuracy"]:  accuracy_score(y_int, preds),
        config.METRIC_NAMES["balanced"]:  balanced_accuracy_score(y_int,
preds),
        config.METRIC_NAMES["precision"]: precision_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["recall"]:    recall_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["f1"]:        f1_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["mcc"]:       matthews_corrcoef(y_int, preds),
        config.METRIC_NAMES["auc"]:       roc_auc_score(y_int, probs),
        config.METRIC_NAMES["ap"]:        average_precision_score(y_int,
probs),
        config.METRIC_NAMES["kappa"]:     cohen_kappa_score(y_int, preds),
    }



def stratified_subsample_indices(y: np.ndarray, n: int, seed: int) ->
np.ndarray:
    if n >= len(y):
        return np.arange(len(y))
    idx_all = np.arange(len(y))
    idx_sub, _ = train_test_split(idx_all, train_size=n, stratify=y,
random_state=seed)
    return idx_sub



def delta_chebyshev(n: int) -> float:

    n1 = n / 2.0
    return config.EXP2_K_CHEBYSHEV / np.sqrt(n1)


def delta_hoeffding(n: int) -> float:

    n1 = n / 2.0
    return config.EXP2_K_HOEFFDING / np.sqrt(n1)


def delta_wald(n: int, p_hat: float) -> float:

    n1 = n / 2.0
    p = max(min(p_hat, 1.0), 0.0)
    return config.EXP2_Z_WALD * np.sqrt(p * (1 - p) / n1)


def delta_wald_conservative(n: int) -> float:

    n1 = n / 2.0
    return config.EXP2_K_WALD_CONS / np.sqrt(n1)
```

```
def load_model_and_data():
    if not os.path.exists(config.CLF_MODEL_PATH):
        print(f" Не знайдено {config.CLF_MODEL_PATH}. Запусти
06_classifier.py")
        sys.exit(1)
    if not os.path.exists(config.LATENT_TEST):
        print(f" Не знайдено {config.LATENT_TEST}. Запусти 06_classifier.py")
        sys.exit(1)

    print(" Завантаження латентів та класифікатора...")
    Z_test = np.load(config.LATENT_TEST)
    y_test = np.load(config.VEC_TEST_Y).astype(np.int64)
    print(f"   Z_test: {Z_test.shape}, y_test: {y_test.shape}")
    print(f"   Баланс класів y test: {y_test.mean():.4f}")

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"   Пристрій: {device}")

    model = MLPClassifier(
        input_dim=config.CLF_INPUT_DIM,
        hidden1=config.CLF_HIDDEN_1,
        hidden2=config.CLF_HIDDEN_2,
        dropout=config.CLF_DROPOUT,
    ).to(device)
    model.load_state_dict(torch.load(config.CLF_MODEL_PATH,
map_location=device))
    model.eval()
    print(f"    Класифікатор завантажено")
    return model, Z_test, y_test, device


@torch.no_grad()
def predict_all(model, Z, device, batch_size=4096) -> np.ndarray:
    model.eval()
    n = Z.shape[0]
    probs = np.zeros(n, dtype=np.float32)
    for start in range(0, n, batch_size):
        end = min(start + batch_size, n)
        x_batch = torch.from_numpy(Z[start:end]).to(device)
        probs[start:end] = model(x_batch).cpu().numpy()
    return probs



def run_experiment(probs_full: np.ndarray, y_test: np.ndarray) ->
pd.DataFrame:

    all_records = []
    for n in config.EXP2_TEST_SIZES:
        print(f"\n  n = {n:>7,}  ({config.EXP2_BOOTSTRAPS} повторень)")
        for b in tqdm(range(config.EXP2_BOOTSTRAPS), desc="  bootstrap"):
            seed = config.RANDOM_SEED + b
            idx = stratified_subsample_indices(y_test, n, seed)
            y_sub     = y_test[idx]
            probs_sub = probs_full[idx]
            metrics = compute_all_metrics(y_sub, probs_sub)
            metrics["n"] = n
            metrics["bootstrap"] = b
```

```
            all_records.append(metrics)
    return pd.DataFrame(all_records)


def aggregate_by_n(df_raw: pd.DataFrame) -> pd.DataFrame:
    metric_cols = [c for c in df_raw.columns if c not in ("n", "bootstrap")]

    grouped = df_raw.groupby("n")[metric_cols].agg(["mean", "std"])
    grouped.columns = [f"{col}_{stat}" for col, stat in grouped.columns]
    grouped = grouped.reset_index().sort_values("n").reset_index(drop=True)


    grouped["Δ_Чебишов"]    = grouped["n"].apply(delta_chebyshev)
    grouped["Δ_Хеффдінг"]   = grouped["n"].apply(delta_hoeffding)
    grouped["Δ_Вальд_конс"] = grouped["n"].apply(delta_wald_conservative)
    # Точний інтервал Вальда — використовує середнє p^₁ для цього n
    grouped["Δ_Вальд"]      = grouped.apply(
        lambda row: delta_wald(int(row["n"]), float(row["p1_mean"])), axis=1
    )


    p1_final = grouped.iloc[-1]["p1_mean"]
    q0_final = grouped.iloc[-1]["q0_mean"]
    grouped["δ₁_%"] = np.abs(grouped["p1_mean"] - p1_final) / p1_final * 100
    grouped["δ₀_%"] = np.abs(grouped["q0_mean"] - q0_final) / q0_final * 100


    grouped["v₁"] = grouped["p1_mean"].diff().abs()
    grouped["v₀"] = grouped["q0_mean"].diff().abs()

    return grouped



def compute_coverage(df_raw: pd.DataFrame, p1_true: float, q0_true: float) ->
pd.DataFrame:

    records = []
    for n in sorted(df_raw["n"].unique()):
        sub = df_raw[df_raw["n"] == n]
        p1_hats = sub["p1"].values
        q0_hats = sub["q0"].values

        delta_cheb  = delta_chebyshev(n)
        delta_hoef  = delta_hoeffding(n)
        delta_wald_arr = np.array([delta_wald(int(n), float(p)) for p in
p1_hats])
        delta_wald_q0  = np.array([delta_wald(int(n), float(q)) for q in
q0_hats])

        # p₁ coverage
        cp_p1_cheb = np.mean(np.abs(p1_hats - p1_true) <= delta_cheb)
        cp_p1_hoef = np.mean(np.abs(p1_hats - p1_true) <= delta_hoef)
        cp_p1_wald = np.mean(np.abs(p1_hats - p1_true) <= delta_wald_arr)

        # q₀ coverage
        cp_q0_cheb = np.mean(np.abs(q0_hats - q0_true) <= delta_cheb)
        cp_q0_hoef = np.mean(np.abs(q0_hats - q0_true) <= delta_hoef)
        cp_q0_wald = np.mean(np.abs(q0_hats - q0_true) <= delta_wald_q0)
```

```
        records.append({
            "n": int(n),
            "CP_p1_Чебишов":  cp_p1_cheb,
            "CP_p1_Хеффдінг": cp_p1_hoef,
            "CP_p1_Вальд":    cp_p1_wald,
            "CP_q0_Чебишов":  cp_q0_cheb,
            "CP_q0_Хеффдінг": cp_q0_hoef,
            "CP_q0_Вальд":    cp_q0_wald,
        })

    return pd.DataFrame(records)



def print_summary(df: pd.DataFrame) -> None:
    print("\n" + "=" * 110)
    print("  ПІДСУМКОВА ТАБЛИЦЯ: стохастичний аналіз з трьома теоретичними
межами")
    print("=" * 110)
    print(f"{'n':>8} | {'p^₁':>8} {'σ(p^₁)':>9} | {'Δ_Чеб':>9} {'Δ_Хеф':>9}
{'Δ_Вальд':>9} | "
          f"{'δ₁ %':>7} | {'F1':>7}")
    print("-" * 110)
    f1_col = f"{config.METRIC_NAMES['f1']}_mean"
    for _, row in df.iterrows():
        print(f"{int(row['n']):>8,} | "
              f"{row['p1_mean']:>8.4f} {row['p1_std']:>9.4f} | "
              f"{row['Δ_Чебишов']:>9.4f} {row['Δ_Хеффдінг']:>9.4f}
{row['Δ_Вальд']:>9.4f} | "
              f"{row['δ₁_%']:>6.2f}% | {row[f1_col]:>7.4f}")


def print_coverage(df_cov: pd.DataFrame) -> None:
    print("\n" + "=" * 80)
    print("  ЙМОВІРНІСТЬ ПОКРИТТЯ (Coverage Probability) при γ = 0.95")
    print("=" * 80)
    print(f"{'n':>8} | {'p₁: Чеб':>8} {'p₁: Хеф':>8} {'p₁: Вальд':>10} | "
          f"{'q₀: Чеб':>8} {'q₀: Хеф':>8} {'q₀: Вальд':>10}")
    print("-" * 80)
    for _, row in df_cov.iterrows():
        print(f"{int(row['n']):>8,} | "
              f"{row['CP_p1_Чебишов']:>8.3f} {row['CP_p1_Хеффдінг']:>8.3f}
{row['CP_p1_Вальд']:>10.3f} | "
              f"{row['CP_q0_Чебишов']:>8.3f} {row['CP_q0_Хеффдінг']:>8.3f}
{row['CP_q0_Вальд']:>10.3f}")

    print("\nІнтерпретація:")
    print("  • CP ≥ 0.95 — теоретична межа виконується (або вища за
номінал)")
    print("  • CP ≈ 1.00 — межа КОНСЕРВАТИВНА (реально точніша за
гарантовану)")
    print("  • CP < 0.95 — порушення припущень теореми")



def main() -> None:
    print("=" * 60)
    print(" 08_experiment2.py — Стохастичний аналіз збіжності ")
    print("=" * 60)
```

```
    print(f"\n Параметри:")
    print(f"   Розміри n_test:   {config.EXP2_TEST_SIZES}")
    print(f"   Bootstrap-ів:     {config.EXP2_BOOTSTRAPS}")
    print(f"   Рівень довіри γ:  {config.EXP2_GAMMA}")
    print(f"\n   Теоретичні коефіцієнти (Δ = K/√n₁):")
    print(f"   K_Чебишов:        {config.EXP2_K_CHEBYSHEV:.4f}")
    print(f"   K_Хеффдінг:       {config.EXP2_K_HOEFFDING:.4f}")
    print(f"   K_Вальд (конс.):  {config.EXP2_K_WALD_CONS:.4f}")

    model, Z_test, y_test, device = load_model_and_data()

    print("\n Прогін класифікатора по всьому test...")
    t0 = time.time()
    probs_full = predict_all(model, Z_test, device)
    print(f"    {time.time()-t0:.1f}s")

    print("\n Bootstrap-аналіз...")
    start = time.time()
    df_raw = run_experiment(probs_full, y_test)
    print(f"\n  Bootstrap зайняв: {(time.time()-start)/60:.2f} хв")


    os.makedirs(config.RESULTS_TABLES, exist_ok=True)
    raw_path = config.EXP2_RESULTS_CSV.replace(".csv", "_raw.csv")
    df_raw.to_csv(raw_path, index=False, encoding="utf-8")
    print(f"\n Сирі дані: {raw_path}")


    df_summary = aggregate_by_n(df_raw)
    df_summary.to_csv(config.EXP2_RESULTS_CSV, index=False, encoding="utf-8")
    print(f" Агрегована таблиця: {config.EXP2_RESULTS_CSV}")



    p1_true = float(df_summary.iloc[-1]["p1_mean"])
    q0_true = float(df_summary.iloc[-1]["q0_mean"])
    print(f"\n Опорні значення (з повного test):")
    print(f"   p₁ = {p1_true:.6f}")
    print(f"   q₀ = {q0_true:.6f}")

    df_coverage = compute_coverage(df_raw, p1_true, q0_true)
    df_coverage.to_csv(config.EXP2_COVERAGE_CSV, index=False, encoding="utf-
8")
    print(f" Coverage Probability: {config.EXP2_COVERAGE_CSV}")


    print_summary(df_summary)
    print_coverage(df_coverage)

    print("\n" + "=" * 60)
    print("  Експеримент 2 завершено!")
    print("    Це ключові результати для розділу 2 диплому.")
    print("=" * 60)


if __name__ == "__main__":
    main()

import os
```

```
import sys
import time
import csv

import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import (
    accuracy_score, balanced_accuracy_score, precision_score, recall_score,
    f1_score, matthews_corrcoef, roc_auc_score, cohen_kappa_score,
    average_precision_score,
)

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config


def set_seeds(seed):
    np.random.seed(seed); torch.manual_seed(seed);
torch.cuda.manual_seed_all(seed)



class MLPClassifierLatent(nn.Module):

    def __init__(self, input_dim=32, hidden1=16, hidden2=8, dropout=0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden1), nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),  nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden2, 1), nn.Sigmoid(),
        )
    def forward(self, x): return self.net(x).squeeze(-1)


class MLPClassifierDirect(nn.Module):

    def __init__(self, input_dim=100, hidden1=16, hidden2=8, dropout=0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden1), nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),  nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden2, 1), nn.Sigmoid(),
        )
    def forward(self, x): return self.net(x).squeeze(-1)



@torch.no_grad()
def evaluate_model(model, X, y, device, batch_size=4096) -> dict:
    model.eval()
    n = X.shape[0]
    probs = np.zeros(n, dtype=np.float32)
    for start in range(0, n, batch_size):
        end = min(start + batch_size, n)
        x_batch = torch.from_numpy(X[start:end]).to(device)
        probs[start:end] = model(x_batch).cpu().numpy()
    preds = (probs > config.CLF_THRESHOLD).astype(int)
    y_int = y.astype(int)
```

```
    return {
        config.METRIC_NAMES["accuracy"]:  accuracy_score(y_int, preds),
        config.METRIC_NAMES["balanced"]:  balanced_accuracy_score(y_int,
preds),
        config.METRIC_NAMES["precision"]: precision_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["recall"]:    recall_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["f1"]:        f1_score(y_int, preds,
zero_division=0),
        config.METRIC_NAMES["mcc"]:       matthews_corrcoef(y_int, preds),
        config.METRIC_NAMES["auc"]:       roc_auc_score(y_int, probs),
        config.METRIC_NAMES["ap"]:        average_precision_score(y_int,
probs),  # НОВА
        config.METRIC_NAMES["kappa"]:     cohen_kappa_score(y_int, preds),
    }



def train_pipeline_b(X_train_s, y_train, X_val_s, y_val, device) ->
MLPClassifierDirect:
    train_ds = TensorDataset(torch.from_numpy(X_train_s),
torch.from_numpy(y_train.astype(np.float32)))
    val_ds   = TensorDataset(torch.from_numpy(X_val_s),
torch.from_numpy(y_val.astype(np.float32)))
    train_loader = DataLoader(train_ds, batch_size=config.CLF_BATCH_SIZE,
shuffle=True)
    val_loader   = DataLoader(val_ds,   batch_size=config.CLF_BATCH_SIZE,
shuffle=False)

    model = MLPClassifierDirect(
        input_dim=config.AE_INPUT_DIM,
        hidden1=config.CLF_HIDDEN_1, hidden2=config.CLF_HIDDEN_2,
dropout=config.CLF_DROPOUT,
    ).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=config.CLF_LR)
    bce = nn.BCELoss()

    n_params = sum(p.numel() for p in model.parameters())
    print(f" Параметрів у Pipeline B: {n_params:,}")
    print(f" Навчання Pipeline B (макс {config.CLF_EPOCHS} епох)\n")

    best_val = float("inf"); patience = 0; history = []; best_state = None
    for epoch in range(1, config.CLF_EPOCHS + 1):
        t0 = time.time()
        model.train()
        for x_batch, y_batch in train_loader:
            x_batch = x_batch.to(device); y_batch = y_batch.to(device)
            optimizer.zero_grad()
            loss = bce(model(x_batch), y_batch)
            loss.backward(); optimizer.step()
        model.eval()
        val_loss = val_n = 0
        with torch.no_grad():
            for x_batch, y_batch in val_loader:
                x_batch = x_batch.to(device); y_batch = y_batch.to(device)
                val_loss += bce(model(x_batch), y_batch).item() *
x_batch.size(0)
                val_n += x_batch.size(0)
        val_loss /= val_n
        dt = time.time() - t0
```

```
        history.append({"epoch": epoch, "val_loss": val_loss})
        print(f"Epoch {epoch:02d}/{config.CLF_EPOCHS} ({dt:4.1f}s)
val_loss={val_loss:.4f}")
        if val_loss < best_val - 1e-6:
            best_val = val_loss; patience = 0
            best_state = {k: v.clone() for k, v in
model.state_dict().items()}
        else:
            patience += 1
            if patience >= config.CLF_PATIENCE:
                print(f"\n Early stopping на епосі {epoch}")
                break
    if best_state is not None:
        model.load_state_dict(best_state)

    history_path = os.path.join(config.RESULTS_TABLES,
"pipeline_b_history.csv")
    with open(history_path, "w", newline="") as f:
        w = csv.DictWriter(f, fieldnames=history[0].keys())
        w.writeheader(); w.writerows(history)
    print(f"\n Історія Pipeline B: {history_path}")

    b_path = os.path.join(config.MODELS_DIR, "classifier_baseline.pt")
    torch.save(model.state_dict(), b_path)
    print(f" Модель Pipeline B: {b_path}")
    return model



def evaluate_pipeline_a(device) -> dict:
    if not os.path.exists(config.CLF_MODEL_PATH) or not
os.path.exists(config.LATENT_TEST):
        print(f" Pipeline A не готовий. Запусти 06_classifier.py")
        sys.exit(1)
    print(" Завантаження Pipeline A (DAE + класифікатор)...")
    Z_test = np.load(config.LATENT_TEST)
    y_test = np.load(config.VEC_TEST_Y).astype(np.int64)
    model = MLPClassifierLatent(
        input_dim=config.CLF_INPUT_DIM,
        hidden1=config.CLF_HIDDEN_1, hidden2=config.CLF_HIDDEN_2,
dropout=config.CLF_DROPOUT,
    ).to(device)
    model.load_state_dict(torch.load(config.CLF_MODEL_PATH,
map_location=device))
    print(" Оцінка Pipeline A на test...")
    return evaluate_model(model, Z_test, y_test, device)



def evaluate_pipeline_b(device) -> dict:
    set_seeds(config.RANDOM_SEED)

    print(" Завантаження W2V векторів...")
    X_train = np.load(config.VEC_TRAIN_X); X_val = np.load(config.VEC_VAL_X);
X_test = np.load(config.VEC_TEST_X)
    y_train = np.load(config.VEC_TRAIN_Y); y_val = np.load(config.VEC_VAL_Y);
y_test = np.load(config.VEC_TEST_Y)


    print(" StandardScaler (та сама нормалізація що й у Pipeline A)...")
    import joblib
```

```
    scaler = joblib.load(config.AE_SCALER_PATH)
    X_train_s = scaler.transform(X_train).astype(np.float32)
    X_val_s   = scaler.transform(X_val).astype(np.float32)
    X_test_s  = scaler.transform(X_test).astype(np.float32)

    model = train_pipeline_b(X_train_s, y_train, X_val_s, y_val, device)
    print(" Оцінка Pipeline B на test...")
    return evaluate_model(model, X_test_s, y_test, device)



def print_comparison(a, b) -> None:
    print("\n" + "=" * 80)
    print("  Порівняння Pipeline A (DAE+MLP) vs Pipeline B (W2V→MLP
напряму)")
    print("=" * 80)
    print(f"{'Метрика':<44} {'Pipeline A':>12} {'Pipeline B':>12} {'Δ
(A−B)':>12}")
    print("-" * 80)
    for name in a.keys():
        diff = a[name] - b[name]
        sign = "□" if diff > 0 else ("●" if diff < 0 else "○")
        print(f"{name:<44} {a[name]:>12.4f} {b[name]:>12.4f} {diff:>+11.4f}
{sign}")


def save_comparison(a, b) -> None:
    os.makedirs(config.RESULTS_TABLES, exist_ok=True)
    with open(config.EXP3_RESULTS_CSV, "w", newline="", encoding="utf-8") as
f:
        w = csv.writer(f)
        w.writerow(["Метрика", "Pipeline A (DAE+MLP)", "Pipeline B
(W2V→MLP)", "Δ (A−B)"])
        for name in a.keys():
            w.writerow([name, f"{a[name]:.6f}", f"{b[name]:.6f}", f"{a[name]-
b[name]:+.6f}"])
    print(f"\n {config.EXP3_RESULTS_CSV}")



def main() -> None:
    print("=" * 60)
    print(" 09_comparison.py — Pipeline з DAE vs без")
    print("=" * 60)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f" Пристрій: {device}\n")

    metrics_a = evaluate_pipeline_a(device)
    print(" Pipeline A оцінено")
    print("\n" + "-" * 60)
    metrics_b = evaluate_pipeline_b(device)
    print(" Pipeline B оцінено")

    print_comparison(metrics_a, metrics_b)
    save_comparison(metrics_a, metrics_b)

    print("\n" + "=" * 60)
    print("  Готово!")
    print("=" * 60)
```

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


import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import torch
import torch.nn as nn
from sklearn.metrics import roc_curve, auc

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import config


plt.style.use("seaborn-v0_8")
matplotlib.rcParams.update({
    "font.family": "DejaVu Sans",
    "font.size":         12,
    "axes.titlesize":    14,
    "axes.labelsize":    12,
    "legend.fontsize":   11,
    "xtick.labelsize":   11,
    "ytick.labelsize":   11,
    "figure.dpi":        100,
    "savefig.dpi":       300,
    "savefig.bbox":      "tight",
})

FIG_SIZE = (10, 6)


def save(name: str) -> None:
    os.makedirs(config.RESULTS_FIGS, exist_ok=True)
    path = os.path.join(config.RESULTS_FIGS, name)
    plt.savefig(path); plt.close()
    print(f"    {path}")



class MLPClassifier(nn.Module):
    def __init__(self, input_dim=32, hidden1=16, hidden2=8, dropout=0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden1), nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden1, hidden2),  nn.ReLU(), nn.Dropout(dropout),
            nn.Linear(hidden2, 1), nn.Sigmoid(),
        )
    def forward(self, x): return self.net(x).squeeze(-1)



def plot_p1_q0(df):
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    ax.plot(df["n"], df["p1_mean"], "o-", color="#1f77b4",
            label=r"$\hat{p}_1^{(n)}$ — оцінка $P(\hat{y}{=}1 | y{=}1)$")
```

```
    ax.fill_between(df["n"], df["p1_mean"]-df["p1_std"],
df["p1_mean"]+df["p1_std"],
                    alpha=0.2, color="#1f77b4", label=r"$\pm 1\sigma$ ($p_1$)")
    ax.plot(df["n"], df["q0_mean"], "s-", color="#d62728",
            label=r"$\hat{q}_0^{(n)}$ — оцінка $P(\hat{y}{=}1 | y{=}0)$")
    ax.fill_between(df["n"], df["q0_mean"]-df["q0_std"],
df["q0_mean"]+df["q0_std"],
                    alpha=0.2, color="#d62728", label=r"$\pm 1\sigma$ ($q_0$)")
    ax.set_xscale("log")
    ax.set_xlabel("Розмір тестової вибірки, n")
    ax.set_ylabel("Значення оцінки")
    ax.set_title("Стохастична збіжність оцінок $p_1$ та $q_0$")
    ax.legend(loc="best"); ax.grid(True, which="both", alpha=0.3)
    save("01_convergence_p1_q0.png")



def plot_recall(df):
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    col_m = f"{config.METRIC_NAMES['recall']}_mean"
    col_s = f"{config.METRIC_NAMES['recall']}_std"
    asy = df.iloc[-1][col_m]
    ax.plot(df["n"], df[col_m], "o-", color="#2ca02c", label="Повнота
(Recall)")
    ax.fill_between(df["n"], df[col_m]-df[col_s], df[col_m]+df[col_s],
alpha=0.2, color="#2ca02c", label=r"$\pm 1\sigma$")
    ax.axhline(asy, ls="--", color="black", alpha=0.5, label=f"Асимптота =
{asy:.4f}")
    ax.set_xscale("log")
    ax.set_xlabel("n"); ax.set_ylabel("Повнота (Recall)")
    ax.set_title("Збіжність Повноти (Recall) при $n \\to \\infty$")
    ax.legend(); ax.grid(True, which="both", alpha=0.3)
    save("02_convergence_recall.png")


def plot_precision(df):
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    col_m = f"{config.METRIC_NAMES['precision']}_mean"
    col_s = f"{config.METRIC_NAMES['precision']}_std"
    asy = df.iloc[-1][col_m]
    ax.plot(df["n"], df[col_m], "o-", color="#ff7f0e", label="Точність
(Precision)")
    ax.fill_between(df["n"], df[col_m]-df[col_s], df[col_m]+df[col_s],
alpha=0.2, color="#ff7f0e", label=r"$\pm 1\sigma$")
    ax.axhline(asy, ls="--", color="black", alpha=0.5, label=f"Асимптота =
{asy:.4f}")
    ax.set_xscale("log")
    ax.set_xlabel("n"); ax.set_ylabel("Точність (Precision)")
    ax.set_title("Збіжність Точності (Precision) при $n \\to \\infty$")
    ax.legend(); ax.grid(True, which="both", alpha=0.3)
    save("03_convergence_precision.png")



def plot_three_bounds(df):
    """Чебишов vs Хеффдінг vs Вальд vs σ емпір — у log-log шкалі."""
    fig, ax = plt.subplots(figsize=FIG_SIZE)

    if "Δ_Чебишов" in df.columns:
        ax.plot(df["n"], df["Δ_Чебишов"], "D-", color="#9467bd", lw=2,
                label=r"Чебишов $\Delta = 2.236/\sqrt{n_1}$")
```

```
    if "Δ_Хеффдінг" in df.columns:
        ax.plot(df["n"], df["Δ_Хеффдінг"], "^-", color="#8c564b", lw=2,
                label=r"Хеффдінг $\Delta = 1.358/\sqrt{n_1}$")
    if "Δ_Вальд_конс" in df.columns:
        ax.plot(df["n"], df["Δ_Вальд_конс"], "v-", color="#17becf", lw=2,
                label=r"Вальд (конс.) $\Delta = 0.98/\sqrt{n_1}$")
    if "Δ_Вальд" in df.columns:
        ax.plot(df["n"], df["Δ_Вальд"], "*-", color="#e377c2", lw=2,
                label=r"Вальд (точний)")

    ax.plot(df["n"], df["p1_std"], "o--", color="#1f77b4", lw=2,
            label=r"Емпірична $\sigma(\hat{p}_1^{(n)})$")

    ax.set_xscale("log"); ax.set_yscale("log")
    ax.set_xlabel("n (log-шкала)")
    ax.set_ylabel("Ширина довірчого інтервалу (log-шкала)")
    ax.set_title(rf"Порівняння трьох теоретичних меж довірчого інтервалу
($\gamma = {config.EXP2_GAMMA}$)")
    ax.legend(loc="best"); ax.grid(True, which="both", alpha=0.3)
    save("04_delta_three_bounds.png")



def plot_metrics_overview(df):
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    metrics = [
        ("f1",       "F1-міра (F1)",                    "#1f77b4"),
        ("accuracy", "Загальна точність (Acc)",         "#2ca02c"),
        ("balanced", "Збалансована точність (BA)",      "#ff7f0e"),
        ("mcc",      "Коеф. кореляції Метьюса (MCC)",   "#d62728"),
        ("ap",       "Середня точність (AP)",           "#9467bd"),
    ]
    for key, label, color in metrics:
        col_m = f"{config.METRIC_NAMES[key]}_mean"
        col_s = f"{config.METRIC_NAMES[key]}_std"
        if col_m not in df.columns:
            continue
        ax.plot(df["n"], df[col_m], "o-", color=color, label=label)
        ax.fill_between(df["n"], df[col_m]-df[col_s], df[col_m]+df[col_s],
alpha=0.12, color=color)
    ax.set_xscale("log")
    ax.set_xlabel("n"); ax.set_ylabel("Значення метрики")
    ax.set_title("Збіжність основних метрик класифікації при зростанні $n$")
    ax.legend(); ax.grid(True, which="both", alpha=0.3)
    save("05_metrics_overview.png")



def plot_roc():
    if not os.path.exists(config.CLF_MODEL_PATH) or not
os.path.exists(config.LATENT_TEST):
        print("     Пропущено ROC: запусти 06_classifier.py")
        return
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    Z_test = np.load(config.LATENT_TEST)
    y_test = np.load(config.VEC_TEST_Y).astype(int)
    model = MLPClassifier(config.CLF_INPUT_DIM, config.CLF_HIDDEN_1,
config.CLF_HIDDEN_2, config.CLF_DROPOUT).to(device)
    model.load_state_dict(torch.load(config.CLF_MODEL_PATH,
map_location=device))
    model.eval()
```

```
    with torch.no_grad():
        probs = np.zeros(len(Z_test), dtype=np.float32); bs = 4096
        for s in range(0, len(Z_test), bs):
            e = min(s+bs, len(Z_test))
            probs[s:e] =
model(torch.from_numpy(Z_test[s:e]).to(device)).cpu().numpy()
    fpr, tpr, _ = roc_curve(y_test, probs); roc_auc = auc(fpr, tpr)
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    ax.plot(fpr, tpr, "-", color="#1f77b4", lw=2, label=f"ROC (AUC =
{roc_auc:.4f})")
    ax.plot([0,1], [0,1], "k--", alpha=0.4, label="Випадковий класифікатор")
    ax.set_xlabel("Хибнопозитивні (FPR)"); ax.set_ylabel("Істиннопозитивні
(TPR)")
    ax.set_title("ROC-крива на повному test")
    ax.legend(loc="lower right"); ax.grid(True, alpha=0.3)
    save("06_roc_curve.png")



def plot_learning_curve():
    if not os.path.exists(config.EXP1_RESULTS_CSV):
        print("     Пропущено Learning Curve: запусти 07_experiment1.py")
        return
    df = pd.read_csv(config.EXP1_RESULTS_CSV)
    f1_col = config.METRIC_NAMES["f1"]; acc_col =
config.METRIC_NAMES["accuracy"]
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    ax.plot(df["n_train"], df[f1_col], "o-", color="#1f77b4", lw=2, ms=8,
label="F1-міра (F1)")
    ax.plot(df["n_train"], df[acc_col], "s-", color="#2ca02c", lw=2, ms=8,
label="Загальна точність (Acc)")
    ax.set_xscale("log")
    ax.set_xlabel("$n_{train}$ (log-шкала)"); ax.set_ylabel("Значення
метрики")
    ax.set_title("Learning Curve")
    ax.legend(); ax.grid(True, which="both", alpha=0.3)
    save("07_learning_curve.png")



def plot_pipeline_comparison():
    if not os.path.exists(config.EXP3_RESULTS_CSV):
        print("     Пропущено порівняння: запусти 09_comparison.py")
        return
    df = pd.read_csv(config.EXP3_RESULTS_CSV)
    metrics = df["Метрика"].tolist()
    a_vals = df.iloc[:, 1].astype(float).tolist()
    b_vals = df.iloc[:, 2].astype(float).tolist()
    x = np.arange(len(metrics)); width = 0.38
    fig, ax = plt.subplots(figsize=(13, 6))
    ax.bar(x-width/2, a_vals, width, label="Pipeline A (DAE+MLP)",
color="#1f77b4")
    ax.bar(x+width/2, b_vals, width, label="Pipeline B (W2V→MLP)",
color="#ff7f0e")
    for i, (a, b) in enumerate(zip(a_vals, b_vals)):
        ax.text(i-width/2, a+0.01, f"{a:.3f}", ha="center", fontsize=9)
        ax.text(i+width/2, b+0.01, f"{b:.3f}", ha="center", fontsize=9)
    short_names = [m.split("(")[-1].replace(")", "") for m in metrics]
    ax.set_xticks(x); ax.set_xticklabels(short_names, rotation=0)
    ax.set_ylabel("Значення метрики")
    ax.set_title("Порівняння двох pipeline'ів")
```

```
    ax.set_ylim(0, max(max(a_vals), max(b_vals)) * 1.10)
    ax.legend(); ax.grid(True, axis="y", alpha=0.3)
    save("08_pipeline_comparison.png")



def plot_autoencoder_loss():
    path = os.path.join(config.RESULTS_TABLES, "autoencoder_history.csv")
    if not os.path.exists(path):
        print("     Пропущено autoencoder_loss"); return
    df = pd.read_csv(path)
    fig, ax = plt.subplots(figsize=FIG_SIZE)
    ax.plot(df["epoch"], df["train_loss"], "o-", color="#1f77b4",
label="Train loss")
    ax.plot(df["epoch"], df["val_loss"], "s-", color="#d62728",
label="Validation loss")
    ax.set_xlabel("Епоха"); ax.set_ylabel("Loss (MSE денойзингу)")
    ax.set_title("Динаміка навчання денойзингового автоенкодера")
    ax.legend(); ax.grid(True, alpha=0.3)
    save("09_autoencoder_loss.png")


def plot_classifier_loss():
    path = os.path.join(config.RESULTS_TABLES, "classifier_history.csv")
    if not os.path.exists(path):
        print("     Пропущено classifier_loss"); return
    df = pd.read_csv(path)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
    ax1.plot(df["epoch"], df["train_loss"], "o-", color="#1f77b4",
label="Train")
    ax1.plot(df["epoch"], df["val_loss"], "s-", color="#d62728",
label="Validation")
    ax1.set_xlabel("Епоха"); ax1.set_ylabel("BCE Loss")
    ax1.set_title("Функція втрат класифікатора")
    ax1.legend(); ax1.grid(True, alpha=0.3)
    ax2.plot(df["epoch"], df["train_acc"], "o-", color="#1f77b4",
label="Train")
    ax2.plot(df["epoch"], df["val_acc"], "s-", color="#d62728",
label="Validation")
    ax2.set_xlabel("Епоха"); ax2.set_ylabel("Загальна точність (Acc)")
    ax2.set_title("Точність класифікатора")
    ax2.legend(); ax2.grid(True, alpha=0.3)
    save("10_classifier_loss.png")



def plot_coverage():
    if not os.path.exists(config.EXP2_COVERAGE_CSV):
        print("    Пропущено coverage: запусти 08_experiment2.py"); return
    df = pd.read_csv(config.EXP2_COVERAGE_CSV)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))


    ax1.plot(df["n"], df["CP_p1_Чебишов"], "D-", color="#9467bd", lw=2, ms=8,
label="Чебишов")
    ax1.plot(df["n"], df["CP_p1_Хеффдінг"], "^-", color="#8c564b", lw=2,
ms=8, label="Хеффдінг")
    ax1.plot(df["n"], df["CP_p1_Вальд"], "*-", color="#e377c2", lw=2, ms=10,
label="Вальд")
    ax1.axhline(config.EXP2_GAMMA, ls="--", color="black", alpha=0.6,
```

```
                label=f"Номінальне γ = {config.EXP2_GAMMA}")
    ax1.set_xscale("log")
    ax1.set_xlabel("n (log-шкала)")
    ax1.set_ylabel("Coverage Probability")
    ax1.set_title("Ймовірність покриття для $p_1$")
    ax1.set_ylim(-0.05, 1.10)
    ax1.legend(loc="lower right"); ax1.grid(True, which="both", alpha=0.3)

    # Coverage для q₀
    ax2.plot(df["n"], df["CP_q0_Чебишов"], "D-", color="#9467bd", lw=2, ms=8,
label="Чебишов")
    ax2.plot(df["n"], df["CP_q0_Хеффдінг"], "^-", color="#8c564b", lw=2,
ms=8, label="Хеффдінг")
    ax2.plot(df["n"], df["CP_q0_Вальд"], "*-", color="#e377c2", lw=2, ms=10,
label="Вальд")
    ax2.axhline(config.EXP2_GAMMA, ls="--", color="black", alpha=0.6,
                label=f"Номінальне γ = {config.EXP2_GAMMA}")
    ax2.set_xscale("log")
    ax2.set_xlabel("n (log-шкала)")
    ax2.set_ylabel("Coverage Probability")
    ax2.set_title("Ймовірність покриття для $q_0$")
    ax2.set_ylim(-0.05, 1.10)
    ax2.legend(loc="lower right"); ax2.grid(True, which="both", alpha=0.3)

    plt.suptitle("Експериментальна перевірка теоретичних меж довірчого
інтервалу",
                 fontsize=14, fontweight="bold", y=1.02)
    save("11_coverage_probability.png")



def main():
    print("=" * 60)
    print(" 10_visualization.py — Побудова графіків")
    print("=" * 60)

    print("\n Графіки експерименту 2...")
    if os.path.exists(config.EXP2_RESULTS_CSV):
        df_exp2 = pd.read_csv(config.EXP2_RESULTS_CSV)
        plot_p1_q0(df_exp2)
        plot_recall(df_exp2)
        plot_precision(df_exp2)
        plot_three_bounds(df_exp2)
        plot_metrics_overview(df_exp2)
    else:
        print("     stochastic_analysis.csv не знайдено")

    print("\n Coverage Probability (нове)...")
    plot_coverage()

    print("\n ROC-крива...")
    plot_roc()

    print("\n Learning Curve...")
    plot_learning_curve()

    print("\n Порівняння pipeline'ів...")
    plot_pipeline_comparison()

    print("\n Динаміка навчання моделей...")
    plot_autoencoder_loss()
```

```
    plot_classifier_loss()

    print("\n" + "=" * 60)
    print(f"  Готово! Графіки у: {config.RESULTS_FIGS}")
    print("=" * 60)


if __name__ == "__main__":
    main()


import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family'] = 'DejaVu Sans'

params = [
    {
        "fig": "4.2", "code": "X1",
        "name": "Точність класифікації (F1-міра)",
        "unit": "",
        "worst": 0.68, "mid": 0.74, "best": 0.80,
        "direction": "more",
        "v1": 0.7566, "v2": 0.7661,
        "fname": "ris_4_2_X1.png",
    },
    {
        "fig": "4.3", "code": "X2",
        "name": "Розмірність простору ознак",
        "unit": "вимірів",
        "worst": 128, "mid": 64, "best": 16,
        "direction": "less",
        "v1": 32, "v2": 100,
        "fname": "ris_4_3_X2.png",
    },
    {
        "fig": "4.4", "code": "X3",
        "name": "Час навчання моделі",
        "unit": "хв",
        "worst": 60, "mid": 42, "best": 25,
        "direction": "less",
        "v1": 42, "v2": 38,
        "fname": "ris_4_4_X3.png",
    },
    {
        "fig": "4.5", "code": "X4",
        "name": "Час розробки",
        "unit": "людино-год",
        "worst": 900, "mid": 625, "best": 350,
        "direction": "less",
        "v1": 620, "v2": 486,
        "fname": "ris_4_5_X4.png",
    },
]


def score(val, worst, best):

    return (val - worst) / (best - worst) * 100


def make_graph(p):
```

```
    worst, best = p["worst"], p["best"]


    lo, hi = min(worst, best), max(worst, best)
    xs = [lo + (hi - lo) * i / 200 for i in range(201)]
    ys = [score(x, worst, best) for x in xs]

    fig, ax = plt.subplots(figsize=(7, 4.5))


    ax.plot(xs, ys, color="#1f4e79", linewidth=2, label="Шкала бальної
оцінки")


    for val, lab in [(p["worst"], "гірше"), (p["mid"], "середнє"),
(p["best"], "краще")]:
        b = score(val, worst, best)
        ax.scatter([val], [b], color="#1f4e79", s=40, zorder=5)
        ax.annotate(lab, (val, b), textcoords="offset points",
                    xytext=(0, -16), ha="center", fontsize=9,
color="#555555")


    b1 = score(p["v1"], worst, best)
    b2 = score(p["v2"], worst, best)
    ax.scatter([p["v1"]], [b1], color="#c00000", s=90, marker="o", zorder=6,
               label=f"Варіант 1 (Pipeline A): {p['v1']} -> {round(b1)}
балів")
    ax.scatter([p["v2"]], [b2], color="#e69138", s=90, marker="s", zorder=6,
               label=f"Варіант 2 (Pipeline B): {p['v2']} -> {round(b2)}
балів")

    for v, b, col in [(p["v1"], b1, "#c00000"), (p["v2"], b2, "#e69138")]:
        ax.plot([v, v], [0, b], color=col, linestyle="--", linewidth=0.8,
alpha=0.6)
        ax.plot([lo, v], [b, b], color=col, linestyle="--", linewidth=0.8,
alpha=0.6)

    xlabel = f"{p['code']}, {p['unit']}".rstrip(", ") if p["unit"] else
p["code"]
    ax.set_xlabel(xlabel, fontsize=11)
    ax.set_ylabel("Бальна оцінка B", fontsize=11)
    ax.set_title(f"Рисунок {p['fig']} — {p['name']}", fontsize=11)
    ax.set_ylim(-8, 108)
    ax.grid(True, linestyle=":", alpha=0.5)
    ax.legend(fontsize=8, loc="best")
    fig.tight_layout()
    fig.savefig(p["fname"], dpi=300, bbox_inches="tight")
    plt.close(fig)
    print(f"Збережено {p['fname']}  (Варіант1={round(b1)} балів,
Варіант2={round(b2)} балів)")


if __name__ == "__main__":
    for p in params:
        make_graph(p)
    print(" 4 графіки згенеровано.")
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ

Аналіз збіжності та надійності бінарної класифікації текстових даних на основі стохастичних методів та латентних представлень

Виконав:
студент IV курсу, групи КА-21
Христов Максим Богданович

Керівник:
Доцент кафедри ММСА, к.ф.-м.н., доц.
Спекторський Ігор Якович

Актуальність

- Соціальні мережі та цифрові сервіси генерують мільярди повідомлень щодня — автоматична класифікація текстів стає критичною задачею
- Поширена практика — повідомляти лише точкову оцінку (наприклад F1=0.85) без аналізу її стабільності
- На вибірці 500 об'єктів та сама модель може давати F1 від 0.71 до 0.81 — розкид майже 5 п.п. (за результатами експерименту 2)
- Без довірчого інтервалу неможливо коректно порівнювати моделі, відтворити результат чи оцінити ризик у промисловому застосуванні — потрібна методологія, що дає статистично обґрунтоване твердження замість одного числа

Мета та завдання

Мета:

Дослідити вплив обсягу вибірки на стабільність метрик
бінарної класифікації тональності тексту та визначити мінімальний
обсяг даних, достатній для отримання математично надійних результатів.

Завдання:

1. Побудувати стохастичну модель бінарного класифікатора
2. Вивести три довірчі інтервали (Чебишова, Вальда, Хеффдінга)
3. Реалізувати pipeline: W2V → DAE → MLP
4. Експериментально перевірити теоретичні межі через bootstrap
5. Визначити точки насичення n*_train і n*_test
6. Виконати функціонально-вартісний аналіз продукту

3

Об'єкт і предмет дослідження

Об'єкт дослідження	Предмет дослідження	Методологічний підхід
Процес бінарної класифікації текстових даних соціальних мереж	Метрики якості бінарного класифікатора та закономірності їх стохастичної стабільності залежно від обсягу вибірки	Поєднання теоретичного аналізу через апарат теорії ймовірностей з експериментальною верифікацією через bootstrap-метод.

4

Стохастична модель бінарного класифікатора

$p_1 = P\{\xi_k = 1 \mid b_k = 1\}$ — ймовірність правильного розпізнавання позитивного об'єкту.

$p_0 = P\{\xi_k = 0 \mid b_k = 0\}$ — ймовірність правильного розпізнавання негативного об'єкту.

$q_0 = 1 - p_0$ — ймовірність помилкового спрацьовування : негативний об'єкт названий позитивним.

$q_1 = 1 - p_1$ — ймовірність пропуску : позитивний об'єкт названий негативним.

- TP (True Positive) — правильно розпізнані позитивні,
- TN (True Negative) — правильно розпізнані негативні,
- FP (False Positive) — негативні названі позитивними,
- FN (False Negative) — позитивні названі негативними;
- n1 та n0 — кількість об'єктів кожного класу.

$$p_1^n = \frac{TP}{n_1}; \quad q_0^n = \frac{FP}{n_0}; \quad p_0^n = \frac{TN}{n_0}; \quad q_1^n = \frac{FN}{n_1}$$

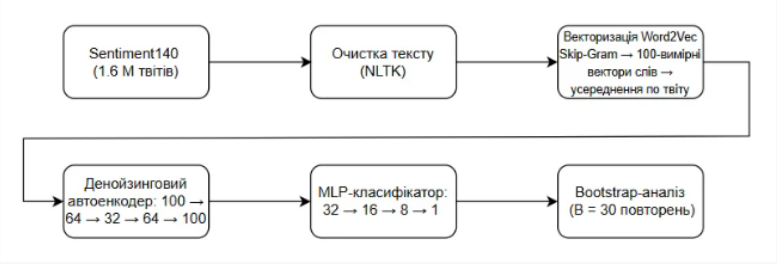
5

Довірчі межі

	Нерівність Чебишова	Нерівність Хеффдінга	Інтервал Вальда
Ширина інтервалу (при γ = 0.95, σ = 1/2 — максимум стандартного відхилення індикатора)	Δ = 2.236 / √n	Δ = 1.358 / √n	Δ = 0.980 / √n
Умови застосування	Без припущень про розподіл	Для обмежених в.в. (індикатори ∈ [0,1])	Через ЦГТ (асимптотично)
Характеристика	Універсальна, гарантована	Строга, неасимптотична	Найтісніша, потребує великого n

6

Архітектура: Word2Vec → DAE → MLP



Модульна архітектура дозволяє досліджувати стохастичну стабільність ознак незалежно від властивостей класифікатора.

7

Навчання автоенкодера

- Задача: стиснути 100-вимірний Word2Vec-вектор до 32, зберігши суть. Навчання — через відновлення чистого входу із зашумленого (σ = 0.3).
- Проблема першої версії: posterior collapse — латент вироджувався в константу. Роздільність класів впала з 2.13 до 0.08 (менше 4% сигналу).
- Рішення: StandardScaler замість MinMaxScaler, активація tanh замість сигмоїди, денойзингове формулювання.
- Результат: роздільність 1.33 (62% сигналу), усі 32 нейрони активні, реконструкція 0.42.

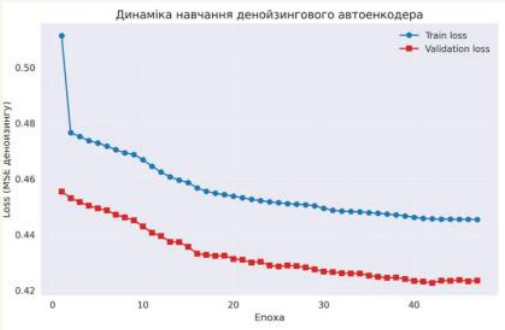


Рисунок 3.1 — Динаміка тренувальної та валідаційної функцій втрат денойзингового автоенкодера.

8

Навчання автоенкодера (візуалізація)

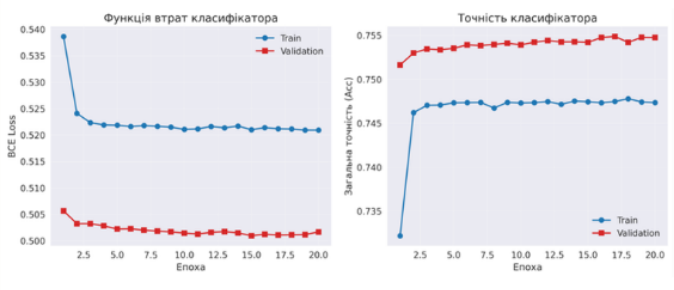


Рисунок 3.3 — Динаміка функції втрат BCE та загальної точності класифікатора.

Метрика	Значення
Загальна точність (Acc)	0.7545
Збалансована точність (BA)	0.7545
Точність ($Precision$)	0.7498
Повнота ($Recall$)	0.7635
$F1$ – міра ($F1$)	0.7566
Коефіцієнт кореляції Метьюса (MCC)	0.5090
Площа під ROC – кривою (AUC)	0.8360
Середня точність ($Average\ Precision, AP$)	0.8347
Каппа Коена	0.5089

9

Експеримент 1 — точка насичення тренувальної вибірки

n_train	*F1*	*Acc*	*AP*	*Delta F1*
1 000	0.5690	0.5304	0.5285	—
5 000	0.5981	0.6143	0.6634	+0.0291
10 000	0.6565	0.6654	0.7322	+0.0584
50 000	0.7299	0.7321	0.8095	+0.0734
100 000	0.7343	0.7383	0.8188	+0.0044
500 000	0.7505	0.7517	0.8318	+0.0161
1 100 000	0.7577	0.7528	0.8342	+0.0073

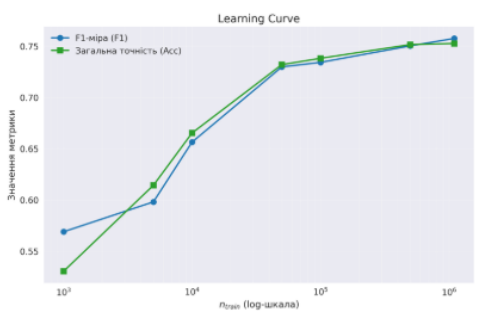


Рисунок 3.4 — Learning Curve. Залежність F1-міри та загальної точності від обсягу тренувальної вибірки в логарифмічній шкалі. Спостерігається класична S-подібна форма з виходом на плато.

n*_train ≈ 100 000

Збільшення вибірки в 11 разів дає лише 3.2 % приросту F1 — не виправдовує суттєвого зростання часу навчання.

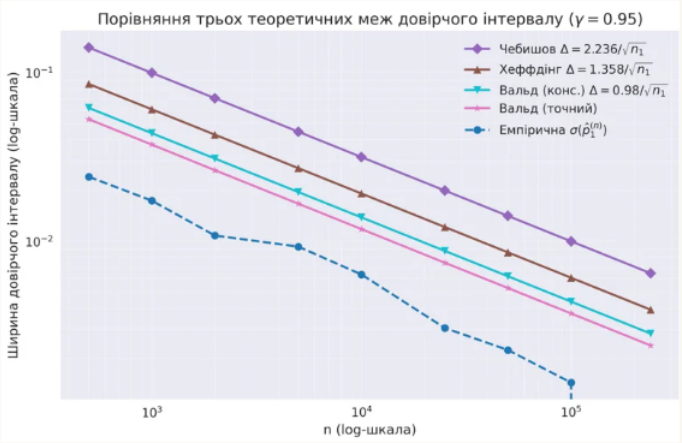
10

Експеримент 2 : стохастична збіжність

1) Точка надійної оцінки: n*_test ≈ 25 000
Усі три теоретичні підходи розділу 2 отримали повне експериментальне підтвердження.

2) При збільшенні n у 100 разів σ зменшилось у 10.52 рази
Фактичні значення (2.60; 4.04; 10.52) близькі до теоретично очікуваних (3.16; 3.16; 10.00)

3) Ймовірність покриття: Чебишов і Хеффдінг дають 1.000 (консервативні універсальні межі), Вальд — приблизно 0.96, що відповідає номінальному рівню довіри γ = 0.95.



Δ_Чеб : Δ_Хеф : Δ_Вальд = 2.28 : 1.39 : 1.00

11

Експеримент 2 : стохастична збіжність (Візуалізація)

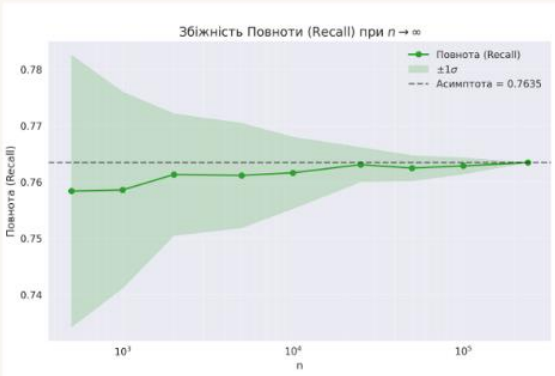


Рисунок 3.6 — Збіжність Повноти до асимптотичного значення 0.7635.

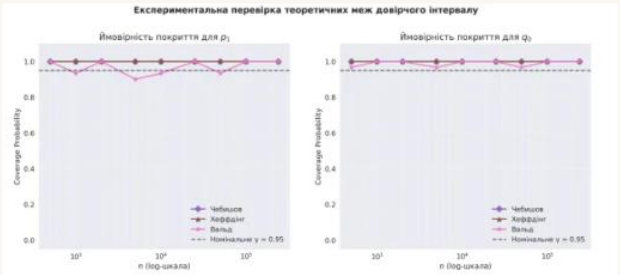


Рисунок 3.9 — Експериментальна перевірка теоретичних меж довірчого інтервалу.

12

Експеримент 3 : порівняння архітектур

- Pipeline B виграє 1–3 % за більшістю метрик
- Стиснення 100 → 32 зберігає 62 % дискримінативного сигналу — втрата 38 % є природною ціною компактнішого представлення
- Pipeline A обраний з трьох методологічних міркувань:

1. виокремлення етапу вилучення ознак як окремої стадії
2. обчислювальна ефективність (32-вимірні латенти)
3. pipeline-агностичність методології: стохастичні властивості метрик ідентичні у вхідному та латентному просторах

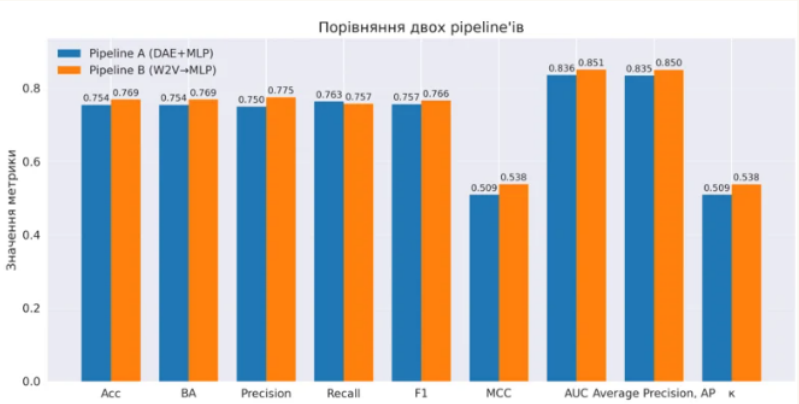


Рисунок 3.11 — Порівняння Pipeline A (DAE+MLP) та Pipeline B (W2V→MLP напряму) за всіма дев'ятьма метриками.

13

Функціонально-вартісний аналіз

Завдання	Новизна	Складн.	$T_р$, дні	$K_п$	$T_о$, людино-днів
Варіант 1 (Pipeline A)					
1. Модуль векторизації	Б	2	27	1.01	21.82
2. Автоенкодер + класифікатор	А	2	36	1.26	36.29
3. Bootstrap-аналіз + візуалізація	В	2	27	0.90	19.44
Разом за варіантом 1					77.55 (620.4 год)
Варіант 2 (Pipeline B)					
1. Модуль векторизації	Б	2	27	1.01	21.82
2. Класифікатор напряму	В	2	27	0.90	19.44
3. Аналіз надійності (межа Вальда) + CSV-виведення	В	1	14	0.90	10.08
Разом за варіантом 2					51.34 (410.7 год)

Стаття витрат	Варіант 1, грн	Варіант 2, грн
Заробітна плата розробника	155 100.00	102 700.00
Єдиний соціальний внесок (22 %)	34 122.00	22 594.00
Витрати на машинний час	60 094.21	39 779.00
Накладні витрати (67 %)	103 917.00	68 809.00
Повна вартість розробки	353 233.21	233 882.00

$$K_{тер}1 = 62.95 / 353\,233.21 = 1.782 \cdot 10^{-4}$$

$$K_{тер}2 = 62.44 / 233\,882.00 = 2.670 \cdot 10^{-4}$$

Конкордація експертів W = 0.771 перевищує нормативний поріг 0.67, тому оцінки достовірні; за коефіцієнтом техніко-економічного рівня оптимальним визнано Pipeline B, тоді як Pipeline A залишений у роботі з методологічних міркувань — для виокремлення етапу вилучення ознак, що є центральним для теми дослідження і не входить до економічного критерію ФВА.

14

Висновки та наукова новизна

Наукова новизна:

- Об'єднання трьох класичних меж (Чебишов, Вальд, Хеффдінг) в єдину методологію аналізу надійності класифікаційних метрик
- Експериментальна верифікація на масштабному датасеті — 1.6 млн об'єктів
- Доведено pipeline-агностичність: ієрархія меж і швидкість $O(1/\sqrt{n})$ зберігаються як у вхідному, так і в стиснутому латентному просторі

Практичне значення:

- $n^*_{train} \approx 100\,000$ — достатньо для досягнення $F1 \approx 0.75$
- $n^*_{test} \approx 25\,000$ — достатньо для оцінки з $\Delta \approx 0.01$ при $\gamma = 0.95$
- Для критичних застосувань — нерівність Чебишова; для типових — інтервал Вальда

15

Дякую за увагу