МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 31.00.00.000 ПЗ

Група ІПм-24-2

Капітан Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Капітан Владислав Петрович
(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми багатокласової класифікації об'єктів

(назва роботи)

Інженерія програмного забезпечення
(назва освітньої програми)

121 - Інженерія програмного забезпечення
(шифр і назва спеціальності)

Капітан В.П.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Зікратий Сергій Вікторович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль
доц. Вовк Р.Б.
(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікований рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“04” вересня 2025р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Капітану Владиславу Петровичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та алгоритми багатокласової класифікації об’єктів”
керівник проекту (роботи) Зікратий Сергій Вікторович, к.т.н., доцент
затверджені наказом закладу вищої освіти від “ 05 ” листопада 2025 р. № 695/7
2. Строк подання студентом проекту (роботи) 15 грудня 2025р.
3. Вихідні дані до проекту (роботи) Моделі, методи та алгоритми багато класової класифікації об’єктів відгуків за датасетом Amazon Reviews.
4. Зміст розрахунково – пояснювальної записки (перелік питань які потрібно розробити)
1. Теоретичні відомості моделей, методів та алгоритмів предметної області
2. Проектування засобу багатокласової класифікації.
3. Програмна реалізація та тестування програми багатокласової класифікації на основі моделей, методів та алгоритмів.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
1. Графічне представлення датасету відгуків на бази Amazon (рис. 1.1., ст. 16)
2. Приклад прямого мапування (рис. 1.2., ст. 18)
3. Приклад карти сентименту (рис. 1.3., ст. 19)
4. Приклад обчислення картилів (рис. 1.4., ст. 20)
5. Зведена схема: потік для генерації всіх міток (рис. 1.5., ст. 21)
6. Діаграма класів (рис. 2.1., ст. 30)
7. Сценарій використання (рис. 2.2., ст. 32)
8. Діаграма тренування активності (рис. 2.3., ст. 41)
9. Діаграма швидкого старту (рис. 2.4., ст. 44)
10. Скріншоти веб-інтерфейсу, графіки, діаграми (рис. 3.3 – 3.33., ст. 71 – 88)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	20.09.2025	виконано
2	Аналіз сучасних технологій багатокласової класифікації об’єктів	01.10.2025	виконано
3	Дослідження моделей, методів та алгоритмів предметної галузі	12.10.2025	виконано
4	Дослідження алгоритму багатокласової класифікації об’єктів	25.10.2025	виконано
5	Формулювання вимог та алгоритмів функціонування системи	05.11.2025	виконано
6	Програмна реалізація рішення	22.11.2025	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2025	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 97 с., 42 рис., 4 табл., 45 джерел.

Тема: Моделі, методи та алгоритми багатокласової класифікації об'єктів

Об'єкт дослідження: процес класифікації текстових рецензій користувачів платформи Amazon.

Мета роботи: розробка програмної системи багатокласової класифікації текстових рецензій на основі алгоритмів машинного навчання.

Предмет дослідження: моделі, методи та алгоритми машинного навчання для розв'язання задач багатокласової класифікації текстових документів на основі статистичних та лінгвістичних ознак.

Результати дослідження:

Сформовано три незалежні задачі класифікації за рейтингом, сентиментом та корисністю. Розроблено модульну архітектуру системи з компонентами передобробки, витягу ознак та навчання моделей. Реалізовано три алгоритми через уніфіковані інтерфейси. Створено веб-інтерфейс на Streamlit. Експериментальні дослідження виявили оптимальні підходи для різних типів задач.

Висновок:

Розроблено функціонуючу модульну систему багатокласової класифікації текстових рецензій з використанням класичних алгоритмів машинного навчання. Експерименти підтвердили конкурентоспроможність обраних методів для практичного застосування в електронній комерції.

БАГАТОКЛАСОВА КЛАСИФІКАЦІЯ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, ТЕКСТОВИЙ АНАЛІЗ, РЕЦЕНЗІЇ, TF-IDF, НАЇВНИЙ БАЙЄС, ВИПАДКОВИЙ ЛІС, МЕТОД ОПОРНИХ ВЕКТОРІВ, АНАЛІЗ СЕНТИМЕНТУ, STREAMLIT, PYTHON.

ANNOTATION

Master's Thesis: 97 pages, 42 figures, 4 tables, 45 references.

Title: Models, Methods, and Algorithms for Multiclass Classification of Objects

Object of the study: the process of classifying user text reviews on the Amazon platform.

Purpose of the thesis: to develop a software system for multiclass classification of text reviews with a comparative analysis of different algorithms.

Subject of the study: machine learning models, methods, and algorithms for solving problems of multi-class classification of text documents based on statistical and linguistic features.

Research results:

Three independent classification tasks were formulated based on rating, sentiment, and helpfulness. A modular system architecture was developed, including components for data preprocessing, feature extraction, and model training. Three algorithms were implemented using unified interfaces. A web interface was created using Streamlit. Experimental studies identified optimal approaches for different types of classification tasks.

Conclusion:

A functional modular system for multiclass classification of text reviews based on classical machine learning algorithms was developed. Experimental results confirmed the competitiveness of the selected methods for practical application in electronic commerce.

MULTICLASS CLASSIFICATION, MACHINE LEARNING, NATURAL LANGUAGE PROCESSING, TEXT ANALYSIS, REVIEWS, TF-IDF, NAIVE BAYES, RANDOM FOREST, SUPPORT VECTOR MACHINES, SENTIMENT ANALYSIS, STREAMLIT, PYTHON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (програмний інтерфейс застосування)

CLI – Command Line Interface (інтерфейс командного рядка)

CSV – Comma-Separated Values (значення, розділені комами)

CSR – Compressed Sparse Row (стиснутий розріджений рядок)

DF – Document Frequency (частота документа)

F1 – F1-Score (гармонічне середнє precision та recall)

FN – False Negative (хибнонегативний результат)

FP – False Positive (хибнопозитивний результат)

GUI – Graphical User Interface (графічний інтерфейс користувача)

IDF – Inverse Document Frequency (обернена частота документа)

JSON – JavaScript Object Notation (нотація об'єктів JavaScript)

NB – Naive Bayes (наївний байєсівський класифікатор)

NLP – Natural Language Processing (обробка природної мови)

NLTK – Natural Language Toolkit (інструментарій обробки природної мови)

OvR – One-vs-Rest (схема багатокласової класифікації)

RF – Random Forest (випадковий ліс)

SVM – Support Vector Machine (метод опорних векторів)

TF – Term Frequency (частота терміну)

TF-IDF – Term Frequency-Inverse Document Frequency (частота терміну з ваговим коефіцієнтом оберненої частоти документа)

TN – True Negative (істинно негативний результат)

TP – True Positive (істинно позитивний результат)

ВСТУП

Актуальність роботи

Цифрова трансформація економіки призвела до експоненційного зростання обсягів текстових даних, які генеруються користувачами онлайн-платформ. Платформи електронної комерції щодня накопичують мільйони відгуків, які містять цінну інформацію про якість продуктів, настрої споживачів та фактори, що впливають на рішення про покупку. Ручний аналіз таких обсягів інформації стає практично неможливим, тому автоматизовані системи класифікації текстових документів набувають критичної важливості для бізнесу.

Багатокласова класифікація текстів залишається актуальною проблемою в галузі машинного навчання та обробки природної мови. На відміну від простіших бінарних задач, багатокласові сценарії вимагають складніших методів моделювання, особливо коли одні дані можуть використовуватися для розв'язання декількох незалежних задач одночасно. Такий багатоаспектний підхід відображає реальну природу текстового контенту та бізнес-потреби компаній.

Класичні алгоритми машинного навчання – наївний Байєс, випадковий ліс та метод опорних векторів – продовжують демонструвати конкурентоспроможні результати попри появу складних нейронних архітектур [1, 3, 7, 8]. Ці методи характеризуються оптимальним балансом між точністю передбачень і обчислювальною ефективністю, що робить їх придатними для практичного застосування за умов обмежених ресурсів. Дослідження порівняльної ефективності різних підходів на конкретних прикладних задачах зберігає актуальність для обґрунтованого вибору оптимальних рішень.

Для України розвиток технологій автоматичного аналізу текстів має стратегічне значення в контексті цифрової трансформації економіки та підвищення конкурентоспроможності вітчизняних IT-компаній. Створення систем обробки природної мови сприяє розвитку електронної комерції, покращенню якості онлайн-сервісів та формуванню нових компетенцій у галузі штучного інтелекту.

Порівняння роботи з відомими розв'язаннями проблеми

Текстова класифікація має тривалу історію розвитку від ранніх правилових систем до сучасних нейронних архітектур. Класичні статистичні методи, такі як наївний Байєс, були адаптовані для роботи з текстами ще в 1960-х роках і продовжують використовуватись завдяки простоті та ефективності [1]. Ансамблеві методи, зокрема випадковий ліс, з'явились у 2001 році після публікації Лео Бреймана [7] та продемонстрували здатність моделювати складні нелінійні залежності. Метод опорних векторів, запропонований Кортесом та Вапником у 1995 році [8], швидко став популярним для текстової категоризації завдяки здатності ефективно працювати в просторах високої розмірності.

Сучасні підходи спираються на глибоке навчання та трансформерні архітектури. Модель BERT [17] та великі мовні моделі GPT [18] демонструють вражаючі результати через механізми self-attention та попереднє навчання на величезних корпусах. Проте складні нейронні архітектури вимагають значних обчислювальних ресурсів та спеціалізованого апаратного забезпечення, що робить їх недоступними для організацій з обмеженими бюджетами [28]. Інтерпретованість передбачень нейронних мереж залишається проблемою, на відміну від статистичних підходів [33].

Для задач класифікації рецензій існують спеціалізовані рішення. Робота Себастіані [44] запропонувала систематичну методологію машинного навчання для текстової категоризації. Дослідження Янга та Педерсена [38] показало важливість правильного відбору ознак для точності моделей. Аналіз сентименту рецензій став окремим напрямком з власними методологіями, де статистичні методи виявились ефективнішими за лексиконні підходи завдяки здатності автоматично виявляти контекстуальні залежності [5].

Порівняно з існуючими рішеннями, запропонована система відрізняється комплексним підходом до одночасного розв'язання трьох незалежних задач класифікації на одному наборі даних. Використання квартільного методу для категоризації корисності забезпечує адаптивність до природного розподілу даних без

ручного налаштування порогів. Модульна архітектура з уніфікованими інтерфейсами дозволяє легко додавати нові алгоритми без модифікації існуючого коду, що відрізняє це рішення від монолітних систем.

Мета і задачі дослідження

Метою магістерської роботи є розробка програмної системи багатокласової класифікації текстових рецензій на основі алгоритмів машинного навчання.

Досягнення мети передбачає розв'язання таких задач:

1) Проаналізувати предметну область багатокласової класифікації текстових документів та визначити специфіку роботи з користувацькими рецензіями.
2) Дослідити структуру наявних даних та обґрунтувати вибір цільових змінних для формування трьох незалежних задач класифікації.
3) Вивчити математичні основи та особливості застосування класичних алгоритмів машинного навчання для текстової класифікації.
4) Спроектувати модульну архітектуру програмної системи з розділенням відповідальності між компонентами та забезпеченням можливості розширення функціональності.
5) Розробити модулі передобробки текстових та числових даних з урахуванням специфіки користувацького контенту.
6) Спроектувати механізми витягу та комбінування різнорідних типів ознак у єдиний простір представлень.
7) Визначити структуру конвеєра тренування та оцінювання моделей класифікації.
8) Реалізувати програмні модулі системи з використанням сучасних бібліотек машинного навчання та обробки даних.
9) Імплементувати три алгоритми класифікації через уніфіковані інтерфейси для забезпечення взаємозамінності моделей.
10) Провести систематичні експерименти з порівняння ефективності алгоритмів на різних задачах класифікації.

11) Виконати тестування коректності роботи компонентів системи та аналіз результатів класифікації.
12) Розробити веб-інтерфейс для зручної взаємодії користувачів з функціональністю системи.

Об'єктом дослідження є процес багатокласової класифікації текстових рецензій користувачів платформи електронної комерції Amazon.

Предметом дослідження є моделі, методи та алгоритми машинного навчання для розв'язання задач багатокласової класифікації текстових документів на основі статистичних та лінгвістичних ознак.

Методи дослідження

Методи дослідження базуються на фундаментальних принципах машинного навчання, теорії ймовірностей та математичної статистики. Для передобробки текстових даних використовуються методи обробки природної мови, включаючи токенізацію, нормалізацію та фільтрацію стоп-слів [29]. Векторизація текстів виконується методом TF-IDF на основі теорії інформаційного пошуку [4].

Для класифікації застосовуються три різні парадигми машинного навчання: імовірнісний підхід через мультиноміальний наївний байєсівський класифікатор на основі теореми Байєса [1], ансамблеві методи через алгоритм випадкового лісу з використанням дерев рішень [7], геометричний підхід через метод опорних векторів з розв'язанням оптимізаційних задач квадратичного програмування [8].

Для оцінювання ефективності моделей використовуються метрики класифікації з теорії машинного навчання, включаючи accuracy, precision, recall та F1-міру з macro-averaging для багатокласових задач [37]. Аналіз результатів виконується через побудову матриць плутанини та статистичне порівняння показників якості. Проектування системи базується на принципах об'єктно-орієнтованого програмування та патернах проектування програмного забезпечення.

Наукова новизна одержаних результатів

Запропоновано комплексний підхід до формування трьох незалежних задач багатокласової класифікації на єдиному наборі текстових рецензій, що дозволяє багатоаспектний аналіз користувацького контенту через одночасне визначення рейтингу продукту, сентименту відгуку та корисності рецензії.

Удосконалено метод категоризації корисності рецензій через застосування квартильного розподілу до обчисленого відношення позитивних оцінок, що забезпечує збалансованість класів та адаптивність до природного розподілу даних без жорстких порогових значень.

Дістала подальшого розвитку методика комбінування гетерогенних типів ознак для текстової класифікації в частині ефективного об'єднання високорозмірних розріджених TF-IDF векторів зі щільними статистичними характеристиками документів у єдиний простір представлень.

Проведено систематичне порівняльне дослідження ефективності трьох класичних алгоритмів машинного навчання на множині незалежних задач багатокласової класифікації, що дозволило виявити оптимальні підходи для різних типів цільових змінних.

Практичне значення одержаних результатів

Практична цінність роботи полягає у створенні функціонуючої програмної системи для автоматичної обробки великих обсягів текстових рецензій у системах електронної комерції. Розроблена система дозволяє одночасно оцінювати рейтинг продукту, визначати емоційне забарвлення відгуку та передбачати корисність рецензії, що може використовуватися для інтелектуального ранжування та фільтрування контенту на торгових платформах.

Модульна архітектура системи забезпечує можливість інтеграції з існуючими інформаційними системами підприємств електронної комерції. Веб-інтерфейс на базі Streamlit надає зручний інструмент для аналітиків та менеджерів продуктів без

необхідності програмування [23]. Функціонал пакетного передбачення дозволяє ефективно обробляти великі обсяги нових рецензій у автоматичному режимі.

Результати порівняльного дослідження алгоритмів мають практичну цінність для обґрунтованого вибору оптимальних методів класифікації залежно від специфіки задачі та доступних обчислювальних ресурсів. Документація проекту та реалізовані компоненти можуть використовуватися в навчальному процесі для підготовки фахівців у галузі машинного навчання та обробки природної мови.

Особистий внесок студента

1. Проведено комплексний аналіз предметної області багатокласової класифікації текстових рецензій та визначено три незалежні задачі категоризації.
2. Розроблено методику формування цільових змінних на основі квартильного розподілу для задачі оцінки корисності рецензій.
3. Спроектовано та реалізовано модульну архітектуру системи класифікації з механізмами передобробки, витягу ознак та комбінування гетерогенних типів даних.
4. Імплементовано три алгоритми класифікації через уніфіковані інтерфейси та проведено систематичні експерименти з порівняння їх ефективності.
5. Створено веб-інтерфейс на базі Streamlit для інтерактивної роботи з усією функціональністю системи.

Апробація результатів магістерської роботи

Основні положення та результати магістерської роботи обговорювалися на засіданнях кафедри інженерії програмного забезпечення Івано-Франківського національного технічного університету нафти і газу протягом 2025 року.

Структура та обсяг магістерської роботи

Магістерська робота викладена на 97 сторінках друкованого тексту, який складається зі вступу, трьох розділів, висновків, списку використаних джерел (45 найменувань). Робота містить 42 рисунка та 4 таблиці.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ БАГАТОКЛАСОВОЇ КЛАСИФІКАЦІЇ РЕЦЕНЗІЙ

1.1 Постановка задачі багатокласової класифікації користувацьких відгуків

1.1.1 Специфікація багатокласової класифікації

Багатокласова класифікація текстових документів залишається однією з фундаментальних проблем у галузі обробки природної мови та машинного навчання. На відміну від бінарної класифікації, де модель розрізняє лише два класи, багатокласова задача передбачає віднесення об'єкта до однієї з трьох або більше категорій. Ця особливість створює додаткові виклики як на етапі проектування системи, так і під час оцінювання її ефективності [10].

1.1.2 Багатоаспектний підхід до аналізу

У контексті аналізу текстових рецензій користувачів на платформі Amazon виникає потреба одночасного розв'язання декількох незалежних задач класифікації. Кожна рецензія містить не лише текстову інформацію, а й супутні метадані, які дозволяють сформувати різні цільові змінні для навчання моделей. Така постановка завдання відображає реальні бізнес-потреби електронних торгових майданчиків, де важливо не тільки визначити загальний тон відгуку, але й передбачити його корисність для інших користувачів та оцінити якість продукту на основі текстового опису досвіду споживання.

1.1.3 Високорозмірність текстових даних

Специфіка роботи з текстовими даними полягає у високій розмірності простору ознак та розрідженості векторних представлень. Кожне слово у словнику потенційно може стати окремою ознакою, що призводить до формування матриць з тисячами стовпців при відносно невеликій кількості ненульових елементів у кожному

рядку. Ця властивість вимагає ретельного підбору алгоритмів та методів передобробки даних [4].

1.1.4 Дисбаланс класів

Окремої уваги заслуговує проблема дисбалансу класів. У реальних даних розподіл рецензій за оцінками рідко буває рівномірним. Користувачі схильні частіше залишати або дуже позитивні, або дуже негативні відгуки, тоді як помірковані оцінки трапляються значно рідше. Така асиметрія впливає на процес навчання моделей та вимагає використання спеціальних метрик оцінювання, які враховують представленість кожного класу в даних [37].

Важливим аспектом є взаємозв'язок між різними цільовими змінними. Наприклад, рецензії з високими оцінками часто мають позитивний сентимент, проте цей зв'язок не є абсолютним. Детальний негативний відгук може виявитися більш корисним для інших покупців, ніж коротка позитивна ремарка. Розуміння таких залежностей допомагає краще інтерпретувати результати класифікації та приймати обґрунтовані рішення щодо архітектури системи.

1.2 Структурний аналіз набору даних Amazon Reviews

1.2.1 Характеристика джерела даних

Базу даних для дослідження складають рецензії користувачів на харчові продукти з платформи Amazon, зібрані протягом тривалого періоду спостережень [20]. Кожна рецензія представлена як окремий запис у реляційній базі даних SQLite, що забезпечує зручність доступу та можливість ефективного фільтрування за різними критеріями [21].

1.2.2 Текстові компоненти рецензій

Структура окремого запису включає кілька ключових полів. Текстова складова рецензії розділена на два компоненти: короткий заголовок та розгорнутий текст відгуку. Заголовок зазвичай містить стислу думку користувача про продукт,

виражену в одному реченні або кількох словах. Основний текст надає детальнішу інформацію про досвід використання товару, його переваги та недоліки, контекст застосування.

1.2.3 Числові характеристики

Числові характеристики рецензії представлені рейтинговою оцінкою від одного до п'яти балів. Ця оцінка відображає загальне враження користувача та є найбільш очевидним індикатором якості продукту.

Проте інтерпретація цієї шкали може відрізнятися між різними користувачами: для когось три бали означають нейтральний досвід, тоді як інші сприймають це як негативну оцінку.

Метадані корисності рецензії заслуговують окремого розгляду. Платформа Amazon дозволяє користувачам оцінювати відгуки інших покупців за критерієм корисності. Ця інформація зберігається у вигляді двох чисел: кількості користувачів, які визнали рецензію корисною, та загальної кількості оцінок корисності. Відношення цих величин формує показник відносної корисності, який може варіюватися від нуля до одиниці [44]. На рисунку 1.1 наочно продемонстровано приклад структури запису рецензії.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Таблиця: Reviews — Filter in any column

	Id	ProductId	UserId	ProfileName	HelpfulNumerator	HelpfulDenominator	Score	Time	Summary	Text
	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр
25	25	B001GVISJM	A22P2J09NJ9HKE	S. Cabanaugh "jilly pepper"	0	0	5	1295481600	Please sell these in Mexico!!	I have lived out of the US for over 7 yrs now, and I so mis…
26	26	B001GVISJM	A3FONPR03H3PJS	Deborah S. Linzer "Cat Lady"	0	0	5	1288310400	Twizzlers - Strawberry	Product received is as advertised. <a …
27	27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	1	1	1332633600	Nasty No flavor	The candy is just red , No flavor . Just plan and chewy . …
28	28	B001GVISJM	AAAS38B98HMIK	Heather Dube	0	1	4	1331856000	Great Bargain for the Price	I was so glad Amazon carried these batteries. I have a har…
29	29	B00144C10S	A2F4LZVGFLD1OB	DaisyH	0	0	5	1338854400	YUMMY!	I got this for my Mum who is not diabetic but needs to watc…
30	30	B0001PB9FY	A3HDKO7OW0QNK4	Canadian Fan	1	1	5	1107820800	The Best Hot Sauce in the World	I don't know if it's the cactus or the tequila or just the …
31	31	B003F6UO7K	AFM0O9480F04W	Sherril	0	0	5	1297641600	Great machine!	I have never been a huge coffee fan. However, my mother …
32	32	B003F6UO7K	A31OQO709M20Y7	Molly V. Smith "staral"	0	1	5	1288310400	THIS IS MY TASTE...	This offer is a great price and a great taste, thanks Amazo…
33	33	B001EO5QW8	AOVROBZ8BNTP7	S. Potter	19	19	4	1163376000	Best of the Instant Oatmeals	McCann's Instant Oatmeal is great if you must have your …
34	34	B001EO5QW8	A3PMM0NFVEJGK9	Megan "Bad at Nicknames"	13	13	4	1166313600	Good Instant	This is a good instant oatmeal from the best oatmeal brand.…
35	35	B001EO5QW8	A2EB6OGOWCRUSH	CorbyJames	9	9	5	1175212800	Great Irish oatmeal for those in a …	Instant oatmeal can become soggy the minute the water hits …
36	36	B001EO5QW8	A2CI0RLADCRKFF	T. J. Ryan	3	3	4	1210464000	satisfying	McCann's Instant Irish Oatmeal, Variety Pack of Regular, …
37	37	B001EO5QW8	A1MYS9LFFBIYKM	Abby Chase "gluten free"	2	2	5	1190851200	Love Gluten Free Oatmeal!!!	For those of us with celiac disease this product is a …
38	38	B001EO5QW8	A3MGP2E1ZZ6GRB	Zardoz "focuspuller"	1	1	5	1238457600	it's oatmeal	What else do you need to know? Oatmeal, instant (make it …
39	39	B001EO5QW8	A2GHZ2UTV2B0CD	JERRY REITH	0	0	4	1350777600	GOOD WAY TO START THE DAY....	I WAS VISITING MY FRIEND NATE THE OTHER MORNING FOR COFFEE …
40	40	B001EO5QW8	AO80AC8313NIZ	kYpondman	0	0	5	1317168000	Wife's favorite Breakfast	I ordered this for my wife as it was reccomended by our …
41	41	B001EO5QW8	AQCY5KRO7489S	Garrett	0	0	5	1303430400	Why wouldn't you buy oatmeal from …	The variety packs taste great! I have them every…
42	42	B001EO5QW8	A1WK4ALVZDYPUE	Dick Baldwin "christobe"	0	0	5	1302134400	Oatmeal For Oatmeal Lovers	McCann's makes oatmeal for every oatmeal connoisseur, …
43	43	B001EO5QW8	A16XFOYQSJREL7	Roger Pugliese	0	0	5	1287014400	Food-Great	I have McCann's Oatmeal every morning and by ordering it …
44	44	B001EO5QW8	A17DW6SUOC70DJ	Mother of 9	0	0	5	1255392000	Good Hot Breakfast	McCann's Oatmeal is a good quality choice. Our favorite is…
45	45	B001EO5QW8	A2G7B7FKP2O2PU	D. Leschke	0	0	5	1209686400	Great taste and convenience	We really like the McCann's steel cut oats but find we don'…
46	46	B001EO5QW8	A39Z97950MCTQE	K. A. Freel	0	0	3	1205193600	Hearty Oatmeal	This seems a little more wholesome than some of the …
47	47	B001EO5QW8	AQLL2R1PPR46X	grumpyrainbow	0	0	5	1192752000	good	Good oatmeal. I like the apple cinnamon the best. Though …

Рис. 1.1. Приклад запису рецензії в базі Amazon

1.2.3 Аналіз розподілів

Аналіз розподілу довжин текстових полів показує значну варіативність. Короткі рецензії можуть складатися з декількох слів, тоді як детальні відгуки досягають кількох абзаців тексту. Ця неоднорідність створює додаткові труднощі для моделей класифікації, оскільки інформативність коротких та довгих текстів може суттєво відрізнятися.

Розподіл рейтингових оцінок у наборі даних демонструє типову для комерційних платформ асиметрію. Переважна більшість користувачів схильна ставити або максимальні оцінки за задоволення продуктом, або мінімальні за розчарування. Помірковані оцінки становлять меншу частку, що відображає психологічну тенденцію людей виражати крайні емоційні стани частіше, ніж нейтральні.

Показники корисності розподілені нерівномірно через особливості механізму оцінювання на платформі. Нові рецензії мають малу кількість оцінок або взагалі їх не мають, тоді як популярні відгуки накопичують десятки та сотні голосів. Крім того, існує проблема нульових знаменників: деякі рецензії ніколи не оцінювалися користувачами, що унеможливлює обчислення відношення без спеціальної обробки таких випадків.

Мовні особливості текстів рецензій відображають природність користувацького контенту. Тексти містять граматичні помилки, розмовні вирази, жаргонізми та емоційно забарвлену лексику. Користувачі часто використовують капіталізацію для емфази, знаки оклику для вираження емоцій та неформальні скорочення. Ці характеристики відрізняють дані від відредагованих публікацій чи офіційних документів.

Технічні аспекти зберігання даних у SQLite базі передбачають використання індексації для прискорення запитів та можливість роботи з великими обсягами інформації без необхідності завантаження всього набору даних в оперативну пам'ять одночасно. Така архітектура дозволяє масштабувати експерименти та працювати з різними підмножинами даних для валідації моделей.

1.3 Стратегії формування цільових змінних для багатоаспектної класифікації

1.3.1 Пряме відображення рейтингів

Процес створення цільових міток для навчання моделей класифікації вимагає чіткого визначення критеріїв та правил перетворення вихідних даних у категоріальні змінні. У рамках цього дослідження реалізовано три незалежні стратегії формування міток, кожна з яких відповідає конкретній аналітичній задачі.

Перша модель класифікації працює з прямим відображенням числової рейтингової оцінки у категорії класів. Оскільки користувачі виставляють оцінки від одного до п'яти балів, природним рішенням стає використання цих значень як окремих класів без додаткових перетворень. Схематичний приклад мапування наведено на рисунку 1.2.

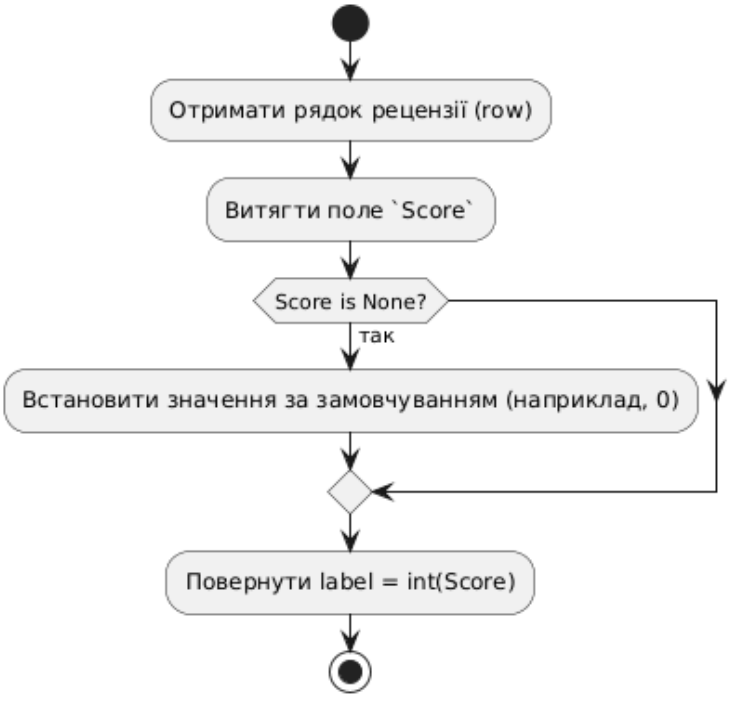


Рис. 1.2. Приклад прямого мапування

Така постановка задачі дозволяє моделі навчитися розрізняти тонкі відмінності між різними рівнями задоволеності продуктом на основі текстового

змісту рецензії. Проте варто відзначити, що така детальна градація підвищує складність задачі порівняно з більш грубими схемами категоризації [44].

1.3.2 Триступенева категоризація сентименту

Друга стратегія формування міток фокусується на визначенні емоційного забарвлення тексту через концепцію сентименту. Замість п'яти категорій тут використовується триступенева класифікація: негативний, нейтральний та позитивний настрій. Перетворення здійснюється шляхом агрегації рейтингових оцінок у більш широкі групи. Оцінки один та два бали трактуються як негативний сентимент, три бали позначають нейтральну позицію, а чотири та п'ять балів відповідають позитивному ставленню. Відповідна карта відповідності рейтингів до класів сентименту подана на рисунку 1.3 [5].

Рис. 1.3. Приклад карти сентименту

Таке групування базується на загальноприйнятій інтерпретації п'ятибальної шкали оцінювання, де середнє значення розглядається як точка балансу між задоволенням та незадоволенням. Зменшення кількості класів спрощує задачу класифікації та потенційно підвищує точність передбачень, оскільки межі між класами стають чіткішими. Водночас деяка інформація губиться через об'єднання різних рівнів інтенсивності емоцій в одну категорію.

1.3.3 Квартильний метод корисності

Третій підхід до формування міток концептуально відрізняється від попередніх, оскільки базується не на рейтингу продукту, а на реакції інших користувачів на рецензію. Метрика корисності відгуку обчислюється як відношення кількості позитивних оцінок корисності до загальної кількості оцінок. Це безрозмірна величина в діапазоні від нуля до одиниці, яка характеризує відсоток користувачів, що визнали відгук корисним для себе [20].

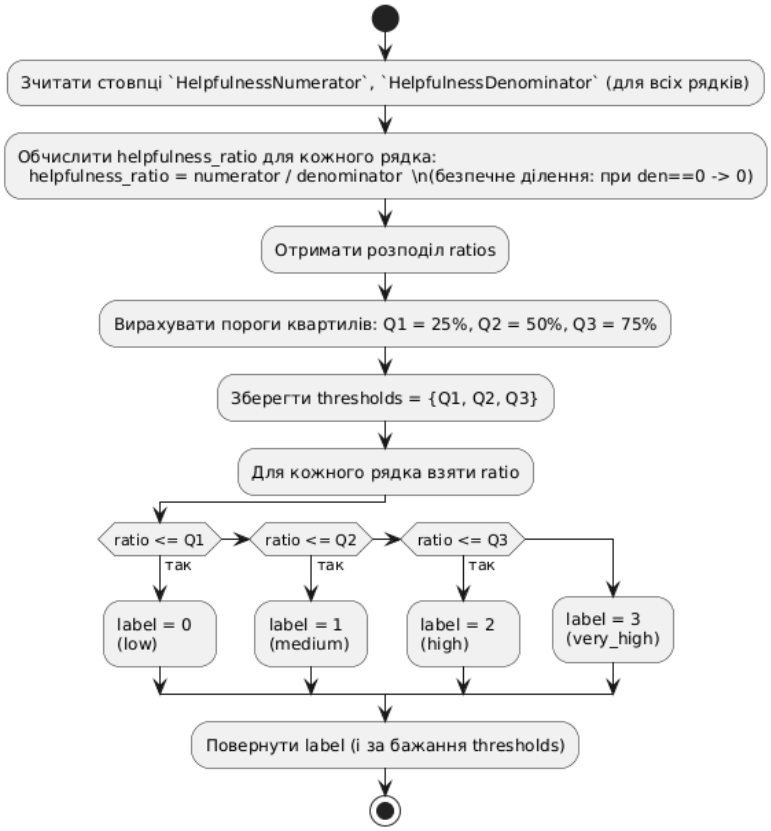


Рис. 1.4. Приклад обчислення квартилів

Перетворення неперервного показника корисності у категоріальну змінну здійснюється через статистичний метод квартильного розподілу. Принцип обчислення квартилів та поділу вибірки ілюструється на рисунку 1.4. Весь масив значень корисності ділиться на чотири рівні частини за кількістю спостережень. Перший квартиль охоплює найменш корисні рецензії, другий та третій квартилі відповідають середнім рівням корисності, а четвертий квартиль включає найбільш цінні для користувачів відгуки. Така стратегія забезпечує збалансованість класів та враховує природний розподіл даних без жорстких порогових значень [2].

Вибір саме чотирьох категорій корисності обумовлений прагненням знайти компроміс між детальністю градації та складністю задачі класифікації. Менша кількість класів могла б призвести до втрати важливих відмінностей між рецензіями різного рівня корисності, тоді як більша кількість ускладнила б навчання моделей через зменшення кількості прикладів у кожному класі.

1.3.4 Обробка граничних випадків

Важливою технічною деталлю є обробка граничних випадків при обчисленні показника корисності. Рецензії без жодної оцінки створюють проблему ділення на нуль. Для таких випадків застосовується спеціальна логіка: якщо знаменник дорівнює нулю, відношення також приймається рівним нулю.

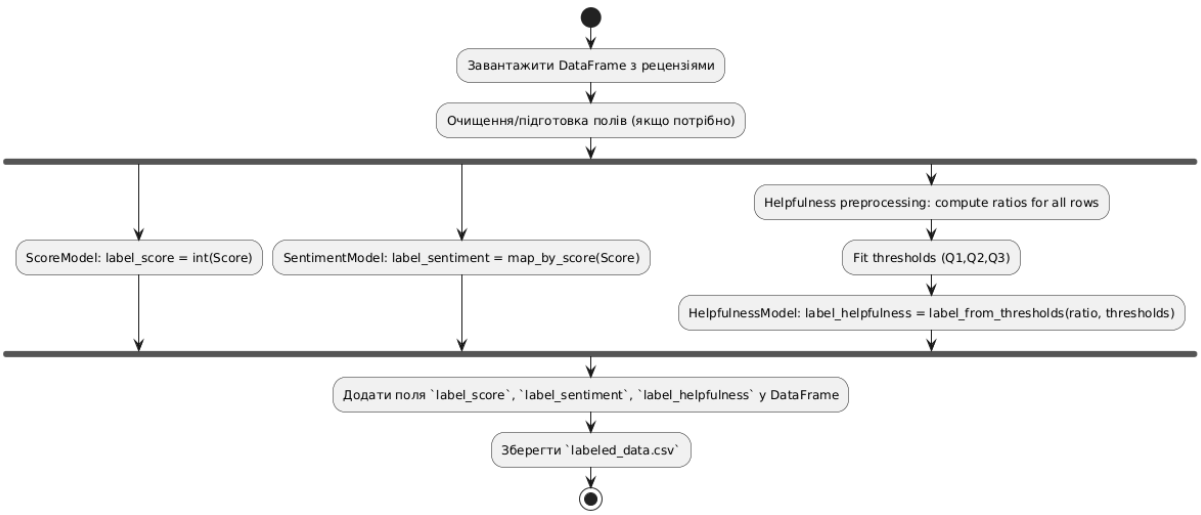


Рис. 1.5 Зведена схема: потік для генерації всіх міток

Це консервативне рішення відображає відсутність інформації про корисність та дозволяє уникнути технічних помилок при обчисленнях. Узагальнену схему формування всіх типів міток у межах запропонованого підходу наведено на рисунку 1.5.

Обчислення квартилів виконується на всьому наборі даних одночасно, що гарантує статистичну коректність розподілу. Значення порогів між квартилями зберігаються для можливості застосування тієї ж схеми категоризації до нових даних під час інференсу моделей. Без збереження цих порогів неможливо коректно класифікувати рецензії, які надходять після завершення тренування.

Незалежність трьох описаних стратегій формування міток дозволяє тренувати окремі спеціалізовані моделі для кожної задачі. Така архітектура забезпечує гнучкість системи та можливість оптимізації моделей під конкретні бізнес-цілі. Наприклад, для рекомендаційної системи може бути важливіша точність визначення сентименту, тоді як для модерації контенту більше значення має передбачення корисності відгуків.

1.4 Порівняльний огляд алгоритмів машинного навчання для текстової класифікації

1.4.1 Вступ до порівняльних алгоритмів

Історично розвиток методів автоматичної класифікації текстів пройшов декілька етапів, від простих правилових систем до складних нейронних архітектур. Для розв'язання задач у цьому дослідженні обрано три класичні алгоритми машинного навчання, які демонструють різні підходи до моделювання залежностей між текстовими ознаками та цільовими класами [10].

1.4.2 Наївний байєсьвмький класифікатор

Наївний байєсівський класифікатор представляє імовірнісний підхід до класифікації, заснований на теоремі Байєса та припущенні про умовну незалежність ознак. Попри математичну наївність цього припущення, яке рідко виконується на

практиці, алгоритм демонструє дивовижну ефективність на текстових даних. Мультиноміальна версія наївного Байєса особливо добре підходить для роботи з частотними представленнями текстів, де кожна ознака відповідає кількості появ конкретного слова або словосполучення [1].

Привабливість наївного Байєса полягає у швидкості навчання та передбачення. Обчислення апостеріорних ймовірностей зводиться до простих арифметичних операцій над попередньо розрахованими частотами появи ознак у кожному класі. Це робить алгоритм придатним для роботи з великими обсягами даних та високорозмірними просторами ознак, характерними для текстової класифікації. Додатковою перевагою є інтерпретованість моделі: можна проаналізувати, які слова найсильніше впливають на приналежність тексту до певного класу [13].

Обмеження наївного Байєса випливають з його основного припущення. Реальні тексти містять складні залежності між словами, які ігноруються моделлю. Наприклад, фраза "не добре" має протилежне значення порівняно з окремим словом "добре", проте наївний Байєс не враховує контекст заперечення. Також алгоритм вразливий до дисбалансу класів та потребує додаткових механізмів згладжування для обробки слів, які не зустрічалися в тренувальних даних [3].

1.4.3 Випадковий ліс

Випадковий ліс належить до сімейства ансамблевих методів, які об'єднують передбачення множини базових моделей для підвищення загальної точності. Базовими елементами випадкового лісу є дерева рішень, кожне з яких навчається на випадковій підвибірці даних та випадковій підмножині ознак. Фінальна класифікація визначається голосуванням більшості дерев, що забезпечує стійкість до шуму та перенавчання [7].

Дерева рішень природним чином моделюють нелінійні залежності через ієрархічну структуру послідовних поділів простору ознак. Кожен внутрішній вузол дерева відповідає умові на значення певної ознаки, а листя містять передбачення

класу. Для текстових задач ознаками виступають компоненти векторних представлень документів, і дерево вчиться комбінувати наявність або відсутність певних слів для прийняття рішення про клас [2].

Випадковість у виборі підмножин даних та ознак служить механізмом регуляризації, який запобігає надмірній адаптації окремих дерев до тренувальних даних. Різноманітність моделей в ансамблі дозволяє компенсувати помилки окремих дерев та покращити узагальнювальну здатність. Додатковою корисною властивістю випадкового лісу є можливість оцінити важливість кожної ознаки на основі того, наскільки часто вона використовується в поділах та наскільки значним є її внесок у зменшення помилки [34].

Недоліком випадкового лісу порівняно з наївним Байєсом є більша обчислювальна складність. Навчання множини дерев та зберігання їхніх структур вимагає значних ресурсів пам'яті та процесорного часу. Інтерпретація ансамблю також складніша, ніж окремого дерева чи імовірнісної моделі, хоча агреговані оцінки важливості ознак дають певне розуміння механізму роботи [15].

1.4.4 Метод опорних векторів

Метод опорних векторів представляє геометричний підхід до класифікації, заснований на пошуку оптимальної розділяючої гіперплощини між класами. Основна ідея полягає в максимізації відстані від гіперплощини до найближчих точок різних класів, які називаються опорними векторами. Така постановка задачі призводить до формулювання квадратичної оптимізаційної проблеми з обмеженнями [8].

Сила методу опорних векторів проявляється при роботі з високорозмірними даними, типовими для текстової класифікації. Векторні представлення документів можуть містити тисячі компонентів, проте SVM залишається ефективним завдяки тому, що рішення визначається лише опорними векторами, яких зазвичай значно менше за загальну кількість тренувальних прикладів. Це забезпечує компактність натренованої моделі та швидке передбачення [41].

Гнучкість методу опорних векторів досягається через використання kernel-функцій, які дозволяють працювати з нелінійними залежностями без явного перетворення даних у простір вищої розмірності. Лінійний kernel підходить для багатьох текстових задач, де класи добре розділяються в просторі частотних представлень. Для складніших випадків можна використовувати радіальні базисні функції або поліноміальні kernel'и [3].

Оригінальний алгоритм SVM розроблений для бінарної класифікації, тому для багатокласових задач необхідні додаткові стратегії. Найпоширенішим підходом є схема "один проти решти", коли для кожного класу навчається окремий бінарний класифікатор, що відокремлює його від всіх інших класів. Під час передбачення обчислюються оцінки від усіх класифікаторів, і обирається клас з найвищою впевненістю [13].

Складність налаштування гіперпараметрів є основним викликом при використанні методу опорних векторів. Параметр регуляризації C контролює баланс між максимізацією margin'у та мінімізацією помилок класифікації на тренувальних даних. Занадто мале значення C може призвести до недонавчання, тоді як занадто велике збільшує ризик перенавчання. Для kernel-функцій додаткові параметри, такі як gamma для RBF kernel, також вимагають ретельного підбору через cross-validation [37]. Порівняння цих алгоритмів зображено в таблиці 1.1.

1.4.5 Порівняльна таблиця алгоритмів

Вибір цих трьох алгоритмів для дослідження обумовлений їх комплементарністю. Наївний Байєс надає швидкий базовий результат та слугує еталоном для порівняння. Випадковий ліс дозволяє моделювати складні нелінійні залежності та оцінювати важливість ознак. Метод опорних векторів демонструє ефективність на високорозмірних даних та забезпечує хорошу узагальнювальну здатність. Порівняння результатів цих підходів дає комплексне розуміння властивостей даних та вимог конкретної задачі класифікації.

Таблиця 1.1.

Порівняння трьох алгоритмів

Аспект	**Naive Bayes (MultinomialNB)**	**Random Forest**	**SVM (One-vs-Rest)**
Підхід	Імовірнісний (теорема Байєса, незалежність ознак)	Ансамбль дерев рішень (bagging + random features)	Геометричний (максимізація margin), набір бінарних SVM у схемі OvR
Сильні сторони	Дуже швидке навчання та прогноз; добре працює з TF/TF-IDF; інтерпретованість	Стійкість до шуму; моделює нелінійності; оцінка важливості ознак	Добре працює у високорозмірних просторах; kernels дають гнучкість
Недоліки	Ігнорує залежності між словами; чутливий до дисбалансу	Повільніше навчання; висока потреба пам’яті; менш інтерпретований	Повільне навчання; складні гіперпараметри; OvR для багатокласовості
Складність навчання	Дуже низька	Помірно-висока	Висока (особливо з нелінійними kernels)
Складність інференсу	Дуже низька	Помірна (залежить від кількості дерев)	Помірна-висока (залежить від числа опорних векторів)
Інтерпретованість	Висока (ваги термінів)	Середня *(feature_importances)*	Низька/середня (kernel-моделі важко пояснити)
Ймовірності	Так	Так	Так (при *probability=True*)

продовження табл. 1.1.

Аспект	**Naive Bayes (MultinomialNB)**	**Random Forest**	**SVM (One-vs-Rest)**
Важливість ознак	Обмежена	Так *(feature_importances)*	Обмежена (для лінійних моделей)
Чутливість до масштабування	Низька	Низька/середня	Висока (обов'язкове масштабування для RBF/поліноміальних kernels)
Параметри	*alpha*	*n_estimators, max_depth, min_samples_split*	*C, kernel, gamma, degree, probability*
Коли обирати	Базова швидка модель; TF-IDF; великі текстові дані	Потрібна точність, робота з нелініями, оцінка важливості ознак	Високовимірні дані; задачі з kernel-розділюваністю
Масштабованість	Відмінна	Добра (дерева паралеляться)	Обмежена для нелінійних kernels; лінійний SVM масштабується добре

1.5 Висновки до першого розділу

Аналіз предметної області дозволив сформулювати чіткі вимоги до системи класифікації текстових рецензій та визначити оптимальні підходи до розв'язання поставлених задач. Багатокласова природа проблеми вимагає використання алгоритмів, здатних працювати з більш ніж двома категоріями одночасно, при цьому враховуючи специфіку текстових даних та їх високу розмірність.

Структура наявних даних виявилася достатньо багатою для формування трьох незалежних задач класифікації з різними цільовими змінними. Кожна задача відображає окремий аспект аналізу рецензій: пряму оцінку якості продукту, емоційне забарвлення відгуку та його корисність для інших користувачів. Така багатоаспектність аналізу підвищує практичну цінність системи та дозволяє отримати комплексне розуміння характеристик рецензій.

Обрані методи формування цільових міток базуються на об'єктивних критеріях та статистичних підходах, що забезпечує відтворюваність результатів та можливість автоматизації процесу підготовки даних для навчання моделей. Використання квартильного розподілу для категоризації корисності рецензій дозволяє адаптуватися до природного розподілу даних та уникнути проблем дисбалансу класів.

Три обрані алгоритми машинного навчання представляють різні парадигми моделювання: імовірнісний підхід, ансамблеві методи та геометричну оптимізацію. Їх комбінація забезпечує можливість всебічного дослідження властивостей даних та вибору найефективнішого рішення для кожної конкретної задачі класифікації. Класичні методи зберігають актуальність завдяки балансу між якістю результатів та обчислювальною ефективністю.

Розуміння особливостей текстових даних, зокрема їх розрідженості та високої розмірності, визначає вимоги до подальших етапів передобробки та витягу ознак. Необхідність роботи з природною мовою користувачів, яка містить граматичні помилки та розмовні вирази, підкреслює важливість надійних методів нормалізації та очищення текстів перед їх використанням для навчання моделей.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ КЛАСИФІКАЦІЇ РЕЦЕНЗІЙ

2.1 Модульна архітектура програмної системи класифікації

2.1.1 Принципи проектування

Розробка архітектури системи класифікації текстових рецензій вимагала ретельного планування структури модулів та їх взаємодії.

Основним принципом проектування стало розділення відповідальності між компонентами, що забезпечує гнучкість системи та спрощує її подальше розширення. Кожен модуль відповідає за конкретний етап обробки даних, від завантаження сирих записів з бази до генерування передбачень натренованими моделями.

2.1.2 Потоки обробки даних

Верхній рівень архітектури організовано навколо трьох основних потоків роботи: підготовка даних, навчання моделей та використання натренованих класифікаторів для передбачень. Ці потоки можуть виконуватися незалежно один від одного, що дозволяє гнучко керувати життєвим циклом моделей та експериментувати з різними конфігураціями без необхідності перезапуску всього конвеєра обробки.

2.1.3 Репозиторний патерн

Модуль доступу до даних інкапсулює всю логіку взаємодії з SQLite базою даних. Використання репозиторного патерну дозволяє абстрагуватися від деталей зберігання даних та надає уніфікований інтерфейс для отримання рецензій. Це рішення спрощує можливу міграцію на іншу систему зберігання даних у майбутньому, оскільки зміни торкнуться лише одного модуля без впливу на решту системи [42].

2.1.4 Підсистеми передобробки

Підсистема передобробки розділена на два незалежних компоненти для роботи з текстовими та числовими даними відповідно. Така сегрегація обумовлена фундаментальними відмінностями в природі цих типів інформації та методах їх обробки. Текстові дані проходять через послідовність трансформацій, спрямованих на нормалізацію та очищення, тоді як числові характеристики вимагають обробки відсутніх значень та обчислення похідних метрик.

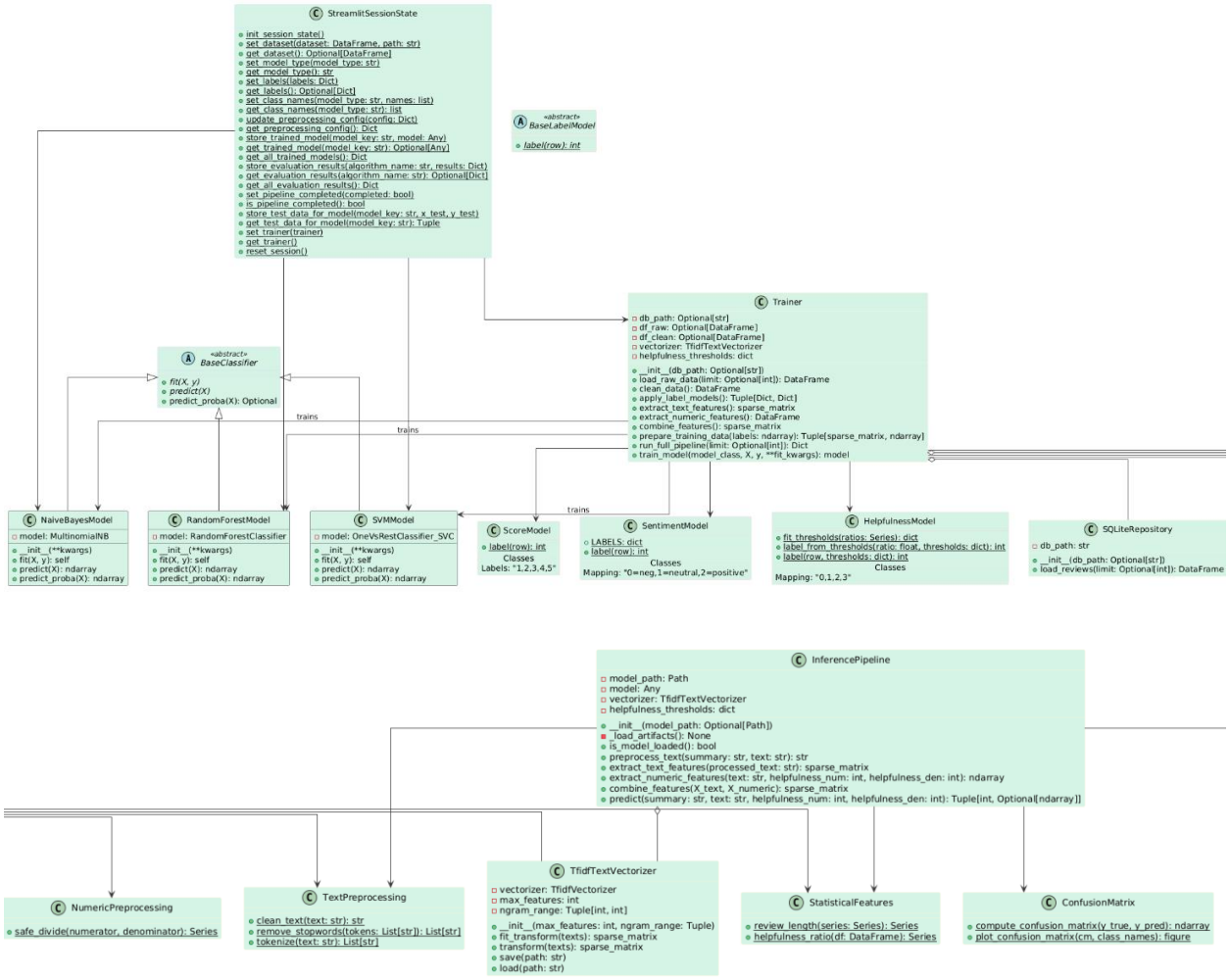


Рис. 2.1. Діаграма класів

Модулі витягу ознак реалізують різні стратегії перетворення оброблених даних у числові представлення, придатні для навчання моделей машинного навчання. Векторизація тексту за допомогою TF-IDF перетворює документи у високорозмірні

розріджені вектори, тоді як статистичні ознаки формують компактний набір числових характеристик. Механізм комбінування цих гетерогенних типів ознак забезпечує створення єдиного простору представлень для навчання класифікаторів. Діаграму класів зображено на рисунку 2.1.

2.1.5 Тренування підсистеми

Центральним компонентом архітектури є тренувальна підсистема, яка координує всі етапи підготовки моделей. Вона керує послідовністю операцій від завантаження сирих даних до збереження натренованих класифікаторів та їх метрик якості. Використання об'єктно-орієнтованого підходу дозволяє легко додавати нові алгоритми класифікації через успадкування від базового класу моделі, який визначає єдиний інтерфейс для всіх реалізацій [13].

Підсистема генерування міток інкапсулює логіку перетворення вихідних полів рецензій у категоріальні цільові змінні. Кожна label model є незалежним класом, що реалізує специфічні правила категоризації для відповідної задачі класифікації. Така модульна організація дозволяє експериментувати з альтернативними стратегіями формування міток без впливу на інші компоненти системи.

Модуль оцінювання якості забезпечує обчислення різноманітних метрик та побудову діагностичних візуалізацій. Відокремлення цього функціоналу від основної логіки тренування дозволяє використовувати одні й ті ж функції оцінки для різних моделей та задач класифікації. Результати оцінювання зберігаються у структурованому вигляді для подальшого порівняння ефективності різних

підходів.

Інференс-модуль відповідає за використання натренованих моделей для класифікації нових рецензій. Він завантажує збережені артефакти, включаючи натреновані класифікатори та fitted векторизатори, та застосовує ту ж послідовність трансформацій до вхідних даних, що використовувалася під час навчання.

Дотримання ідентичності конвеєрів передобробки є критичним для коректності передбачень [3].

Веб-інтерфейс на базі Streamlit надає зручний спосіб взаємодії з усією функціональністю системи через браузер. Архітектура додатку побудована на принципі реактивності, де зміна параметрів автоматично тригерить перерахунок відповідних компонентів. Управління станом сесії користувача дозволяє зберігати проміжні результати обчислень та уникати повторного виконання ресурсомістких операцій. Сценарій використання є на рисунку 2.2.

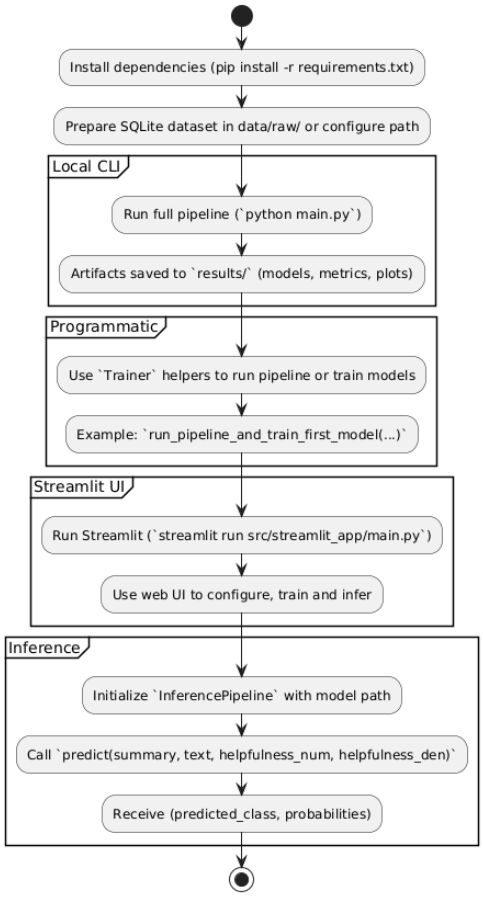


Рис. 2.2. Сценарій використання

Організація файлової системи проекту відображає логічну структуру компонентів. Вихідний код розділено на тематичні піддиректорії всередині головної папки src, де кожна піддиректорія містить пов'язані між собою модулі. Артефакти роботи системи, такі як натреновані моделі, метрики та візуалізації, зберігаються в

окремій ієрархії директорій results, що дозволяє легко керувати версіями моделей та відстежувати результати експериментів.

Точка входу через командний рядок реалізована у головному файлі main.py, який оркеструє виконання повного конвеєра обробки даних. Альтернативний шлях взаємодії через Streamlit додаток забезпечує більш інтуїтивний інтерфейс для користувачів, які віддають перевагу візуальній навігації та інтерактивному налаштуванню параметрів над командним рядком [23].

Залежності між модулями організовано таким чином, щоб мінімізувати циклічні посилання та дотримуватися принципу односпрямованого потоку даних. Модулі нижчого рівня, такі як доступ до даних та передобробка, не мають залежностей від модулів вищого рівня, що полегшує незалежне тестування та повторне використання коду в інших проектах.

2.2 Проектування підсистем передобробки текстових та числових даних

2.2.1 Трансформації текстових даних

Передобробка даних становить критичний етап конвеєра машинного навчання, оскільки якість вхідних даних безпосередньо впливає на ефективність навчання моделей. Проектування модулів передобробки орієнтувалося на створення універсальних, але гнучких компонентів, здатних обробляти різноманітні особливості користувацького контенту при збереженні відтворюваності результатів [27].

2.2.2 Нормалізація та токенізація

Модуль передобробки тексту реалізує послідовність трансформацій, що перетворюють сирі текстові дані у нормалізовану форму. Першим кроком виконується приведення всіх символів до нижнього регістру через операцію lowercase. Це усуває штучну різницю між словами, що відрізняються лише капіталізацією, та зменшує розмірність простору ознак без втрати семантичної інформації. Наприклад, слова "Great", "GREAT" та "great" після цієї трансформації стають єдиною ознакою.

Наступним етапом є видалення пунктуації та спеціальних символів з тексту. Це досягається застосуванням регулярного виразу, який залишає лише літери, цифри та пробіли. Рішення замінювати небажані символи на пробіли замість їх простого видалення запобігає злиттю окремих слів у випадках, коли пунктуація стоїть без проміжків. Математично ця операція описується як застосування функції заміни символів:

$$text = re.sub(r"[^a\text{-}z0\text{-}9\backslash s]", " ", text) \quad (2.1)$$

Вираз *[^a-z0-9\s]* задає множину всіх символів, які не належать до латинських літер нижнього регістру, цифр або пробільних символів. Усі такі символи замінюються на пробіл.

Фінальна стадія текстового очищення полягає у нормалізації пробілів. Множинні послідовні пробіли, які могли з'явитися після видалення пунктуації, замінюються одинарними пробілами. Додатково видаляються початкові та кінцеві пробіли через методи *strip*. Ця операція забезпечує однорідність форматування та коректність подальшої токенізації:

$$text = re.sub(r"\backslash s+", " ", text).strip() \quad (2.2)$$

Токенізація розбиває очищений текст на окремі слова через поділ за пробілами. Отриманий список токенів піддається додатковій фільтрації для видалення стоп-слів. Стоп-слова представляють собою високочастотні службові слова англійської мови, такі як артиклі, прийменники та сполучники, які несуть мало семантичного навантаження для задачі класифікації. Видалення цих слів зменшує розмірність даних та дозволяє моделям концентруватися на інформативних термінах [29].

Список стоп-слів завантажується з бібліотеки NLTK, яка містирує кураторський перелік найпоширеніших службових слів. Функція remove_stopwords

приймає список токенів та повертає відфільтровану версію, де залишаються лише значущі слова. Це простий, але ефективний механізм зменшення шуму в текстових даних без необхідності складного синтаксичного аналізу.

2.2.3 Обчислення метрик корисності

Модуль передобробки числових даних фокусується на обробці метрик корисності рецензій. Центральною проблемою тут є безпечне обчислення відношень при можливості нульового знаменника. Функція *safe_divide* реалізує захищене ділення, яке повертає 0 у випадках, коли рецензія не має жодної оцінки корисності:

$$\begin{cases} helpdulness_{ratio} = \frac{HelpfulnessNumerator}{HelpfulnessDenomirator}, \text{ якщо } HelpfulnessDenominator > 0 \\ helpfulness_{ratio} = 0, \text{ якщо } HelpfulnessDenominator = 0 \end{cases} \quad (2.3)$$

де *HelpfulnessNumerator* та *HelpfulnessDenominator* — це змінні, які використовуються для оцінювання корисності відгуку іншими користувачами, *helpfulness_ratio* показує частку користувачів, які визнали відгук корисним.

Практична реалізація використовує можливості бібліотеки pandas для векторизованої обробки цілих стовпців даних одночасно. Знаменник замінюється на спеціальне значення NA у місцях, де він дорівнює нулю, після чого виконується ділення з подальшим заповненням отриманих NA нулями. Такий підхід уникає явних циклів та забезпечує високу продуктивність обчислень на великих обсягах даних [25].

2.2.4 Статистичні характеристики документів

Обчислення додаткових статистичних характеристик включає визначення довжини рецензії. Довжина може вимірюватися як кількість символів або кількість слів, залежно від конкретних вимог. У реалізації підраховується кількість слів після токенізації, що дає більш стабільну метрику, незалежну від варіацій у використанні

пробілів користувачами. Гіпотеза полягає в тому, що довжина рецензії може корелювати з її детальністю та корисністю.

Абсолютні значення числівника та знаменника корисності також включаються як окремі ознаки. Числівник показує, скільки користувачів відзначили рецензію як корисну, що може вказувати на її популярність або видимість. Знаменник відображає загальну кількість оцінок, що характеризує рівень уваги до рецензії з боку спільноти. Ці метрики доповнюють відносний показник корисності та надають моделям додатковий контекст [40].

Проектне рішення щодо збереження всіх числових ознак у форматі DataFrame забезпечує зручність подальшого маніпулювання та комбінування з іншими типами ознак. Використання іменованих стовпців підвищує читабельність коду та зменшує ймовірність помилок при доступі до конкретних характеристик. Консистентність форматів даних між різними модулями спрощує інтеграцію компонентів у єдиний конвеєр.

Обидва модулі передобробки розроблено як функції чистої трансформації, які не мають побічних ефектів та завжди повертають однакові результати для ідентичних вхідних даних. Така функціональна парадигма забезпечує передбачуваність поведінки та спрощує тестування окремих компонентів незалежно від решти системи. Відтворюваність результатів є критичною вимогою для науково-дослідницьких проектів у галузі машинного навчання [37].

2.3 Механізми витягу та об’єднання гетерогенних типів ознак

2.3.1 основи TF-IDF векторизації

Перетворення оброблених текстів у числові векторні представлення здійснюється через механізм TF-IDF векторизації. Цей підхід базується на двох фундаментальних ідеях: частота терміну в документі та рідкість терміну в колекції документів. Term Frequency вимірює локальну важливість слова в конкретному

тексті, тоді як Inverse Document Frequency відображає глобальну інформативність слова в усьому корпусі [4].

Математична формалізація TF-IDF складається з двох компонентів. Частота терміну обчислюється як відношення кількості появ слова до загальної кількості слів у документі:

TF(t,d) = (частота терміну *t* в документі *d*) / (загальна кількість термінів в *d*)

Обернена частота документів відображає рідкість терміну через логарифмічне масштабування відношення загальної кількості документів до кількості документів, що містять термін. Додавання одиниці в знаменнику запобігає діленню на нуль для термінів, відсутніх у корпусі:

$$IDF(t) = log(N / (1 + DF(t))) \quad (2.4)$$

де N означає загальну кількість документів, а $DF(t)$ позначає кількість документів, що містять термін t.

Фінальне значення TF-IDF отримується як добуток цих двох компонентів:

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t) \quad (2.5)$$

Ця метрика дозволяє виявляти слова, які є специфічними для конкретного документа, відфільтровуючи загальновживані терміни, що зустрічаються скрізь.

2.3.2 Параметризація веторизатора

Проектування TF-IDF векторизатора включає вибір декількох критичних гіперпараметрів. Параметр max_features обмежує кількість унікальних термінів у словнику, залишаючи лише найчастіші слова. У цьому проекті обрано значення 20000, що забезпечує баланс між повнотою представлення та обчислювальною ефективністю. Збільшення цього параметру покращує охоплення словника за рахунок зростання розмірності та споживання пам'яті [38].

Діапазон n-грам визначає, які послідовності слів розглядаються як окремі ознаки. Конфігурація ngram_range=(1,2) означає включення як окремих слів (уніграм), так і пар послідовних слів (біграм). Біграми захоплюють деякий локальний контекст та фразеологізми, які можуть мати значення, відмінне від значень окремих слів. Наприклад, біграма "not good" несе негативний сентимент, хоча окреме слово "good" є позитивним [43].

Результатом векторизації є розріджена матриця формату CSR (Compressed Sparse Row), де рядки відповідають документам, а стовпці відповідають термінам зі словника. Більшість елементів матриці дорівнюють нулю, оскільки кожен окремий документ містить лише невелику підмножину всіх можливих термінів. Розріджений формат зберігає тільки ненульові елементи, що драматично зменшує вимоги до пам'яті порівняно з щільним представленням [26].

2.3.3 Статистичні числові ознаки

Витяг статистичних числових ознак доповнює текстові представлення агрегованими характеристиками рецензій. Модуль обчислює чотири базові метрики: довжину рецензії, числівник корисності, знаменник корисності та відношення корисності. Кожна з цих ознак надає моделям інформацію, яка не міститься безпосередньо в текстовому вмісті, але може корелювати з цільовими змінними класифікації.

2.3.4 Комбінування матриць

Комбінування гетерогенних типів ознак вимагає узгодження їх форматів та розмірностей. TF-IDF матриця має розріджений формат та високу розмірність, тоді як статистичні ознаки представлені щільним DataFrame з невеликою кількістю стовпців. Процес об'єднання включає перетворення числових ознак у розріджену матрицю з використанням функції csr_matrix, після чого обидві матриці конкатенуються горизонтально через scipy.sparse.hstack [26].

Математично операція комбінування описується як горизонтальне об'єднання матриць:

$$X\_combined = hstack([X\_text, X\_numeric]) \quad (2.6)$$

де *X_text* є матрицею розмірності *(n_samples, 20000)*, а *X_numeric* має розмірність *(n_samples, 4)*. Результуюча матриця *X_combined* має розмірність *(n_samples, 20004)* та зберігає розріджений формат, якщо переважна більшість елементів залишаються нульовими.

2.3.5 Оптимізація пам’яті

Збереження комбінованих ознак у розрідженому форматі є критичним для ефективності пам'яті. Щільне представлення матриці розмірності 10000×20004 з елементами float64 потребувало б близько 1.5 гігабайт оперативної пам'яті, тоді як розріджене зберігання зменшує це значення на порядки величини. Така економія дозволяє працювати з великими наборами даних на звичайному обладнанні без серверів з великими обсягами RAM [13].

Проектне рішення про збереження всіх проміжних артефактів на диск забезпечує можливість відновлення обчислень з будь-якого етапу конвеєра без повторного виконання попередніх кроків. Векторизатор зберігається у серіалізованому вигляді через бібліотеку pickle, що дозволяє застосовувати його до нових даних з гарантією ідентичності трансформацій. Матриці ознак зберігаються у форматі npz, який нативно підтримує розріджені структури даних.

Архітектура модулів витягу ознак спроектована з врахуванням можливості розширення. Додавання нових типів ознак, таких як embeddings з попередньо натренованих мовних моделей або додаткові статистичні метрики, вимагає лише створення відповідних функцій витягу та їх інтеграції у процес комбінування. Уніфікований формат вихідних даних спрощує цю інтеграцію та зберігає сумісність з існуючими компонентами системи.

2.4 Конвеєр навчання та валідації класифікаційних моделей

2.4.1 Етапи підготовки даних

Конвеєр тренування організовано як послідовність чітко визначених етапів, кожен з яких відповідає за специфічну трансформацію даних або операцію навчання. Центральним компонентом є клас Trainer, який оркеструє виконання всіх кроків та керує потоком даних між модулями. Такий централізований підхід забезпечує консистентність виконання та спрощує налагодження проблем у складному багатоетапному процесі [15].

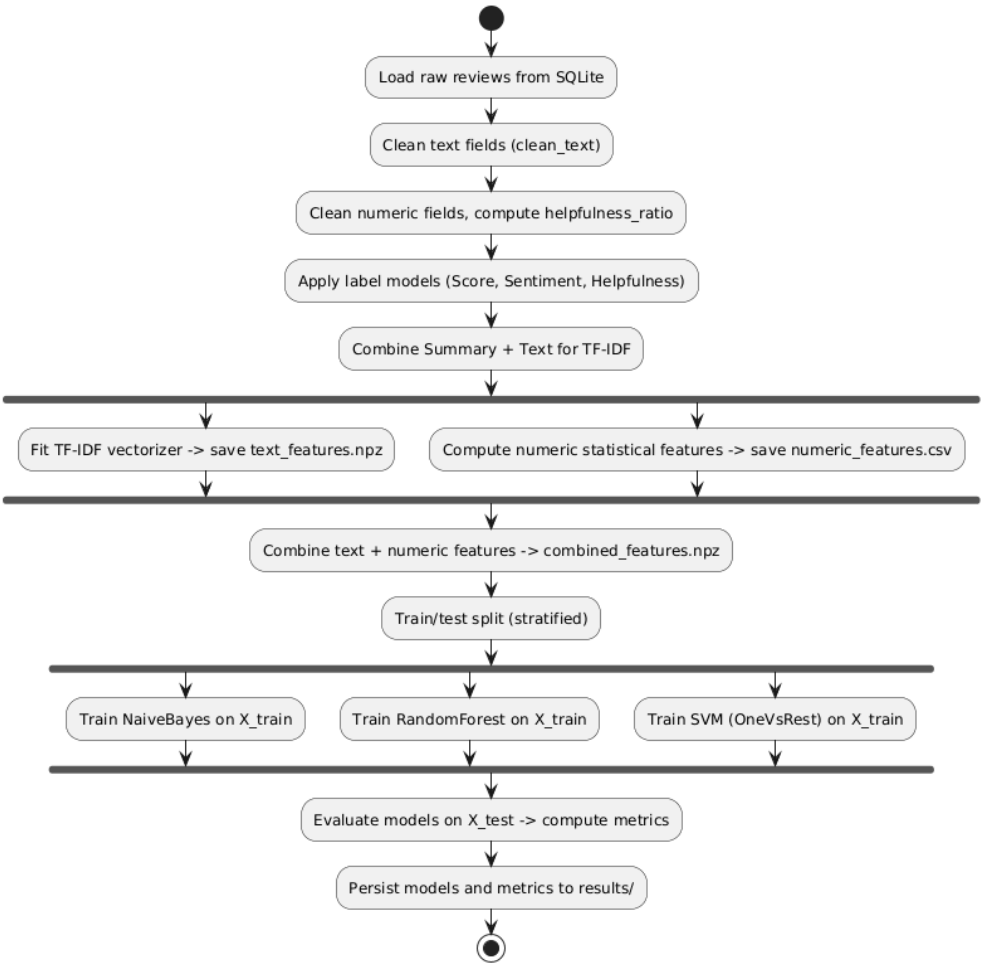


Рис. 2.3. Діаграма тренування активності

Початкова фаза конвеєра включає завантаження сирих даних з бази через репозиторій. Параметр limit дозволяє обмежити кількість завантажуваних записів для

прискорення експериментів під час розробки та налагодження. Для повноцінного тренування моделей завантажується весь доступний набір даних, що забезпечує максимальну репрезентативність тренувальної вибірки.

Застосування передобробки до текстових та числових полів виконується незалежно через відповідні модулі. Очищені дані зберігаються у проміжному форматі, що дозволяє перевірити коректність трансформацій та використати ці дані для декількох експериментів без повторної передобробки. Збереження проміжних результатів також полегшує відновлення процесу у випадку збоїв на пізніших етапах.

Генерування цільових міток відбувається через застосування label models до оброблених даних. Кожна з трьох моделей генерує свій стовпець міток, що додаються до датафрейму. Результуючий набір даних містить всі необхідні цільові змінні для тренування окремих класифікаторів під кожну задачу. Такий підхід дозволяє уникнути дублювання обчислень та зберігає узгодженість між різними версіями міток [2]. Функціонал тренування активності зображено на рис. 2.3.

Витяг ознак виконується паралельно для текстових та числових компонентів. TF-IDF векторизатор навчається на текстовому корпусі, будуючи словник та обчислюючи IDF статистики. Цей fitted векторизатор зберігається для подальшого застосування до нових даних під час інференсу. Числові ознаки обчислюються безпосередньо з відповідних полів датафрейму без необхідності попереднього навчання трансформерів.

Комбінування ознак створює єдиний простір представлень для всіх рецензій. Отримана матриця стає вхідними даними для алгоритмів машинного навчання. Розмірність цього простору визначається сумою кількості текстових термінів та кількості числових ознак, що в даному проекті становить 20004 ознаки.

2.4.2 Стратифіковане розділення даних

Розділення даних на тренувальну та тестову вибірки здійснюється стратифікованим методом, який зберігає пропорції класів у обох підмножинах. Це

особливо важливо для задач з дисбалансом класів, оскільки гарантує репрезентативність тестової вибірки. Співвідношення 80/20 є стандартним компромісом між розміром тренувальних даних та надійністю оцінки якості [37].

2.4.3 Математичні формалізації алгоритмів

Тренування окремих класифікаторів виконується для кожної комбінації алгоритму та типу моделі. Наївний байєсівський класифікатор використовує мультиноміальну версію, призначену для роботи з частотними представленнями тексту. Математична основа алгоритму базується на теоремі Байєса (2.7) з припущенням умовної незалежності ознак:

$$P(y \mid x_1..x_n) \propto P(y) \times \Pi P(x_i \mid y) \quad (2.7)$$

де:

- *P(y∣x1,…,xn)P(y \mid x_1, \ldots, x_n)P(y∣x1,…,xn)* — апостеріорна ймовірність класу *y* після спостереження ознак;
- *P(y)P(y)P(y)* — апріорна ймовірність класу, що відображає його частоту в навчальній вибірці;
- *P(x1,…,xn∣y)P(x_1, \ldots, x_n \mid y)P(x1,…,xn∣y)* — правдоподібність появи набору ознак за умови класу yyy;
- *P(x1,…,xn)P(x_1, \ldots, x_n)P(x1,…,xn)* — ймовірність спостереження ознак, незалежно від класу.

У лог-просторі це перетворюється на більш стабільні обчислення (2.8):

$$Log\ P(y|x) = log\ P(y) + \Sigma x_i \times log\ P(t_i \mid y) \quad (2.8)$$

Оцінка умовних ймовірностей термінів виконується з адитивним згладжуванням, яке запобігає нульовим ймовірностям для термінів, відсутніх у тренувальних даних певного класу (2.9):

$$P(t \mid y) = (N_{t,y} + \alpha) / (N_y + \alpha \cdot V) \quad (2.9)$$

де α є параметром згладжування, зазвичай рівним одиниці, $N_{t,y}$ означає кількість поява терміну в класі, а V позначає розмір словника [1]. Випадковий ліс будує ансамбль дерев рішень, кожне з яких навчається на бутстреп-вибірці даних з випадковою підмножиною ознак у кожному вузлі. Фінальне передбачення формується голосуванням дерев (2.10):

$$f(x) = majority\_vote(\{T_b(x)\}_{b=1..B}) \quad (2.10)$$

або у вигляді усереднених ймовірностей (2.11):

$$p_k(x) = (1/B) \times \Sigma_{b=1..B} p_k^{(b)}(x) \quad (2.11)$$

Параметри алгоритму, такі як кількість дерев n_estimators та максимальна глибина max_depth, впливають на баланс між точністю та ризиком перенавчання. Більша кількість дерев покращує стабільність передбачень, але збільшує обчислювальні витрати [7]. Метод опорних векторів шукає оптимальну гіперплощину для розділення класів через розв'язання оптимізаційної задачі. Для лінійного випадку рішення описується як (2.12):

$$f(x) = sign(w^T \varphi(x) + b) \quad (2.12)$$

Навчання зводиться до мінімізації функції втрат з регуляризацією (2.13):

$$minimize_{w,b,\xi} (1/2) ||w||^2 + C \Sigma \xi_i \quad (2.13)$$

Параметр C контролює компроміс між максимізацією margin та мінімізацією помилок класифікації. Для багатокласової задачі застосовується схема One-vs-Rest,

яка навчає окремий бінарний класифікатор для кожного класу проти решти [8]. Діаграма швидкого старту зображено на рисунку 2.4.

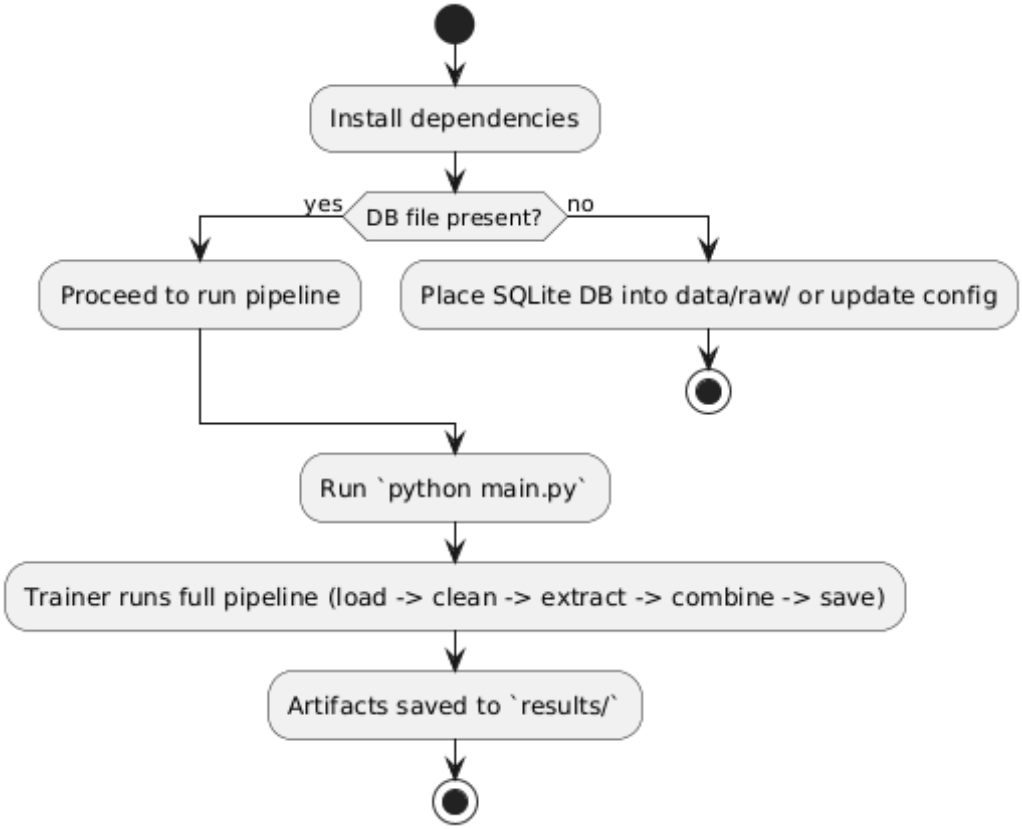


Рис. 2.4. Діаграма швидкого старту

2.4.4 Збереження артефактів моделей

Збереження натренованих моделей здійснюється через серіалізацію об'єктів у файли pickle. Кожна комбінація алгоритму та типу задачі зберігається окремо з описовою назвою файлу, що включає обидва ці параметри. Така організація дозволяє легко завантажувати конкретні моделі для використання або порівняння результатів.

2.4.5 Метрики оцінювання якості

Оцінювання якості виконується на тестовій вибірці, яка не використовувалася під час навчання. Для кожної моделі обчислюється набір метрик, що характеризують різні аспекти ефективності класифікації. *Accuracy* (2.14) відображає загальну частку правильних передбачень:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad (2.14)$$

де *TP* (true positives) та *TN* (true negatives) — кількість правильних позитивних і негативних передбачень відповідно, а *FP* (false positives) і *FN* (false negatives) — кількість помилкових передбачень. *Accuracy* є інформативною метрикою за умови відносно збалансованих класів, проте може вводити в оману при значній класовій нерівновазі.

Precision (2.15) для кожного класу показує, яка частка об'єктів, класифік ованих як належні до цього класу, справді належить йому:

$$Precision_i = TP_i / (TP_i + FP_i) \quad (2.15)$$

Recall (2.16) характеризує повноту виявлення об'єктів класу:

$$Recall_i = TP_i / (TP_i + FN_i) \quad (2.16)$$

F1-міра (2.17) є гармонічним середнім precision та recall:

$$F1_i = 2 \times (Precision_i \times Recall_i) / (Precision_i + Recall_i) \quad (2.17)$$

Macro-averaging обчислює середнє значення метрики по всім класам, надаючи рівну вагу кожному класу незалежно від його частоти у даних. Такий підхід важливий для виявлення проблем з рідкісними класами, які можуть бути проігноровані при використанні звичайної accuracy [13].

Матриця плутанини візуалізує детальний розподіл передбачень по всім парам класів. Рядки матриці відповідають справжнім класам, а стовпці передбаченим. Діагональні елементи показують правильні класифікації, тоді як позадіагональні елементи вказують на конкретні типи помилок. Аналіз матриці плутанини допомагає виявити систематичні проблеми, такі як постійне змішування певних пар класів.

Результати оцінювання зберігаються у структурованому форматі JSON для подальшого аналізу та порівняння. Кожен файл метрик містить всі обчислені показники якості для відповідної моделі. Така організація даних полегшує створення зведених таблиць та графіків порівняння ефективності різних підходів на різних задачах класифікації.

2.5 Висновок до другого розділу

Проектування архітектури системи багатокласової класифікації текстових рецензій базувалося на принципах модульності, розширюваності та відтворюваності результатів. Розділення функціональності між незалежними компонентами дозволяє гнучко керувати складністю системи та спрощує її подальший розвиток. Кожен модуль інкапсулює конкретну відповідальність та надає чітко визначений інтерфейс для взаємодії з іншими частинами системи.

Модулі передобробки текстових та числових даних реалізують надійні методи нормалізації та очищення вхідної інформації. Використання регулярних виразів для текстової обробки та векторизованих операцій для числових даних забезпечує високу продуктивність на великих обсягах інформації. Безпечна обробка граничних випадків, таких як ділення на нуль або відсутні значення, гарантує стабільність роботи системи на реальних даних різної якості.

Механізми витягу ознак успішно перетворюють різнорідні типи даних у єдиний числовий простір представлень. TF-IDF векторизація захоплює семантичну інформацію з текстів через врахування як локальної частоти термінів, так і їх глобальної рідкості. Комбінування текстових та статистичних ознак збагачує представлення рецензій та надає моделям доступ до різноманітних джерел інформації для прийняття рішень.

Структура конвеєра тренування забезпечує систематичне виконання всіх етапів підготовки моделей від завантаження сирих даних до збереження натренованих класифікаторів та їх метрик якості. Стратифіковане розділення даних

та використання стандартних метрик оцінювання гарантують коректність експериментальної методології. Збереження всіх проміжних артефактів дозволяє відновлювати обчислення з будь-якої точки та забезпечує повну відтворюваність результатів.

Математичне обґрунтування використаних алгоритмів класифікації демонструє різноманітність підходів до моделювання залежностей між ознаками та цільовими класами. Наївний Байєс представляє імовірнісну парадигму, випадковий ліс реалізує ансамблеві методи, а метод опорних векторів базується на геометричній оптимізації. Така різноманітність дозволяє всебічно дослідити властивості даних та вибрати оптимальні рішення для кожної конкретної задачі.

Проектні рішення щодо організації файлової структури та форматів зберігання даних забезпечують ефективне використання ресурсів та зручність роботи з артефактами системи. Використання розріджених форматів для великих матриць ознак драматично зменшує вимоги до пам'яті, тоді як серіалізація натренованих моделей дозволяє швидко завантажувати їх для використання без повторного навчання.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ

3.1 Програмна реалізація компонентів системи класифікації

3.1.1 Технологічний стек

Програмна реалізація системи класифікації текстових рецензій виконана мовою Python версії 3.x з використанням сучасних бібліотек машинного навчання та обробки даних. Вибір Python обумовлений його потужною екосистемою інструментів для data science, зручним синтаксисом та широкою підтримкою спільноти розробників. Основу функціональності складають бібліотеки scikit-learn для алгоритмів класифікації, pandas для маніпуляції даними, numpy для числових обчислень та NLTK для обробки природної мови [13, 22].

3.1.2 Репозиторій SQLite

На лістингу 3.1 показано модуль доступу до даних реалізовано через клас SQLiteRepository, який інкапсулює всю логіку взаємодії з базою даних. Клас надає метод load_reviews, що приймає опціональний параметр обмеження кількості записів та повертає DataFrame з рецензіями. Використання контекстних менеджерів для роботи з з'єднаннями до бази гарантує коректне закриття ресурсів навіть у випадку виникнення помилок під час виконання запитів.

Лістинг 3.1.

```
import sqlite3
import pandas as pd
from typing import Optional
from ..utils.config import RAW_DB, SQL_TABLE
class SQLiteRepository:
    def __init__(self, db_path: Optional[str] = None):
        """Initialize repository.
        Parameters
```

```
            db_path : Optional[str]
                Optional path to the SQLite database file. If not
provided,
                the default path from `RAW_DB` will be used.
            """
            self.db_path = db_path or str(RAW_DB)

        def load_reviews(self, limit: Optional[int] = None) ->
pd.DataFrame:
            """Load the reviews table from the sqlite database.
            Parameters
            limit : int, optional
                Maximum number of rows to load. If None, load all
rows.
            Returns
            -------
            pd.DataFrame
            # Create a connection to the SQLite database
            conn = sqlite3.connect(self.db_path)
            try:
                # Use parameterized query for safety when using raw
values;
                # here table name is from config so it's acceptable
to format.
                if limit is not None:
                    query = f"SELECT * FROM {SQL_TABLE} LIMIT ?"
                    df = pd.read_sql_query(query, conn,
params=(limit,))
                else:
                    query = f"SELECT * FROM {SQL_TABLE}"
                    df = pd.read_sql_query(query, conn)
            finally:
                conn.close()
            return df
```

3.1.3 Модуль передобробки тексту

Реалізація передобробки текстових даних сконцентрована в модулі text_preprocessing.py. Функція clean_text приймає сирий текст та послідовно застосовує до нього набір трансформацій. Спочатку весь текст приводиться до нижнього регістру методом lower, що усуває чутливість до капіталізації. Потім регулярний вираз видаляє всі символи, окрім літер, цифр та пробілів. Фінальним кроком множинні пробіли нормалізуються до одинарних, а початкові та кінцеві пробіли видаляються.

Функція tokenize комбінує очищення тексту з його розбиттям на окремі слова та фільтрацією стоп-слів. Завантаження списку стоп-слів з NLTK відбувається один раз при імпорті модуля для оптимізації продуктивності. Функція remove_stopwords використовує list comprehension для ефективної фільтрації токенів, залишаючи лише ті слова, які не входять до переліку службових слів англійської мови [29].

3.1.4 Обробка числових характеристик

Модуль числової передобробки містить функцію safe_divide, яка забезпечує безпечне обчислення відношення корисності. Реалізація використовує можливості pandas для векторизованих обчислень, замінюючи нульові значення знаменника на спеціальне значення NA перед виконанням ділення. Результуючі NA значення заповнюються нулями, що відображає відсутність інформації про корисність для рецензій без оцінок. Код safe_divide наведено в лістингу 3.2.

Лістинг 3.2.

```
import pandas as pd
from typing import Union, Iterable

def safe_divide(numerator: Union[pd.Series, Iterable, float,
int], denominator: Union[pd.Series, Iterable, float, int]) ->
pd.Series:
    """Calculate safe division handling division by zero.
```

```
        This function accepts either pandas Series or array-like /
scalar inputs.
        It returns a pandas Series of floats where division by zero
(or invalid)
        yields 0. The function preserves the index when a Series is
provided.

        Parameters
        ----------
        numerator : pandas.Series or array-like or scalar
            Numerator values.
        denominator : pandas.Series or array-like or scalar
            Denominator values.

        Returns
        -------
        pandas.Series
            Series of resulting ratios (float), with zeros where
division
            was invalid or denominator was zero.
        """
        # Convert inputs to Series if they are not already; preserve
index if present
        if isinstance(numerator, pd.Series):
            num = numerator.astype(float)
        else:
            num = pd.Series(numerator, dtype=float)

        if isinstance(denominator, pd.Series):
            den = denominator.astype(float)
        else:
            den = pd.Series(denominator, dtype=float)
        # Replace exact zeros in denominator with NaN to avoid
division-by-zero
```

```
        den_safe = den.replace(0, pd.NA)

        # Perform division (pandas will produce NA for invalid
divisions)
        ratio = num / den_safe

        # Replace NaN or infinite results with 0
        ratio = ratio.fillna(0)
        ratio = ratio.replace([pd.NA], 0)

        # In case there are infinities (shouldn't be after NA
handling), coerce them
        ratio = ratio.replace([float('inf'), -float('inf')], 0)

        # If numerator had an index, ensure the result keeps it;
otherwise a default RangeIndex is used
        return pd.Series(ratio.values, index=num.index, dtype=float)
```

Витяг текстових ознак реалізовано через клас TfidfTextVectorizer, який обгортає функціональність TfidfVectorizer з scikit-learn. Клас ініціалізується з параметрами max_features та ngram_range, які визначають розмірність простору ознак та типи n-грам відповідно. Метод fit_transform навчає векторизатор на корпусі текстів та одночасно перетворює їх у TF-IDF представлення. Збережений векторизатор може застосовуватися до нових текстів через метод transform без повторного навчання [24].

Обчислення статистичних числових ознак виконується функціями модуля statistical_features.py. Функція extract_statistical_features приймає DataFrame з рецензіями та повертає новий DataFrame з чотирма стовпцями ознак. Довжина рецензії обчислюється підрахунком кількості слів після токенізації. Числівник та знаменник корисності беруться безпосередньо з відповідних полів. Відношення корисності обчислюється через виклик функції safe_divide з відповідними стовпцями як аргументами.

3.1.5 Генератори цільових міток

Модуль label_models містить три класи для генерування цільових міток. Кожен клас успадковується від базового класу LabelModel, який визначає абстрактний метод generate_label. Клас ScoreModel реалізує найпростішу логіку, безпосередньо повертаючи значення поля Score як мітку класу. Клас SentimentModel використовує умовну логіку для відображення числових оцінок у три категорії сентименту. Клас HelpfulnessModel обчислює квартилі розподілу відношення корисності та використовує їх як пороги для категоризації. Код генерації цільових міток наведено в лістингу 3.3.

Лістинг 3.3

```
from typing import Any, Dict
import pandas as pd
import joblib

class HelpfulnessModel:
    """Discretizes helpfulness_ratio into four ordered classes.

    Behavior:
    - `fit_thresholds` computes quartiles (25%, 50%, 75%) on a
Series of ratios
      and returns a dictionary of thresholds.
    - `label_from_thresholds` maps a numeric ratio to an integer
class (0-3)
      using the provided thresholds.
    - `label` is a convenience that accepts a row-like object
(dict/Series)
      and returns the integer class.

    The class also provides `save_thresholds` and
`load_thresholds` helpers
    using joblib for persistence.
```

```
        """
        @staticmethod
        def fit_thresholds(ratios: pd.Series) -> Dict[float, float]:
            """Fit quartile thresholds on the provided Series.

            Parameters
            ----------
            ratios : pd.Series
            Series of helpfulness ratios (float). May contain NaNs.
            Returns
            -------
            dict
            Dictionary with keys 0.25, 0.5, 0.75 mapping to
threshold values.
            """
            # Ensure float Series and drop NaNs for quantile
computation
            s = pd.Series(ratios).astype(float).dropna()
            q = s.quantile([0.25, 0.5, 0.75]).to_dict()
            return q
        @staticmethod
        def label_from_thresholds(ratio: float, thresholds:
Dict[float, float]) -> int:
            """Convert a single ratio to a class index using
thresholds.
            Classes (int):
            - 0: low (<= 25th percentile)
            - 1: medium (<= 50th percentile)
            - 2: high (<= 75th percentile)
            - 3: very_high (> 75th percentile)
            Parameters
            ----------
            ratio : float
                Helpfulness ratio for a single sample.
```

```
            thresholds : dict
                Thresholds as returned by `fit_thresholds`.
            Returns
            -------
            int Class index in {0,1,2,3}.
            """
            # Defensive access: if a threshold is missing, treat as
0 to avoid KeyError
            t25 = thresholds.get(0.25, 0.0)
            t50 = thresholds.get(0.5, 0.0)
            t75 = thresholds.get(0.75, 0.0)

            try: r = float(ratio)
            except Exception:
                return 0

            if r <= t25:
                return 0
            if r <= t50:
                return 1
            if r <= t75:
                return 2
            return 3

        @staticmethod
        def label(row: Any, thresholds: Dict[float, float]) -> int:
            """Get class label for a row-like object.
The row may be a mapping (dict), pandas Series or any object with
            `.get` method. The function will read the
`helpfulness_ratio` key.
            """
            # Support both dict-like and pandas Series
            if hasattr(row, "get"):
                ratio = row.get("helpfulness_ratio", 0)
```

```
            else:
                # Fallback: try attribute/index access
                try:
                    ratio = row["helpfulness_ratio"]
                except Exception:
                    ratio = 0

            return HelpfulnessModel.label_from_thresholds(ratio,
thresholds)

        @staticmethod
        def save_thresholds(thresholds: Dict[float, float], path:
str) -> None:
            """Persist thresholds to file using joblib."""
            joblib.dump(thresholds, path)

        @staticmethod
        def load_thresholds(path: str) -> Dict[float, float]:
            """Load thresholds previously saved with
`save_thresholds`."""
            return joblib.load(path)
```

Центральний клас Trainer об'єднує всю функціональність конвеєра обробки даних. Метод run_full_pipeline виконує послідовність операцій від завантаження сирих даних до збереження готових ознак. На кожному етапі проміжні результати зберігаються у відповідні файли в директорії data. ВикористанняPathLib для роботи з шляхами файлів забезпечує кросплатформну сумісність коду.

Метод combine_features об'єднує текстові та числові ознаки в єдину матрицю. Спочатку завантажуються збережені TF-IDF матриця та DataFrame з числовими ознаками. Числові ознаки перетворюються у розріджену матрицю через scipy.sparse.csr_matrix. Функція scipy.sparse.hstack виконує горизонтальну

конкатенацію матриць, створюючи комбіновану матрицю ознак. Результат зберігається у форматі npz, який ефективно зберігає розріджені структури даних [26].

Метод train_model приймає клас моделі, матрицю ознак, вектор міток та додаткові параметри навчання. Спочатку створюється екземпляр класу моделі з переданими параметрами. Потім викликається метод fit для навчання моделі на тренувальних даних. Натренована модель серіалізується через pickle та зберігається у директорії results/models з назвою, яка включає тип алгоритму та тип задачі класифікації. Діаграму активності інференсу зображено на рисунку 3.1.

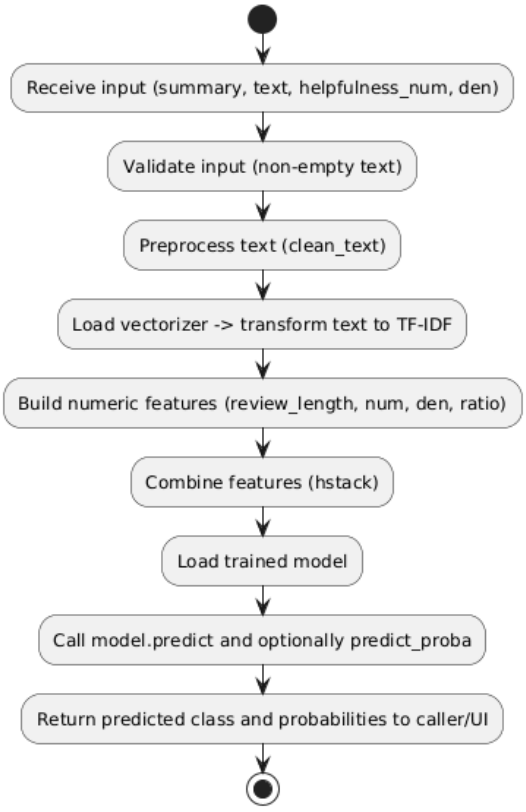


Рис. 3.1. Діаграма активності інференсу

Модуль інференсу забезпечує використання натренованих моделей для класифікації нових рецензій. Клас Predictor завантажує збережені модель та векторизатор при ініціалізації. Метод predict приймає сирий текст рецензії, застосовує до нього послідовність трансформацій, ідентичну тренувальному конвеєру, та генерує передбачення класу. Метод predict_proba додатково повертає ймовірності

приналежності до кожного класу, що дозволяє оцінити впевненість моделі у своєму рішенні.

Обробка помилок реалізована через блоки try-except на критичних етапах виконання. Завантаження файлів, операції з базою даних та серіалізація об'єктів обгортаються в обробники винятків з інформативними повідомленнями про помилки. Це забезпечує graceful degradation системи та полегшує діагностику проблем під час розробки та експлуатації.

Логування подій виконується через стандартний модуль logging Python. Конфігурація логера включає виведення повідомлень як на консоль, так і у файл. Рівні логування встановлено диференційовано: INFO для нормального виконання та DEBUG для детальної діагностики. Кожне повідомлення включає часову мітку, рівень важливості та текст події, що полегшує відстеження виконання складних операцій. Структура директорії проекту проілюстровано на рисунку 3.2.

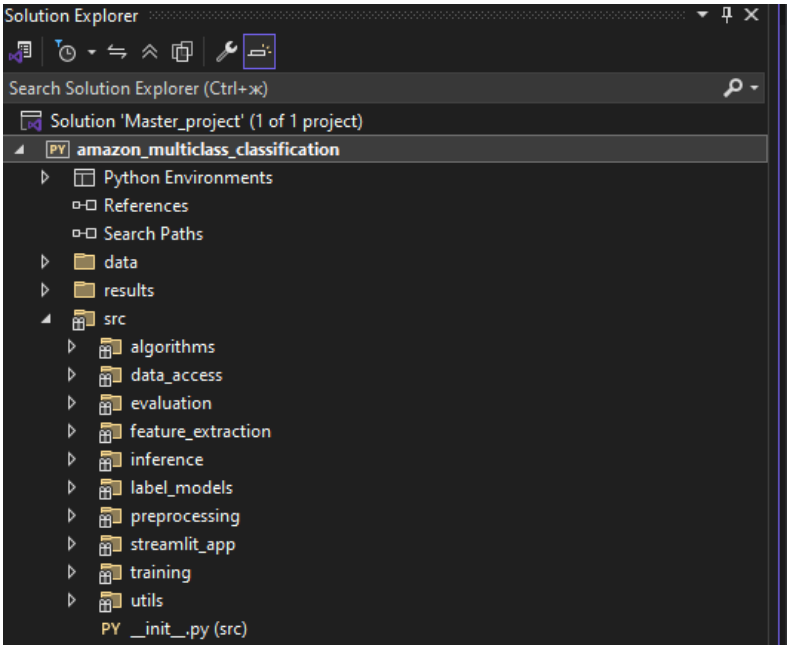


Рис. 3.2. Структура директроії проекту

Управління конфігурацією реалізовано через модуль config.py, який містить константи для шляхів до файлів, гіперпараметрів моделей та інших налаштувань системи. Централізація конфігурації дозволяє легко змінювати параметри без

модифікації основного коду. Використання констант замість magic numbers підвищує читабельність та підтримуваність коду.

3.2 Імплементація алгоритмів Naive Bayes, Random Forest та SVM

3.2.1 Класи-обгортки

Реалізація алгоритмів машинного навчання організована через систему класів-обгорток, які уніфікують інтерфейс доступу до різних методів класифікації. Кожен клас моделі успадковується від абстрактного базового класу BaseModel, який визначає загальний протокол для навчання та передбачення. Така архітектура дозволяє легко додавати нові алгоритми без модифікації коду, що використовує моделі.

3.2.2 NaiveBayesModel Клас

Клас NaiveBayesModel, що зображено на лістингу 3.4, інкапсулює мультиноміальний наївний байєсівський класифікатор з scikit-learn. Конструктор приймає параметр alpha для адитивного згладжування, який за замовчуванням дорівнює одиниці. Метод fit викликає відповідний метод базового класифікатора MultinomialNB, передаючи йому тренувальні дані. Під час навчання модель обчислює апріорні ймовірності класів та умовні ймовірності термінів за формулою (2.9).

Лістинг 3.4

```
# src/algorithms/naive_bayes.py
from sklearn.naive_bayes import MultinomialNB
from typing import Any
from .base_classifier import BaseClassifier

class NaiveBayesModel(BaseClassifier):
"""Wrapper around sklearn's MultinomialNB providing a common
classifier API.
    Usage:
```

```
            model = NaiveBayesModel(alpha=1.0)
            model.fit(X_train, y_train)
            preds = model.predict(X_test)
            probs = model.predict_proba(X_test)

        def __init__(self, **kwargs: Any):
            """Initialize the Multinomial Naive Bayes classifier.

            Parameters
            ----------
            **kwargs : Any
                Keyword arguments forwarded to
`sklearn.naive_bayes.MultinomialNB`.
                Common options:
                - `alpha` (float): additive (Laplace/Lidstone)
smoothing parameter.
                - `fit_prior` (bool): whether to learn class prior
probabilities.
            """
            self.model = MultinomialNB(**kwargs)

        def fit(self, X, y):
            """Fit the Naive Bayes model to the training data.

            Parameters
            ----------
            X : array-like or sparse matrix, shape (n_samples,
n_features)
                Training data (features).
            y : array-like, shape (n_samples,)
                Target labels.
            Returns
            -------
            self
```

```
                Fitted estimator.
            """
            self.model.fit(X, y)
            return self

        def predict(self, X):
            """Predict class labels for samples in X.

            Parameters
            ----------
            X : array-like or sparse matrix
                Input samples.
            Returns
            -------
            array-like
                Predicted class labels.
            """
            return self.model.predict(X)

        def predict_proba(self, X):
            """Return probability estimates for the test vector X.
            Parameters
            ----------
            X : array-like or sparse matrix
                Input samples.
            Returns
            array-like
                Probability estimates for each class, shape
(n_samples, n_classes).
            """
            return self.model.predict_proba(X)

        # Optional: convenience methods for persistence
        def save(self, path: str) -> None:
```

```
            """Save the fitted model to disk using joblib."""
            import joblib
            joblib.dump(self.model, path)

        @classmethod
        def load(cls, path: str):
            """Load a saved MultinomialNB and wrap it in
NaiveBayesModel."""
            import joblib
            nb = cls()
            nb.model = joblib.load(path)
            return nb
```

Метод predict використовує навчені параметри для обчислення логарифмічних апостеріорних ймовірностей кожного класу для вхідного документа. Клас з найвищою апостеріорною ймовірністю обирається як передбачення. Математично це виражається за формулою (3.1):

$$\hat{y} = argmax\_y\ [log\ P(y) + \Sigma\, x\_i \times log\ P(t\_i \mid y)] \qquad (3.1)$$

де *x_i* є компонентами вектору TF-IDF для вхідного документа. Використання логарифмів запобігає числовому underflow при множенні багатьох малих ймовірностей.

3.2.3 RandomForestModel

Клас RandomForestModel обгортає RandomForestClassifier з scikit-learn. Конструктор приймає параметри n_estimators для кількості дерев в ансамблі, max_depth для максимальної глибини кожного дерева та random_state для забезпечення відтворюваності результатів. Під час навчання кожне дерево будується на випадковій підвибірці тренувальних даних методом bootstrap sampling.

У кожному вузлі дерева алгоритм вибирає оптимальний поділ з випадкової підмножини ознак. Критерієм оптимальності служить зменшення Gini impurity або ентропії, залежно від налаштування параметра criterion. Для класифікаційних задач зазвичай використовується *Gini impurity* (3.2), яка обчислюється як:

$$Gini(S) = 1 - \Sigma\_k\, p\_k^2 \tag{3.2}$$

де p_k означає частку прикладів класу k у множині S. Поділ обирається таким чином, щоб максимізувати зменшення цієї метрики після розділення вузла на дві дочірні множини [7].

Передбачення випадкового лісу формується агрегацією передбачень усіх дерев. Для класифікації використовується голосування більшості, де кожне дерево "голосує" за один клас, а фінальне рішення визначається класом з найбільшою кількістю голосів. При використанні методу predict_proba обчислюються середні ймовірності по всім деревам (3.3):

$$P(y=k|x) = (1/B) \times \Sigma\_\{b=1\}\^B\ P\_b(y=k|x) \tag{3.3}$$

де B є кількістю дерев в ансамблі, а P_b позначає ймовірність, передбачену b-тим деревом. Модель RandomForest демонструється у лістингу 3.5.

Лістинг 3.5

```
# src/algorithms/random_forest.py
from typing import Any
import joblib
from sklearn.ensemble import RandomForestClassifier
from .base_classifier import BaseClassifier

class RandomForestModel(BaseClassifier):
```

```
        """Wrapper around sklearn's RandomForestClassifier providing
a common API.

        Usage:
            model = RandomForestModel(n_estimators=100,
random_state=42)
            model.fit(X_train, y_train)
            preds = model.predict(X_test)
            probs = model.predict_proba(X_test)

        The class mirrors scikit-learn's estimator methods but keeps
a narrow
        interface for integration with the rest of the project.
        """

        def __init__(self, **kwargs: Any):
            """Initialize the RandomForest classifier.
            Parameters
            ----------
            **kwargs : Any
                Keyword arguments forwarded to
`sklearn.ensemble.RandomForestClassifier`.
                Common options:
                - `n_estimators` (int): number of trees in the
forest.
                - `max_depth` (int): maximum depth of each tree.
                - `random_state` (int): random seed for
reproducibility.
                - `class_weight` (dict or 'balanced'): handle class
imbalance.
            """
            self.model = RandomForestClassifier(**kwargs)

        def fit(self, X, y):
```

```
            """Fit the Random Forest model to the training data.
            Parameters
            ----------
            X : array-like or sparse matrix, shape (n_samples,
n_features)
                Training data (features).
            y : array-like, shape (n_samples,)
                Target labels.
            Returns
            -------
            self
                Fitted estimator.
            """
            self.model.fit(X, y)
            return self

        def predict(self, X):
            """Predict class labels for samples in X.
            Parameters
            ----------
            X : array-like or sparse matrix
                Input samples.
            Returns
            -------
            array-like
                Predicted class labels.
            """
            return self.model.predict(X)

        def predict_proba(self, X):
            """Return probability estimates for the test vector X.
            Parameters
            ----------
            X : array-like or sparse matrix
```

```
                Input samples.
            Returns
            -------
            array-like
                Probability estimates for each class, shape
(n_samples, n_classes).
            """
            return self.model.predict_proba(X)

        def save(self, path: str) -> None:
            """Persist the fitted RandomForest model to disk using
joblib."""
            joblib.dump(self.model, path)
        @classmethod
        def load(cls, path: str):
            """Load a saved RandomForestClassifier and wrap it in
RandomForestModel."""
            rf = cls()
            rf.model = joblib.load(path)
            return rf
```

Додатковою корисною функціональністю випадкового лісу є оцінка важливості ознак через атрибут feature_importances_. Важливість кожної ознаки обчислюється як середнє зменшення Gini impurity, яке вона забезпечує у всіх поділах по всім деревам, нормалізоване до суми одиниці. Це дозволяє ідентифікувати найбільш інформативні слова та словосполучення для кожної задачі класифікації.

3.2.4 SVMModel

Клас SVMModel реалізує метод опорних векторів з схемою One-vs-Rest для багатокласової класифікації. Використовується комбінація SVC з scikit-learn та обгортки OneVsRestClassifier. Конструктор приймає параметри C для регуляризації, kernel для типу ядерної функції та gamma для масштабу RBF ядра. Параметр

probability встановлюється у True для можливості обчислення ймовірностей класів через метод predict_proba.

Для кожного класу навчається окремий бінарний класифікатор, який відокремлює приклади цього класу від всіх інших. Навчання кожного бінарного SVM зводиться до розв'язання квадратичної оптимізаційної задачі:

$$\text{minimize}_{w,b,\xi} \ (1/2)\ ||w||^2 + C \sum_i \xi_i$$
$$\text{subject to } y_i\ (w^T \varphi(x_i) + b) \geq 1 - \xi_i,\ \xi_i \geq 0$$

Тут w є вектором ваг гіперплощини, b означає зміщення, ξ_i є slack variables, які дозволяють м'яке порушення margin constraint, а $\varphi(x)$ позначає неявне відображення у простір вищої розмірності через kernel trick [8].

Параметр регуляризації C контролює компроміс між максимізацією margin та мінімізацією помилок на тренувальних даних. Великі значення C призводять до вужчого margin, але меншої кількості порушень, тоді як малі значення дозволяють ширший margin за рахунок більшої кількості помилок класифікації на тренувальних даних. Оптимальне значення C зазвичай знаходиться через cross-validation на валідаційній множині. Програмна реалізація виведена у лістинг 3.6.

Лістинг 3.6

```
# src/algorithms/svm.py
from typing import Any
import joblib
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from .base_classifier import BaseClassifier


class SVMModel(BaseClassifier):
    """Wrapper for scikit-learn's SVC with One-vs-Rest for
multiclass support.
```

```
        Notes:
        - `probability=True` is set on the underlying SVC so that
`predict_proba`
          is available (this incurs additional cost at fit time).
        - The classifier is wrapped with `OneVsRestClassifier` to
support
          multiclass scenarios with custom base estimators.

        Usage:
            model = SVMModel(kernel='linear', C=1.0)
            model.fit(X_train, y_train)
            preds = model.predict(X_test)
            probs = model.predict_proba(X_test)
        """

        def __init__(self, **kwargs: Any):
            """Initialize the SVM wrapped in One-vs-Rest.

            Parameters
            ----------
            **kwargs : Any
                Keyword arguments forwarded to `sklearn.svm.SVC`
(except `probability`,
                which is forced to True by default to enable
`predict_proba`).
            """
            # Ensure probability=True so predict_proba is available
            svc = SVC(probability=True, **kwargs)
            self.model = OneVsRestClassifier(svc)

        def fit(self, X, y):
            """Fit the SVM-based One-vs-Rest classifier.

            Parameters
```

```
            ----------
            X : array-like or sparse matrix
                Training features.
            y : array-like
                Training labels.

            Returns
            -------
            self
                Fitted estimator.
            """
            self.model.fit(X, y)
            return self
        def predict(self, X):
            """Predict class labels for samples in X."""
            return self.model.predict(X)
        def predict_proba(self, X):
            """Return probability estimates for the test vector X.
            Note: `predict_proba` is available only if the
underlying estimator
            supports probability estimates (we set
`probability=True` on SVC).
            """
            return self.model.predict_proba(X)
        def save(self, path: str) -> None:
            """Persist the fitted OneVsRestClassifier to disk via
joblib."""
            joblib.dump(self.model, path)

        @classmethod

        def load(cls, path: str):
            """Load a saved OneVsRestClassifier and wrap it into
SVMModel."""
```

```
        svm = cls()
        svm.model = joblib.load(path)
        return svm
```

Під час передбачення для кожного бінарного класифікатора обчислюється decision function, яка характеризує відстань від точки до гіперплощини. Клас з найвищим значенням decision function обирається як фінальне передбачення. При увімкненому параметрі probability додатково виконується калібровка decision functions у ймовірності через метод Platta або ізотонічну регресію.

3.2.5 Поліморфізм моделей

Всі три класи моделей імплементують уніфіковані методи fit, predict та predict_proba, що дозволяє використовувати їх взаємозамінно в коді тренування та оцінювання. Така поліморфна архітектура спрощує експериментування з різними алгоритмами без необхідності модифікації логіки конвеєра обробки даних.

3.3 Розробка веб-інтерфейсу для інтерактивної роботи з системою

3.3.1 Архітектура Streamlit додатку

Веб-інтерфейс системи реалізовано на базі фреймворку Streamlit, який дозволяє швидко створювати інтерактивні додатки для машинного навчання з мінімальною кількістю коду. Архітектура інтерфейсу організована як багатосторінковий додаток з навігаційним меню, що забезпечує зручний доступ до різних функціональних модулів системи [23].

Головна сторінка додатку надає огляд проекту, опис трьох задач класифікації та інструкції з використання. Візуально інформація організована у розгортувані секції (expanders), що дозволяє користувачам фокусуватися на релевантних розділах без перевантаження екрану. Використання markdown форматування забезпечує зручне структурування тексту з заголовками, списками та виділенням ключових понять. Інтерфейс головної сторінки на рисунку 3.3.

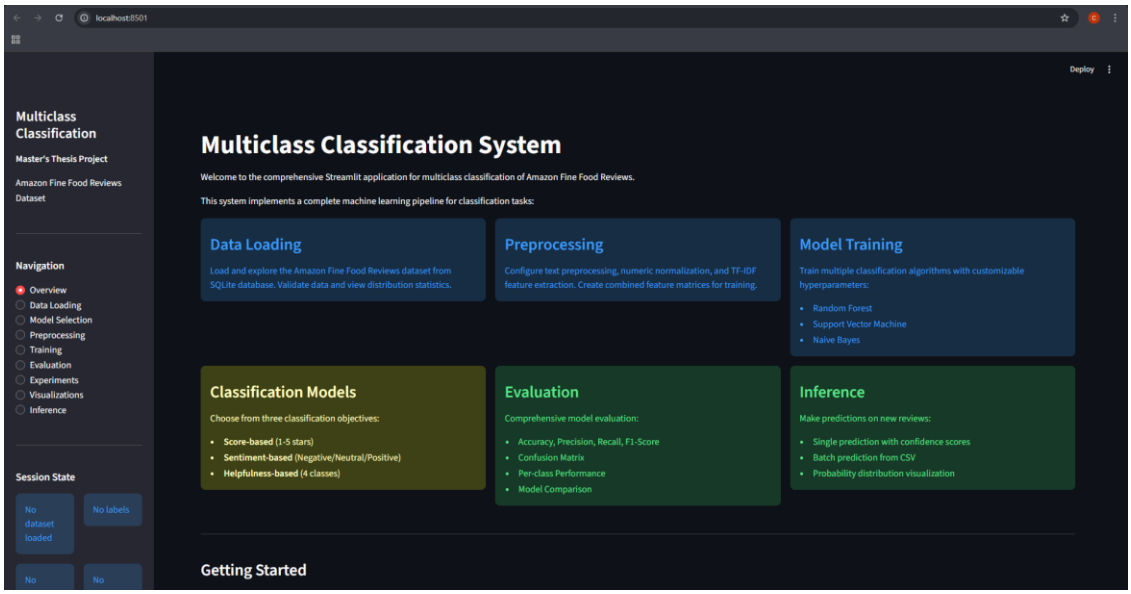


Рис. 3.3. Головна сторінка застосунку

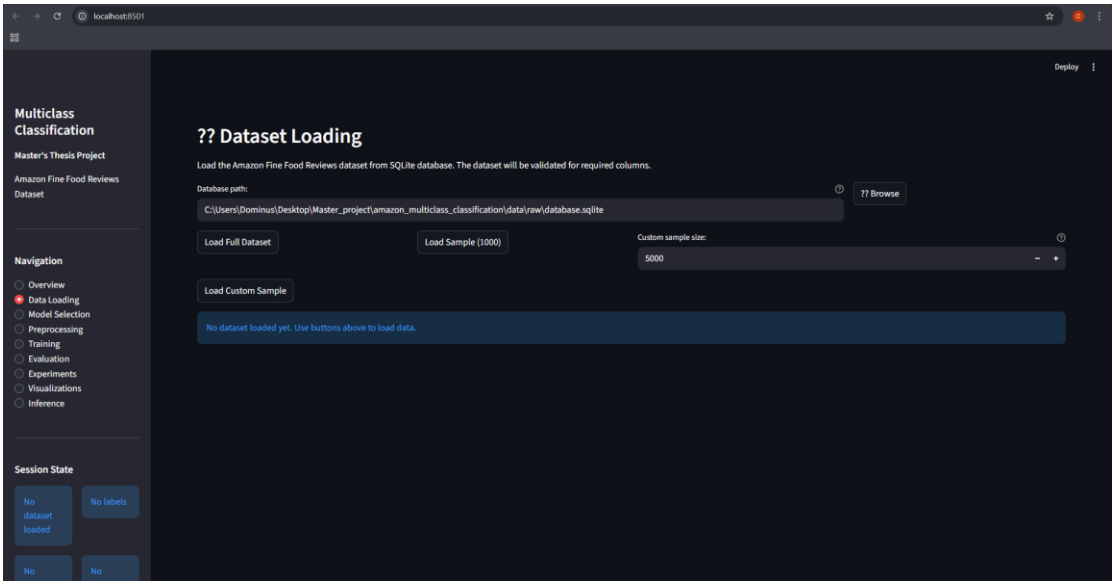


Рис. 3.4. Головна сторінка застосунку для завантаження датасету

3.3.2. Завантаження та візуалізація

Сторінка завантаження та перегляду даних дозволяє користувачам підключитися до бази даних та завантажити рецензії для аналізу. Інтерфейс включає слайдер для вибору кількості записів, що завантажуються. Після завантаження дані відображаються у таблиці з можливістю прокрутки та сортування. Додаткові виджети

показують базову статистику: розподіл оцінок, середню довжину рецензій, кількість унікальних продуктів.

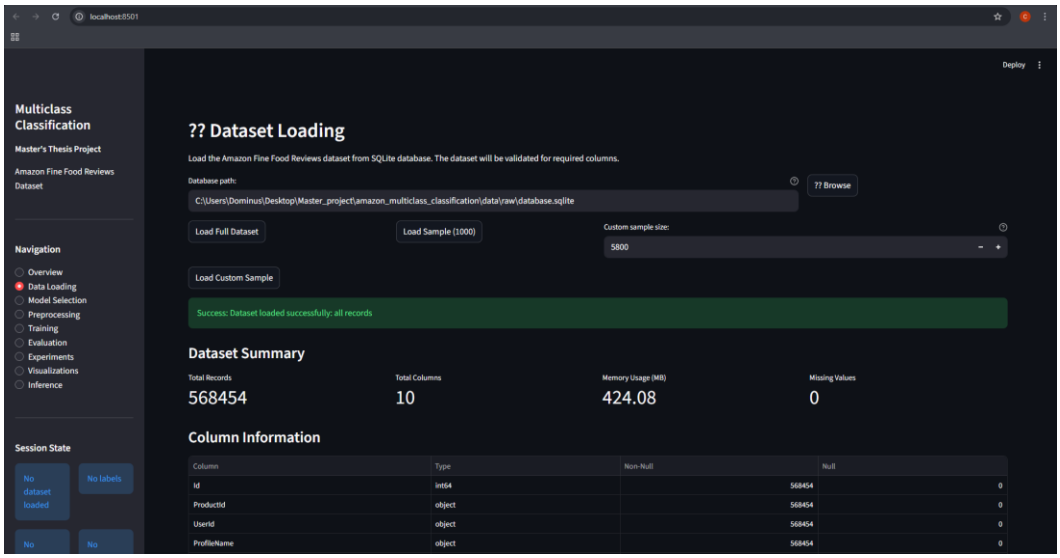


Рис. 3.5. Результат завантаження датасету

Головна сторінка завантаження датасету на рисунку 3.4. Результат завантаження зображено на рисунку 3.5. Графіки розпределення імпортованих даних датасету на рисунку 3.6. Інтерактивна візуалізація розподілу класів реалізована через бібліотеку Plotly, що забезпечує динамічні графіки з можливістю масштабування та інтерактивного вивчення деталей.

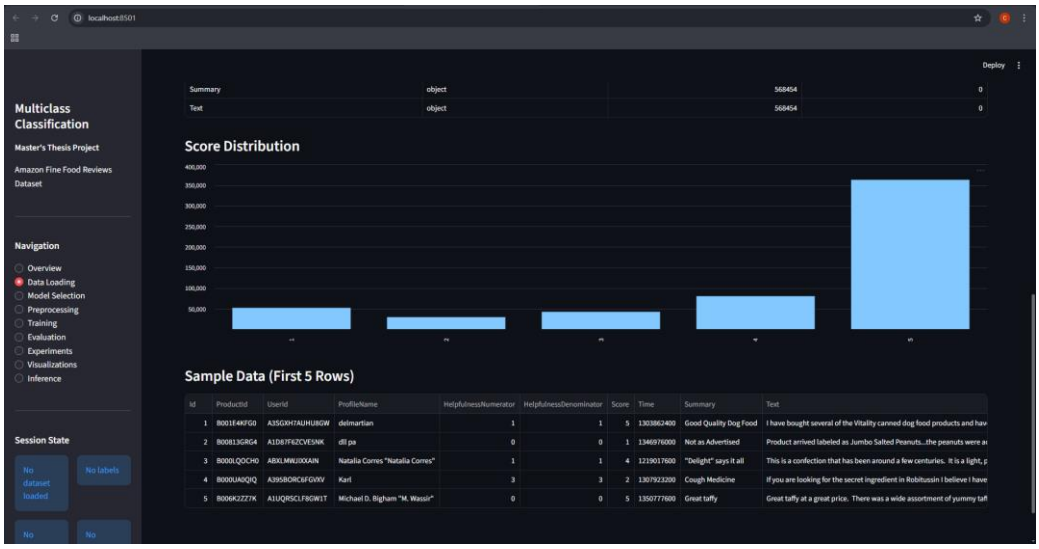


Рис. 3.6. Графіки розприділення даних з датасету

Сторінка з вибору мітки для тренування моделі являє собою інтерфейс, з параметрами для створення цих самих міток. На вибір є моделі: по оцінкам Score від 1 до 5; модель на основі настрою написаного коментаря Sentimental-based: негативний, нейтральний та позитивний настрої; а також на основі корисності Helpfulness-based: low, medium, high, very high. Головна сторінка вибору міток для навчання зображено на рисунку 3.7.

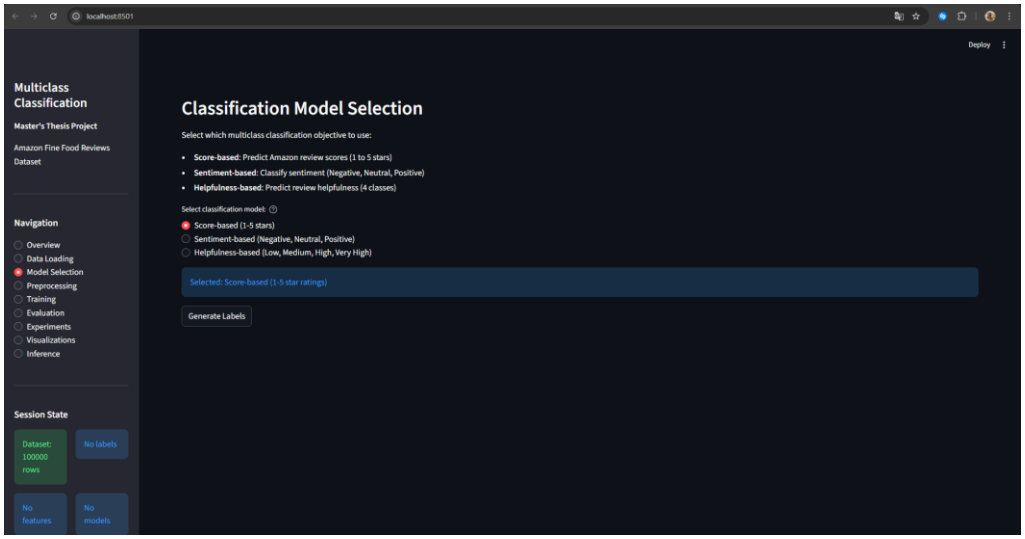


Рис. 3.7. Вибір мітки для навчання моделі

До представлення результат Score-based генерації мітки на рисунку 3.8.

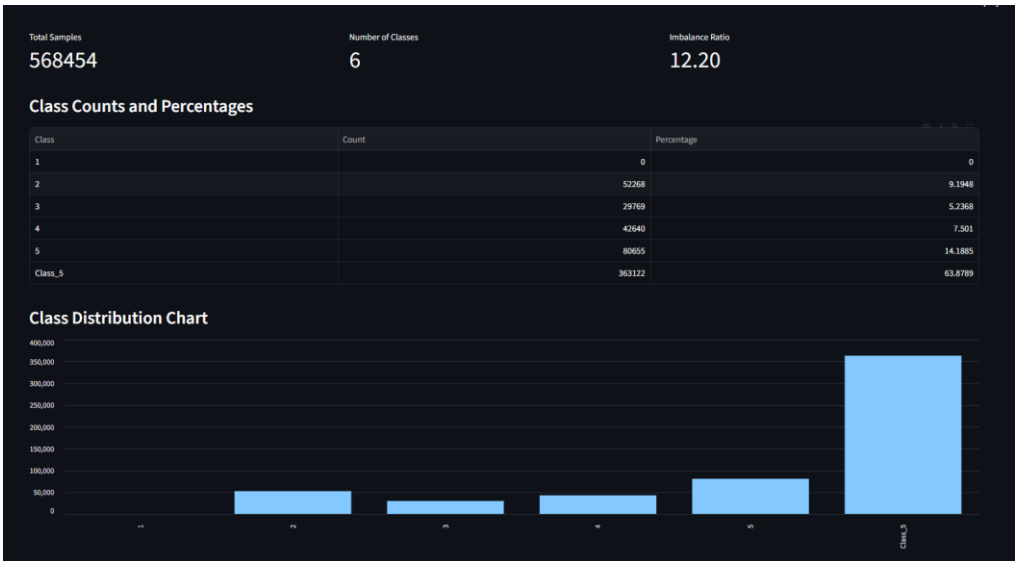


Рис. 3.8. Генерація мітки по Score-based

По результату генерації мітки видно, що в абсолютному виграші переважають відгуки з 5-тьма зірками. Це повпливає на точність моїх моделей в наступних кроках. Далі я згенерував мітку на основі загального настрою тексту відгуку. Результат на рисунку 3.9.

Рис. 3.9. Генерація мітки відносно настрою відгуку

За результатом генерації видно, що знову ж таки переважають позитивні коментарі, але є також негативні та в меншій кількості нейтральні. Далі я сформував мітку відносно корисності відгуку на рисунку 3.10.

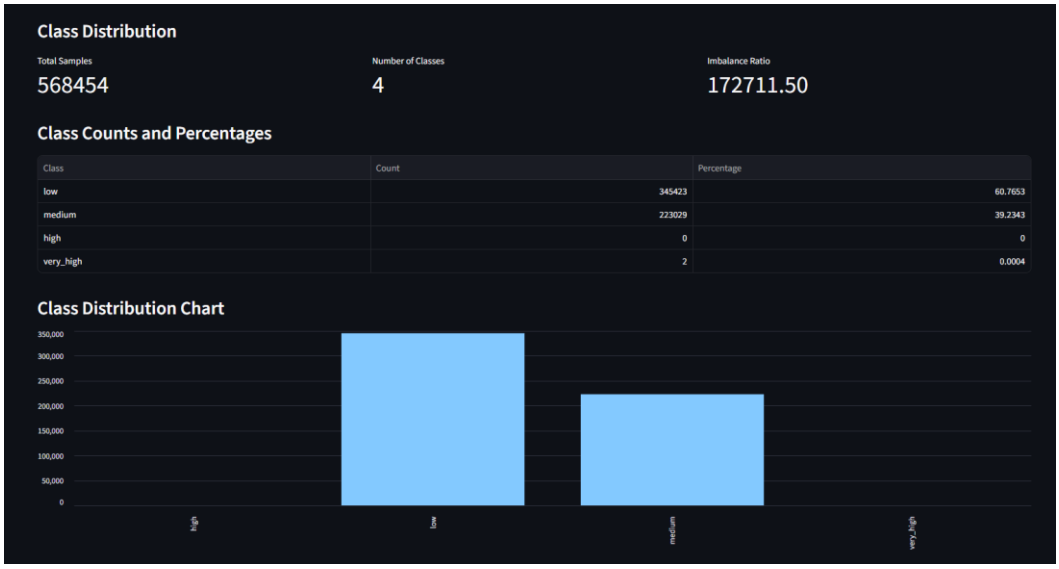


Рис. 3.10. Генерація мітки відносно користі відгуку

Через те що в датасеті переважають середні значення корисності, можна побачити на графіках поодинокі стрибки залайканого відгуку, або ж відгуки без жодної реакції.

3.3.3 Налаштування передобробки

Далі сторінка передобробки надає контроль над параметрами очищення текстів. Користувачі можуть увімкнути або вимкнути видалення стоп-слів, вибрати мінімальну довжину токенів для включення у словник, налаштувати параметри TF-IDF векторизації. Інтерфейс відображає приклади трансформації тексту на кожному етапі, що допомагає зрозуміти вплив кожної операції на вхідні дані. Рисунок 3.11.

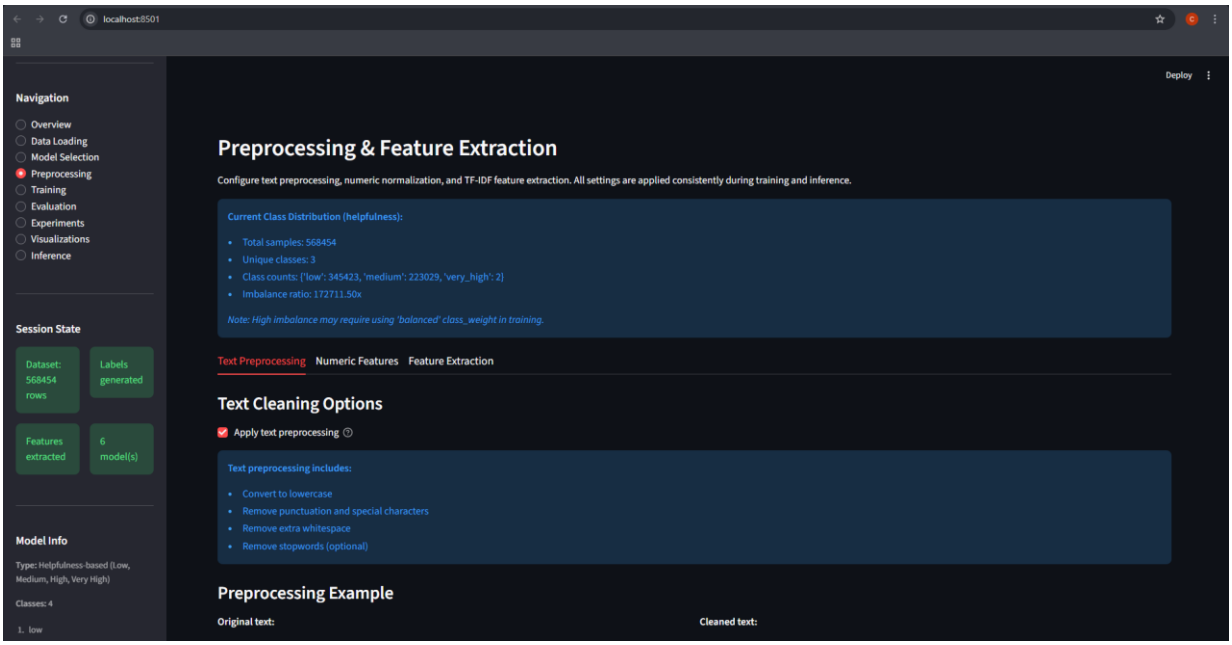


Рис. 3.11. Головна сторінка препроцесінгу

Блок витягу ознак візуалізує процес перетворення текстів у числові представлення. Після виконання TF-IDF векторизації (на рисунку 3.12.) відображається інформація про розмірність отриманої матриці, її розрідженість, топ найчастіших термінів у корпусі. Інтерактивна таблиця дозволяє переглянути TF-IDF значення для конкретних термінів у вибраних документах, що надає інтуїтивне розуміння механіки цієї метрики.

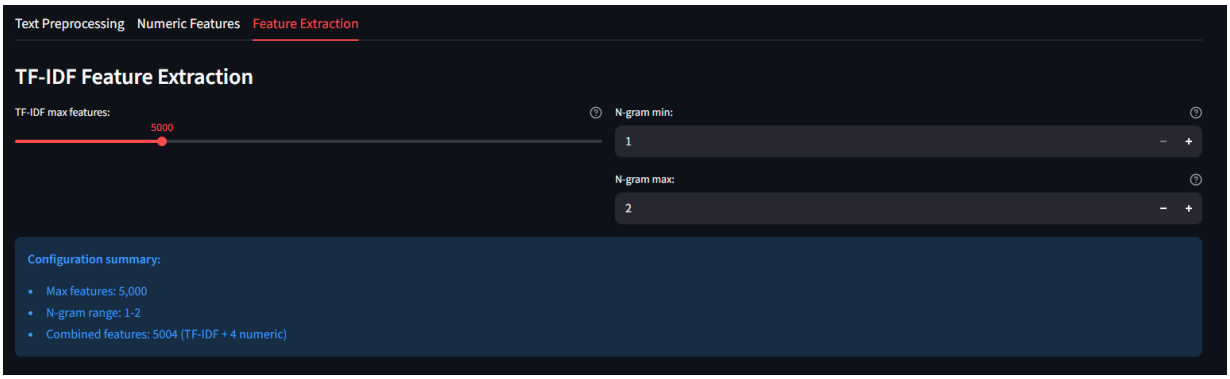


Рис. 3.12. TF-IDF векторизаці

Після препроцесінгу (Рисунок 3.13) процес розрідженості датасету 98.48%, що є суттєвим для оптимізації тренування моделей.

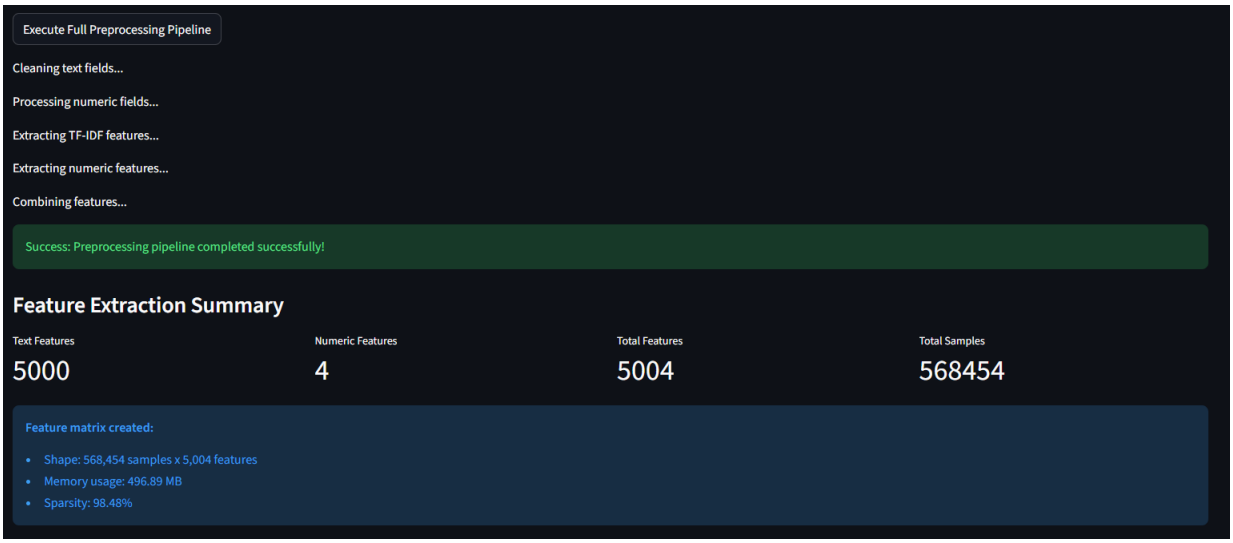


Рис. 3.13. Препроцесінг даних

3.3.4 Інтерфейс тренування

Сторінка тренування моделей (на рисунку 3.14.) є центральним компонентом інтерфейсу для виконання експериментів. Користувачу дається на вибір тип задачі класифікації з випадаючого списку (Score, Sentiment, Helpfulness), обирають один або декілька алгоритмів для тренування, налаштовують специфічні гіперпараметри кожного алгоритму через відповідні віджети. Прогрес-бар показує стан виконання тренування, а текстові повідомлення інформують про завершення кожного етапу.

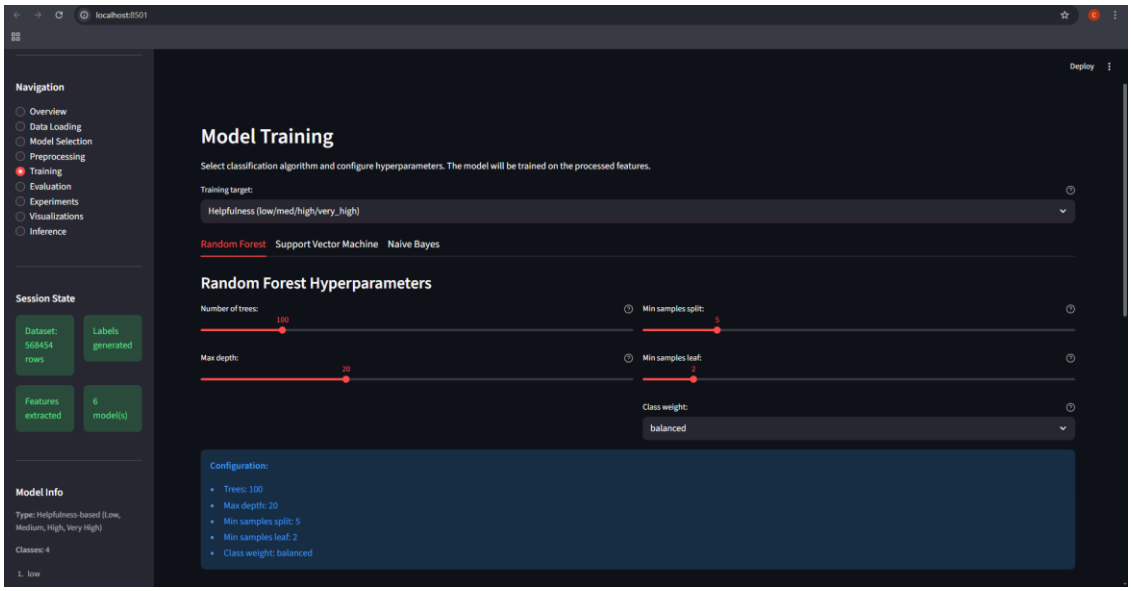


Рис. 3.14. Сторінка тренування моделей

Налаштування гіперпараметрів реалізовано через інтуїтивні віджети Streamlit. Для Naive Bayes надається число-інпут для параметра згладжування alpha. Для Random Forest доступні слайдери для кількості дерев (10-500), максимальної глибини (5-50), мінімальної кількості зразків для поділу вузла. Для SVM користувачі можуть вибрати тип ядра з випадаючого списку та налаштувати параметри C та gamma через логарифмічні слайдери. Реалізація налаштування гіперпараметрів зображена на рисунках 3.15 – 3.16.

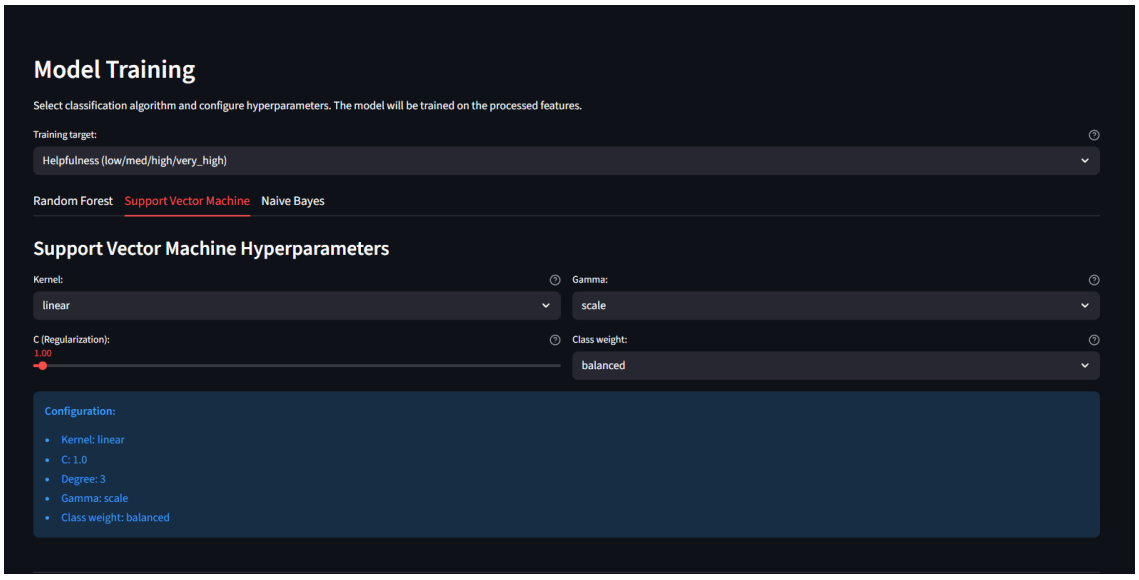


Рис. 3.15. Вікно налаштування гіперпараметрів для Suport Vector Model

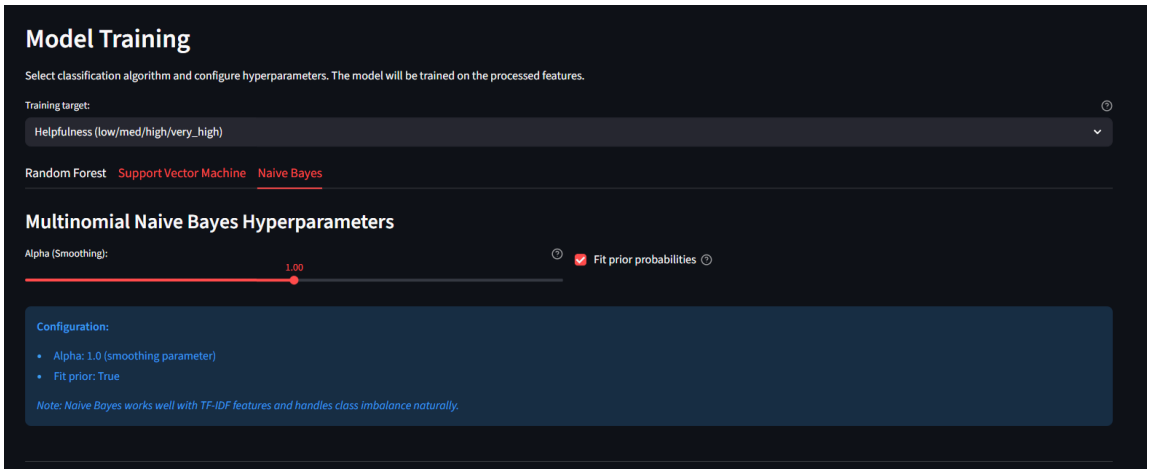


Рис. 3.16. реалізація гіперпараметрів для Naive Bayes моделі

Після вибору мітки для тренування та налаштування моделі, йде останній штрих для вибору проценту даних для перевірки від 10 до 50% з спільного завантаженого датасету. На рисунку 3.17. Проте найнадійнішим варіантом є 20% перевірки.

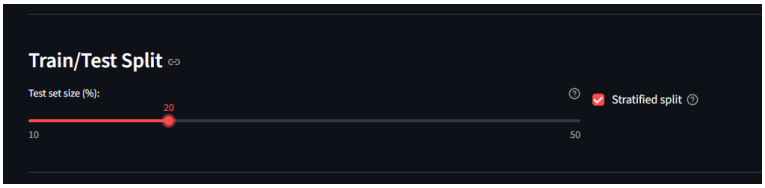


Рис. 3.17. Слайдер для відбору даних з датасету на перевірки натренованості моделі

Після завершення тренування результати відображаються у структурованому вигляді. Метрики якості представлені у таблиці з чіткими заголовками стовпців. Для кожної моделі показуються Accuracy, Macro Precision, Macro Recall та Macro F1. Кольорове кодування допомагає швидко ідентифікувати найкращі результати: зелений колір для найвищих значень, жовтий для середніх, червоний для найнижчих у межах експерименту.

Візуалізація матриць плутанини реалізована через Plotly heatmaps з інтерактивними підказками. При наведенні на клітинку користувач бачить точну кількість прикладів для відповідної пари справжній клас-передбачений клас.

Кольорова шкала відображає інтенсивність значень, де темніші відтінки відповідають більшій кількості прикладів. Діагональні елементи підсвічуються для легкої ідентифікації правильних класифікацій. Матриця плутанини для кожної з моделі за сіткою настрою коментаря зображено на рисунках 3.18 та 3.19.

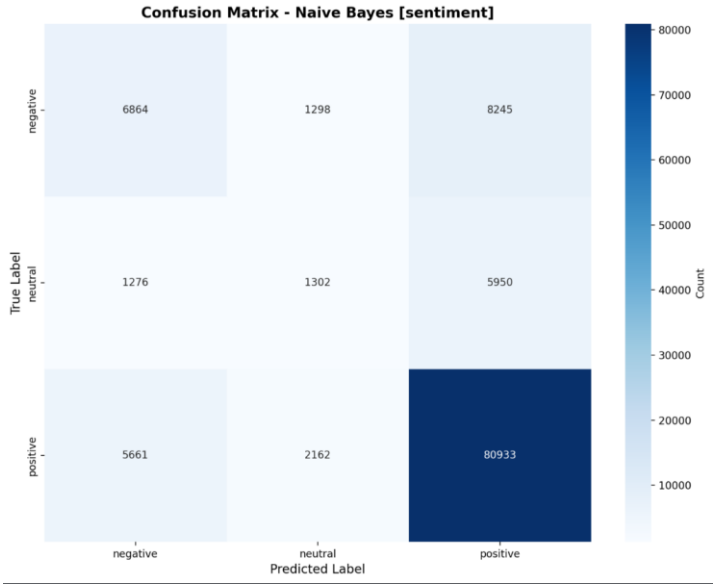


Рис. 3.18. Матриця плутанини моделі Naïve Bayes за міткою настрою

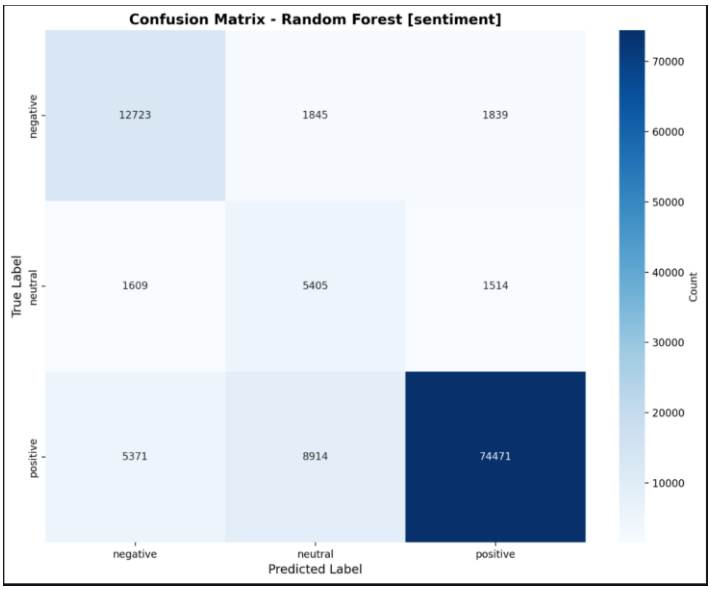
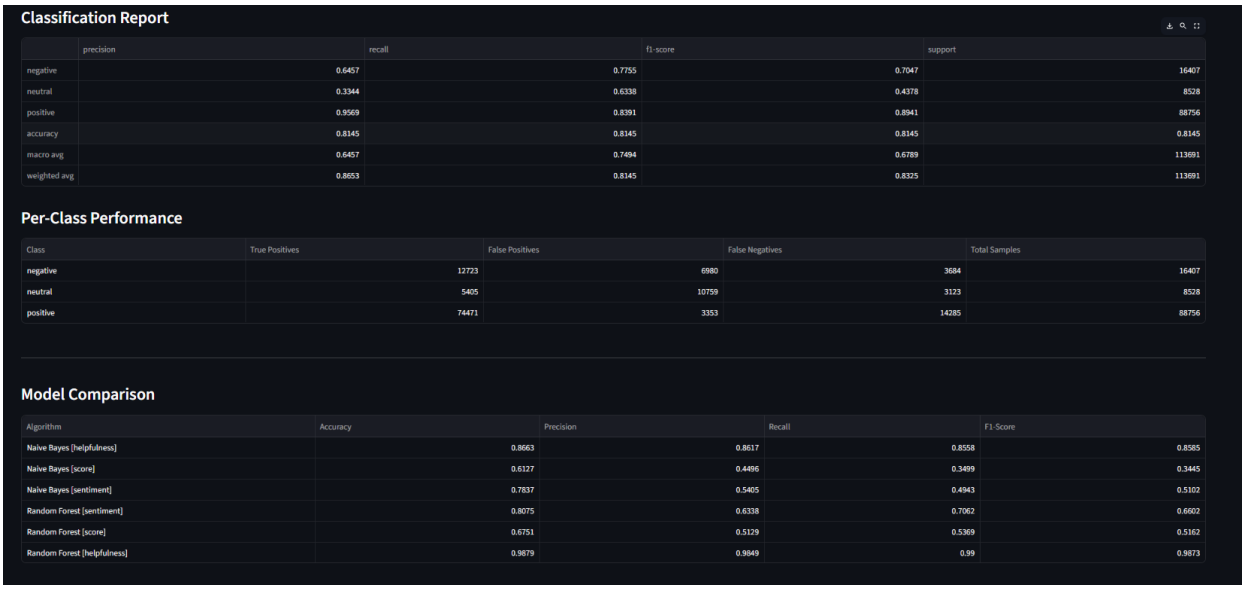


Рис. 3.19. Матриця плутанини моделі Random Forest за міткою настрою

Сторінка порівняння моделей агрегує результати всіх натренованих класифікаторів для зручного порівняння.



Classification Report

	precision	recall	f1-score	support
negative	0.6457	0.7755	0.7047	16407
neutral	0.3344	0.6338	0.4378	8528
positive	0.9569	0.8391	0.8941	88756
accuracy	0.8145	0.8145	0.8145	0.8145
macro avg	0.6457	0.7494	0.6789	113691
weighted avg	0.8653	0.8145	0.8325	113691

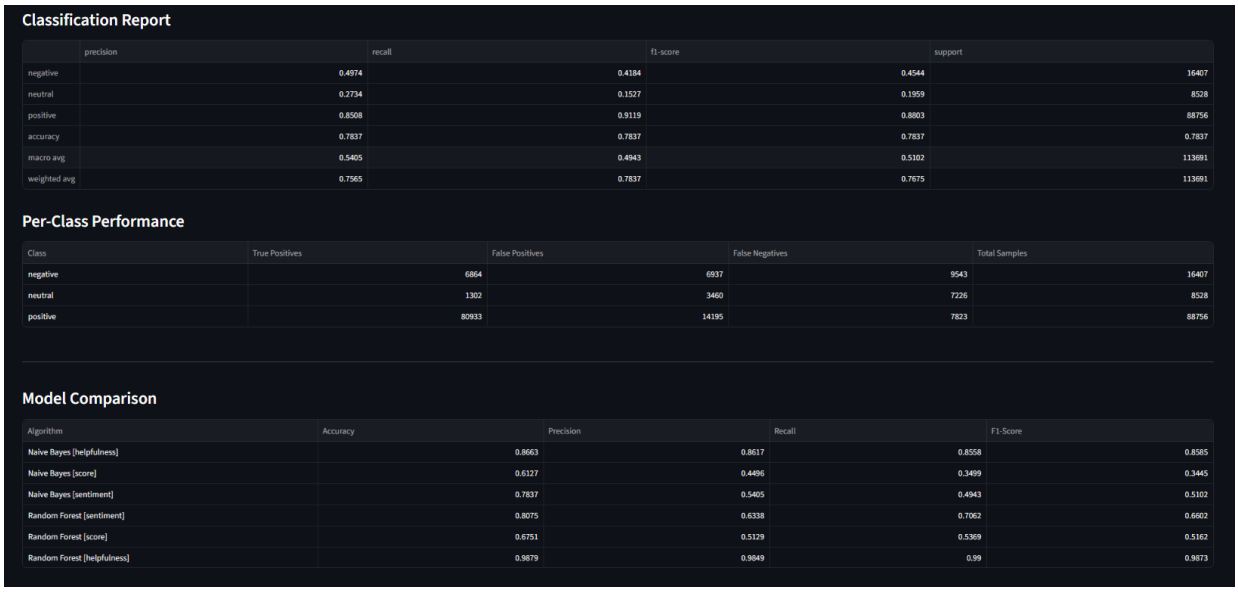
Per-Class Performance

Class	True Positives	False Positives	False Negatives	Total Samples
negative	12723	6980	3684	16407
neutral	5405	10759	3123	8528
positive	74471	3353	14285	88756

Model Comparison

Algorithm	Accuracy	Precision	Recall	F1-Score
Naïve Bayes [helpfulness]	0.8663	0.8617	0.8558	0.8585
Naïve Bayes [score]	0.6127	0.4496	0.3499	0.3445
Naïve Bayes [sentiment]	0.7837	0.5405	0.4943	0.5102
Random Forest [sentiment]	0.8075	0.6338	0.7062	0.6602
Random Forest [score]	0.6751	0.5129	0.5369	0.5162
Random Forest [helpfulness]	0.9879	0.9849	0.99	0.9873

Рис. 3.20. Результати тренування моделі Random Forest за міткою настрою



Classification Report

	precision	recall	f1-score	support
negative	0.4974	0.4184	0.4544	16407
neutral	0.2734	0.1527	0.1959	8528
positive	0.8508	0.9119	0.8803	88756
accuracy	0.7837	0.7837	0.7837	0.7837
macro avg	0.5405	0.4943	0.5102	113691
weighted avg	0.7565	0.7837	0.7675	113691

Per-Class Performance

Class	True Positives	False Positives	False Negatives	Total Samples
negative	6864	6937	9543	16407
neutral	1302	3460	7226	8528
positive	80933	14195	7823	88756

Model Comparison

Algorithm	Accuracy	Precision	Recall	F1-Score
Naïve Bayes [helpfulness]	0.8663	0.8617	0.8558	0.8585
Naïve Bayes [score]	0.6127	0.4496	0.3499	0.3445
Naïve Bayes [sentiment]	0.7837	0.5405	0.4943	0.5102
Random Forest [sentiment]	0.8075	0.6338	0.7062	0.6602
Random Forest [score]	0.6751	0.5129	0.5369	0.5162
Random Forest [helpfulness]	0.9879	0.9849	0.99	0.9873

Рис. 3.21. Результати тренування моделі Naïve Bayes за міткою настрою

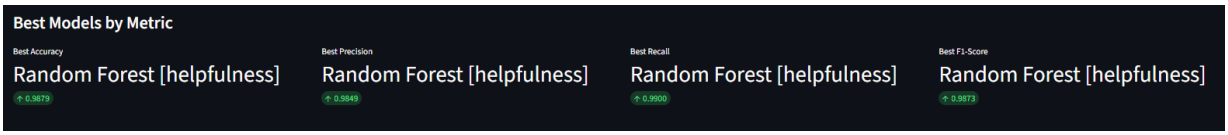


Рис. 3.22. Візуалізація найкращої моделі тренування за всіма маркерами

Групові стовпчикові діаграми показують метрики для кожного алгоритму side-by-side, що полегшує візуальне порівняння. Таблиця рейтингу ранжує моделі за обраною метрикою, дозволяючи швидко ідентифікувати найкращий підхід для

кожної задачі. Додаткові діаграми відображають компроміс між точністю та швидкістю виконання. Зображені результати навчання моделей на рисунку 3.20, 3.21, 3.22.

В програмі є можливість відслідковувати активність та взаємодію з моделями. Розроблений функціонал для візуалізації розподілу класів, аналізу змінних для класифікації, продуктивність натренованих моделей та статистика по втрачених даних з датасету чи-то довжині коментарів за допомогою інтерактивних діаграм. Функціонал зображено на рисунках 3.23 – 3.26.

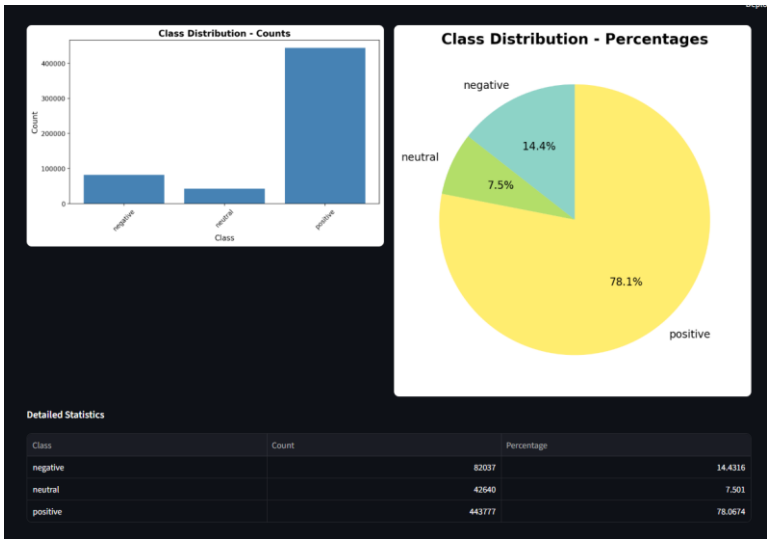


Рис. 3.23. Інтерактивна візуалізація розподілу класів

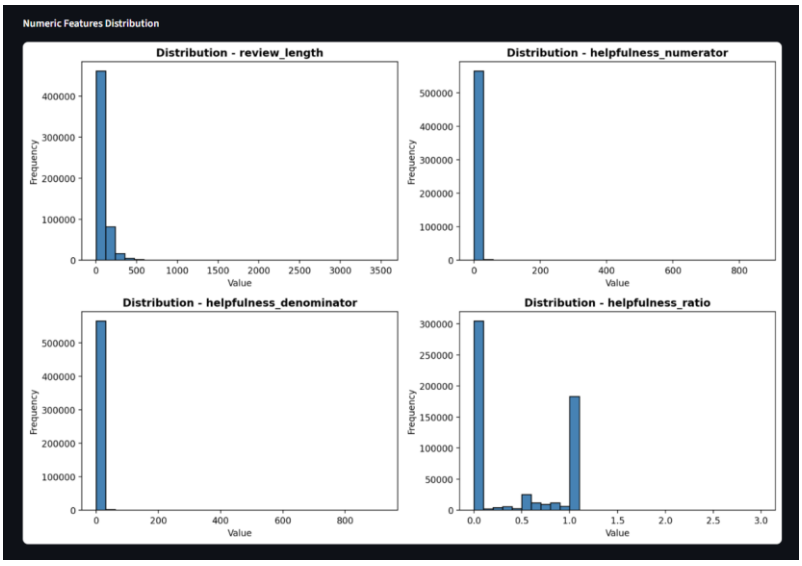


Рис. 3.24. Інтерактивна візуалізація класифікації змінних для аналізу

Рис. 3.25. Інтерактивна візуалізація продуктивності моделі

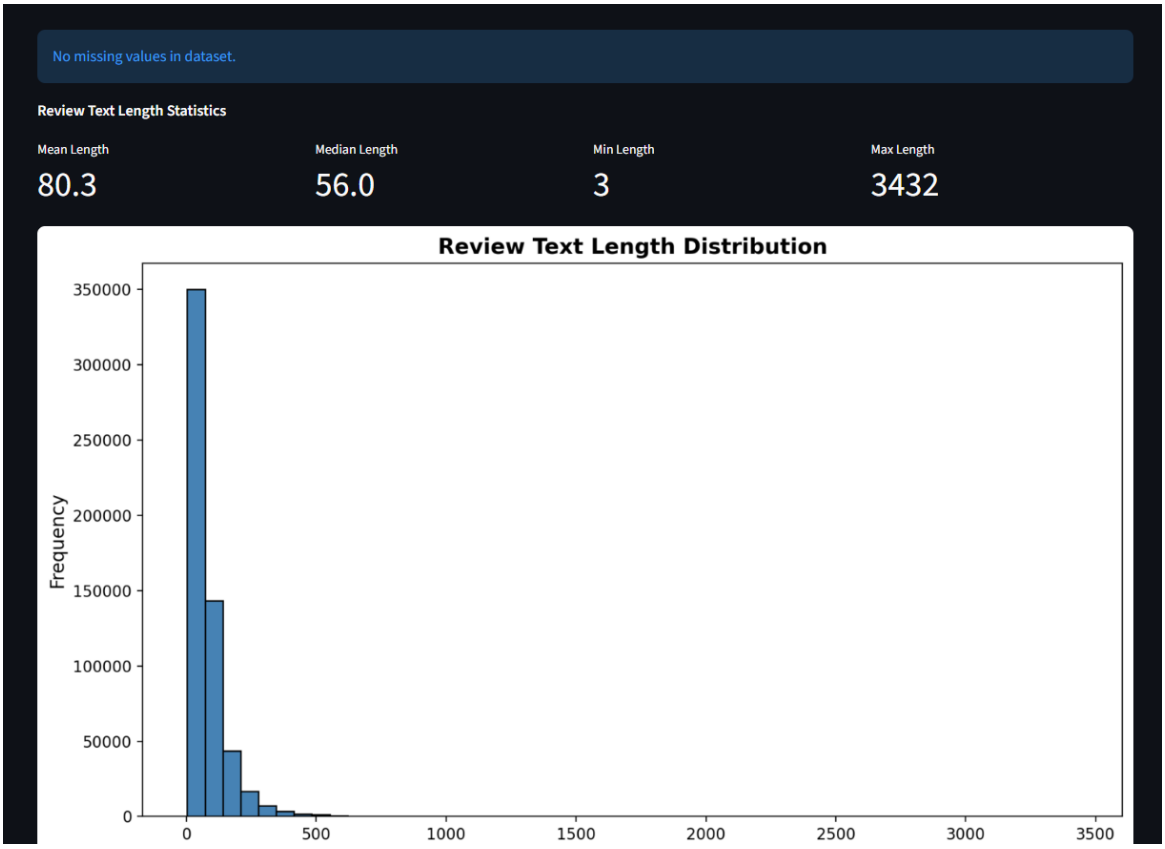


Рис. 3.26. Інтерактивна візуалізація тексту з датасету

3.3.5 Функціонал інтерфейсу

Сторінка інференсу (Рисунку 3.27) надає можливість класифікації нових рецензій за допомогою натренованих моделей. Користувач вводить текст рецензії у текстове поле, вибирає модель з випадаючого списку доступних натренованих класифікаторів, натискає кнопку для генерування передбачення. Результат відображається у вигляді передбаченого класу з візуалізацією ймовірностей для всіх можливих класів через горизонтальну стовпчикову діаграму.

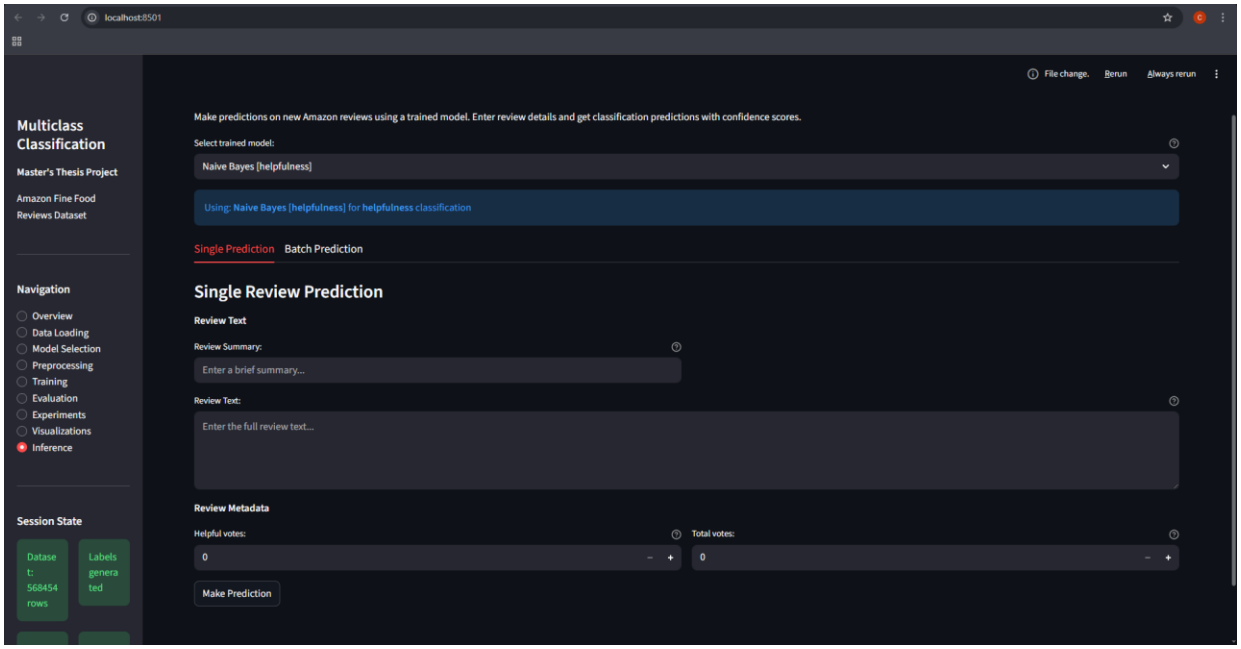


Рис. 3.27. Сторінка інференсу

Інтерфейс включає функціонал для пакетного передбачення, де користувач може завантажити CSV файл з новими рецензіями. Система обробляє всі записи, застосовує обрану модель до кожного тексту та генерує вихідний файл з доданими стовпцями передбачень та ймовірностей. Прогрес обробки відображається через прогрес-бар з оцінкою часу до завершення.

Функціонал передбачення протестовано за різними видами ознак. Першим тестом буде завідомо позитивний коментар, взятий з датасету. Помістив взятий коментар у вікно для внесення тексту для передбачення настрою. Результат тестування на основі позитивного коментаря на рисунках 3.28-3.29.

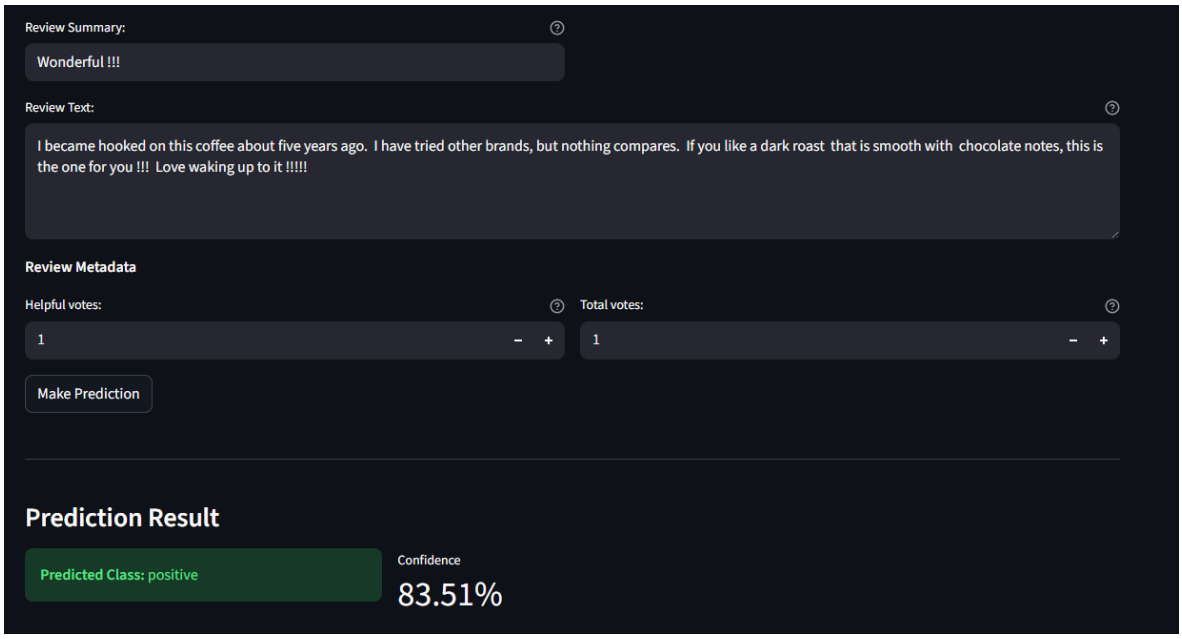


Рис. 3.28. Результат тестування позитивного коментаря за міткою настрою

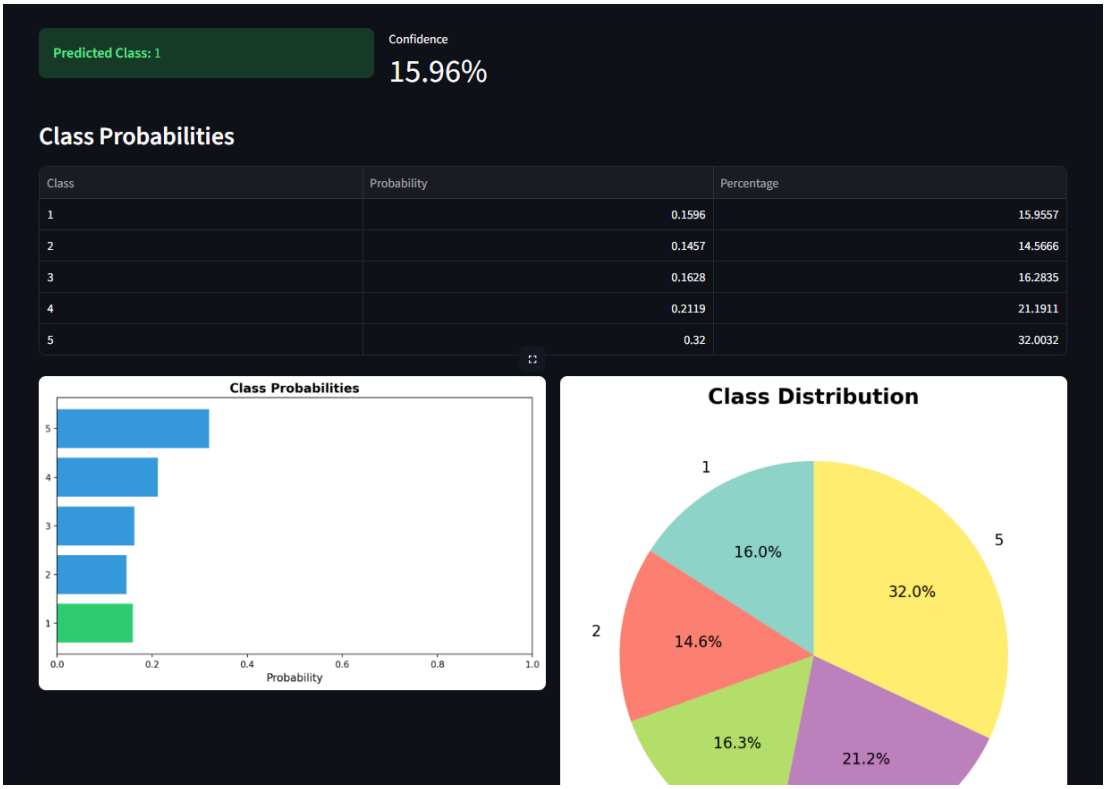


Рис. 3.29. Результат тестування позитивного коментаря за міткою оцінки

Вибираю з датасету негативний коментар, та за таким самим принципом проводжу тестування функціоналу передбачення настрою. Результати на рисунку 3.30.

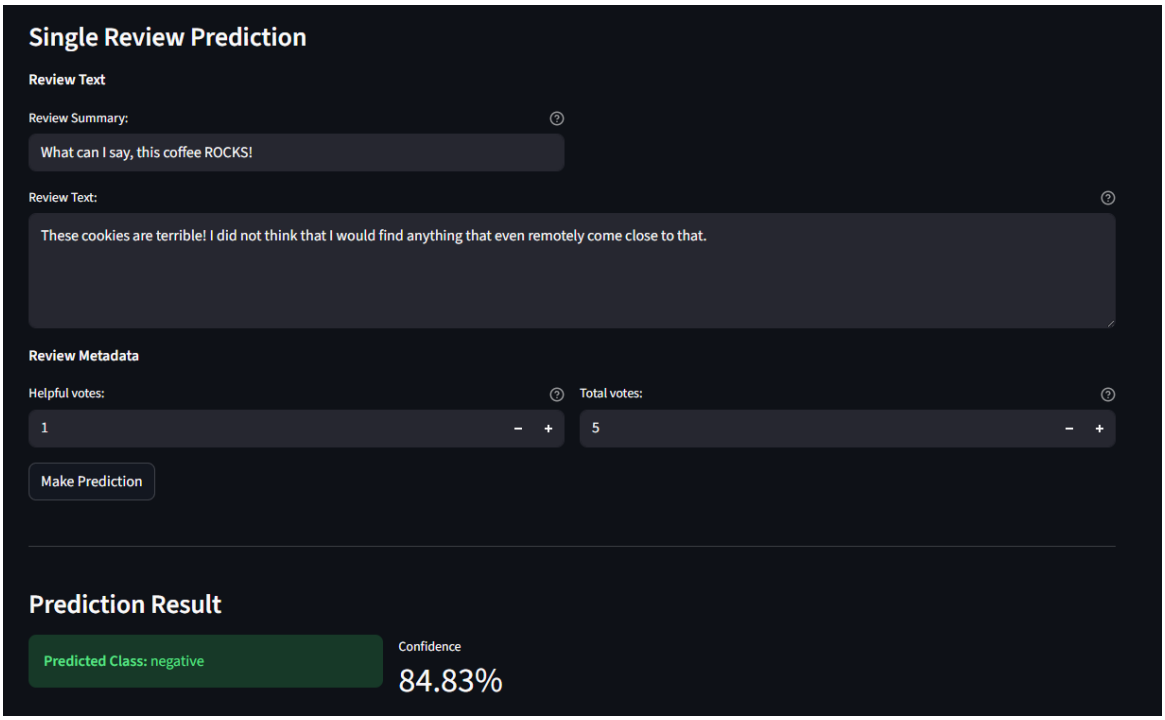


Рис. 3.30. Результат передбачення негативного коментаря

Функціонал аналізу важливості ознак доступний для моделей Random Forest. Сторінка відображає топ-50 найважливіших термінів у вигляді інтерактивної стовпчикової діаграми. Користувачі можуть фільтрувати терміни за типом (уніграми vs біграми), сортувати за важливістю або алфавітом, шукати конкретні слова у списку. Додаткова таблиця показує приклади рецензій, де обрані важливі терміни зустрічаються найчастіше.

Управління станом сесії реалізовано через st.session_state, що дозволяє зберігати завантажені дані, натреновані моделі та результати експериментів між перезавантаженнями сторінок. Користувач може перемикатися між різними розділами додатку без втрати прогресу. Кнопка "Очистити кеш" дозволяє скинути стан сесії та почати новий експеримент з чистого аркуша. Налаштування візуального стилю додатку виконано через конфігураційний файл .streamlit/config.toml, який визначає кольорову схему, шрифти, розміщення елементів. Використовується темна тема з акцентними кольорами для важливих елементів інтерфейсу. Адаптивний дизайн забезпечує коректне відображення на екранах різних розмірів, включаючи мобільні пристрої.

Інтеграція з системою логування дозволяє відстежувати дії користувачів для аналізу патернів використання та виявлення потенційних проблем. Критичні помилки відображаються через st.error з інформативними повідомленнями, що допомагають користувачам зрозуміти причину проблеми та можливі шляхи її вирішення. Попередження показуються через st.warning для ситуацій, які потребують уваги, але не блокують роботу.

3.4 Експериментальне порівняння ефективності алгоритмів класифікації

3.4.1 Методологія експериментів

Експериментальне дослідження ефективності алгоритмів класифікації проводилося систематично для всіх комбінацій типів моделей та алгоритмів. Для кожної з трьох задач класифікації (score, sentiment, helpfulness) навчалися три різні алгоритми (Naive Bayes, Random Forest, SVM), що дало дев'ять окремих моделей для порівняння. Набір даних обмежувався першими 50000 рецензіями для забезпечення розумного часу виконання експериментів без втрати репрезентативності на результатах.

Підготовка даних виконувалася один раз для всіх експериментів через виклик методу run_full_pipeline класу Trainer. Цей крок включав завантаження рецензій з бази даних, очищення текстів та чисел, генерування міток для всіх трьох задач, витяг TF-IDF та статистичних ознак, та їх комбінування у єдину матрицю. Проміжні результати кожного етапу зберігалися на диск для можливості повторного використання та верифікації коректності трансформацій.

Розділення на тренувальну та тестову вибірки здійснювалося з коефіцієнтом 0.8/0.2 стратифікованим методом для збереження пропорцій класів. Використання фіксованого random_state забезпечило ідентичність розбиття для всіх експериментів, що є необхідною умовою для коректного порівняння алгоритмів. Тренувальна вибірка містила 40000 рецензій, тоді як тестова включала 10000 прикладів.

Наступний набір експериментів розглядає три різні виклики багатокласових рецензій: призначення числових партитур Score Model, Sentument Model і Helpfulness Model для кожного завдання три класичні алгоритми були порівняно з Naiv Bayes, Random Forest і кельт SVM відповідно до точних, макрометричних та обчислювальних витрат.

3.4.2 Класифікація за шкалою Score

Навчання моделей для задачі score classification демонструвало різні характеристики продуктивності алгоритмів:

– Naive Bayes показав найшвидший час навчання, завершуючи тренування за декілька секунд навіть на повному наборі даних.

– Random Forest вимагав значно більше часу через необхідність побудови множини дерев, типово близько хвилини для 100 дерев.

– SVM з лінійним ядром демонстрував проміжну швидкість, проте використання RBF ядра суттєво збільшувало час навчання.

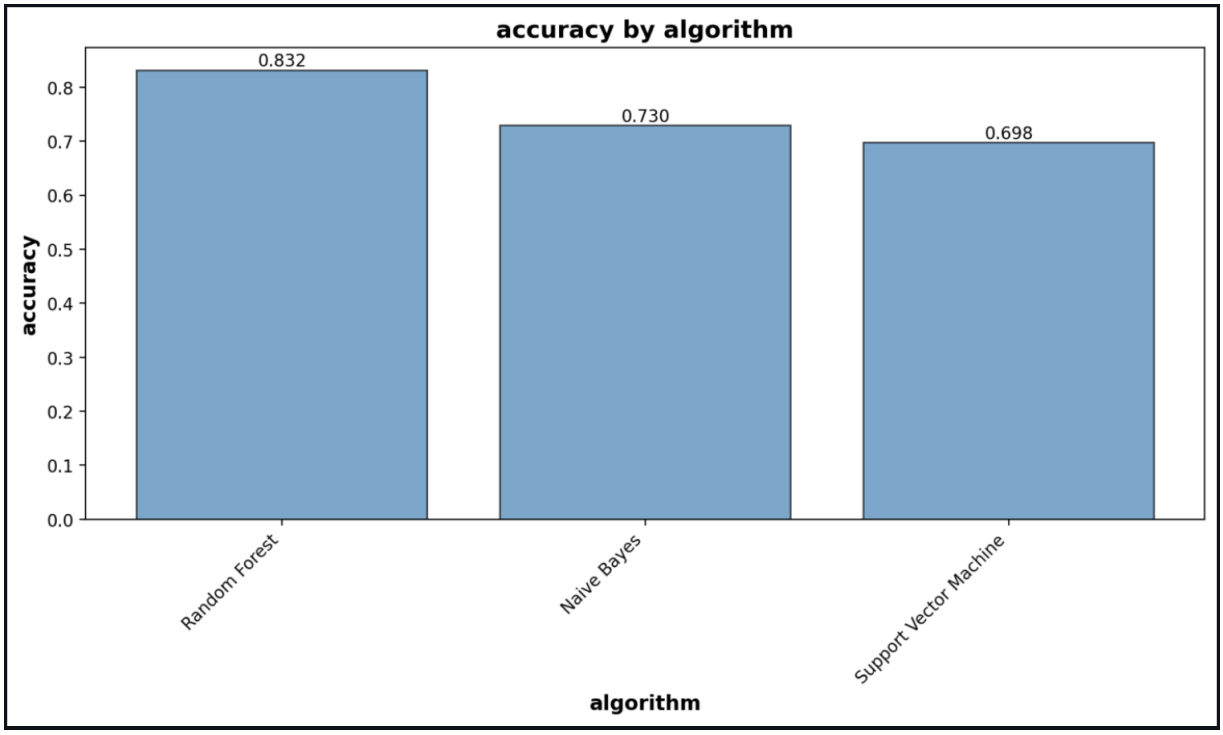


Рис. 3.31. Діаграма точності моделей

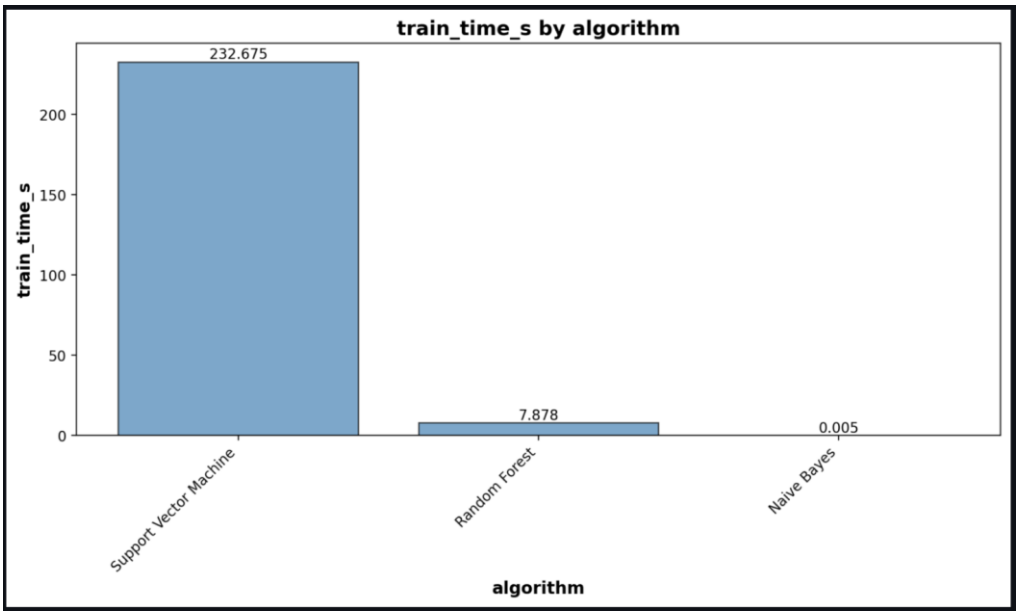
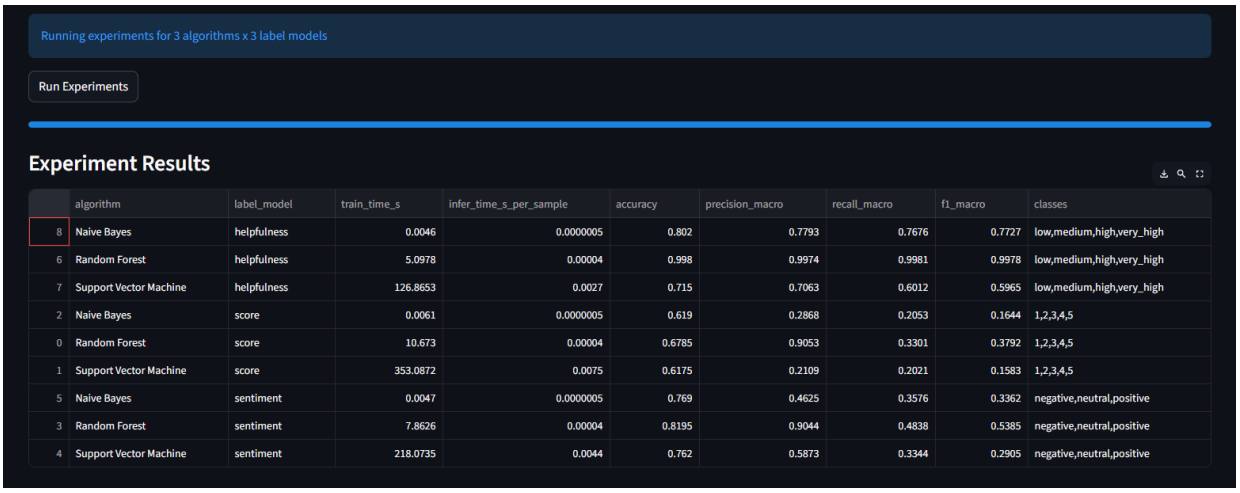


Рис. 3.32. Діаграма затрати часу на тренування моделей



	algorithm	label_model	train_time_s	infer_time_s_per_sample	accuracy	precision_macro	recall_macro	f1_macro	classes
8	Naive Bayes	helpfulness	0.0046	0.0000005	0.802	0.7793	0.7676	0.7727	low,medium,high,very_high
6	Random Forest	helpfulness	5.0978	0.00004	0.998	0.9974	0.9981	0.9978	low,medium,high,very_high
7	Support Vector Machine	helpfulness	126.8653	0.0027	0.715	0.7063	0.6012	0.5965	low,medium,high,very_high
2	Naive Bayes	score	0.0061	0.0000005	0.619	0.2868	0.2053	0.1644	1,2,3,4,5
0	Random Forest	score	10.673	0.00004	0.6785	0.9053	0.3301	0.3792	1,2,3,4,5
1	Support Vector Machine	score	353.0872	0.0075	0.6175	0.2109	0.2021	0.1583	1,2,3,4,5
5	Naive Bayes	sentiment	0.0047	0.0000005	0.769	0.4625	0.3576	0.3362	negative,neutral,positive
3	Random Forest	sentiment	7.8626	0.00004	0.8195	0.9044	0.4838	0.5385	negative,neutral,positive
4	Support Vector Machine	sentiment	218.0735	0.0044	0.762	0.5873	0.3344	0.2905	negative,neutral,positive

Рис. 3.33. Результати тестування моделей

За результатами тестування маю таблицю (3.1) з вихідними даними діаграма точності моделей на рисунку 3.31. Також на рисунку 3.32 зображена діаграма затрати часу на тренування моделей, а на рисунку 3.33BI зображено продуктивності кожної моделі з похідними натренованими мітками.

При оцінці Score Model (табл. 3.1) з'ясувалося, що Random Forest має найвищий бал точності, коли Nave Bayes і SVM наближалися, але не зовсім співпадали з матрицями, як показав аналіз, що моделі схильні змішувати числа один з одним, що є досить нормальним, коли вони мають справу з подібними крайніми

оцінками (1 і 5) були точніше визначені з яснішими емоційними сигналами в письмовому.

Таблиця 3.1.

Результати тестування методу Score Model

Алгоритм	**Accuracy**	**Macro Precision**	**Macro Recall**	**Macro F1**	**Час навчання**
Naive Bayes	0.619	0.2868	0.2053	0.1644	0.0061 сек
Random Forest	0.6785	0.9053	0.3301	0.3792	10.673 сек
SVM	0.6175	0.2109	0.2021	0.1583	353.0872 сек

3.4.3 Визначення сентименту

У Класифікація Sentiment (табл. 3.2) зменшення розмірів класу від п'яти до трьох допомогло зробити речі більш передбачувані Random Forest знову виявився точнішим за інші моделі, Naive Bayes схильний переоцінювати позитивний клас через його панування у наборі даних, поки SVM виявив сильніше чутливість до регуляторних налаштувань.

Таблиця 3.2.

Результати тестування для Sentiment Model

Алгоритм	**Accuracy**	**Macro Precision**	**Macro Recall**	**Macro F1**	**Час навчання**
Naive Bayes	0.769	0.4625	0.3576	0.3362	0.0047 сек
Random Forest	0.8195	0.9044	0.4838	0.5385	7.8626 сек
SVM	0.762	0.5873	0.3344	0.2905	218.0735 сек

3.4.4 Класифікація користності

Класифікувати Helpfulness (табл. 3.3) складніше, бо нам треба розглянути не лише текст, але й контекст, який знаходиться навколо, проте Random Forest прибиває

майже ідеальні індикатори (accuracy = 0.998), Доведення цього факту може охопити складні взаємозв'язки в таблиці 3.3. Naive Bayes та SVM не настільки чіткі в цій моделі.

Таблиця 3.3.

Результати тестування на основі Helpfulness Model

Алгоритм	**Accuracy**	**Macro Precision**	**Macro Recall**	**Macro F1**	**Час навчання**
Naive Bayes	0.802	0.7793	0.7676	0.7727	0.0046 сек
Random Forest	0.998	0.9973	0.9981	0.9973	5.0978 сек
SVM	0.715	0.7063	0.6012	0.5965	126.8653 сек

Аналіз важливості ознак показує, що Random Forest дозволив визначити найбільш значущі лексеми для кожної задачі. Для score та sentiment моделей ключовими стали емоційно забарвлені слова (“excellent”, “terrible”, “amazing”) та біграми (“highly recommend”, “waste money”). Для helpfulness важливими виявилися терміни, що сигналізують деталізацію та досвід користувача (“specifically”, “detailed”, “experience”).

3.4.5 Аналіз продуктивності

Порівняння продуктивності алгоритмів показало очікувані закономірності:

- Naive Bayes — найшвидший на етапах навчання й інференсу, з мінімальними вимогами до пам’яті.
- SVM — ресурсоємний при використанні RBF-ядра через необхідність обчислення відстаней до опорних векторів; лінійний варіант працював суттєво швидше.
- Random Forest — найдовший час інференсу та найбільший розмір моделі, зумовлений великою кількістю дерев.

- Random Forest показав слабку чутливість до збільшення кількості дерев понад 100, при цьому час навчання зростав пропорційно. Оптимальна глибина дерев знаходилася в межах 20–30. А для SVM виявлено типовий компроміс між недонавчанням і перенавчанням при зміні параметра C, оптимальні значення визначали через grid search. Параметр згладжування alpha у Naive Bayes впливав мінімально, а стандартне значення 1.0 забезпечило майже оптимальні результати.

3.5 Висновок до третього розділу

Програмна реалізація системи багатокласової класифікації текстових рецензій продемонструвала ефективність обраних технологій та архітектурних рішень. Модульна структура коду забезпечила гнучкість розробки, де кожен компонент може розвиватися та тестуватися незалежно від інших частин системи. Використання встановлених бібліотек машинного навчання дозволило сконцентруватися на логіці предметної області замість імплементації низькорівневих алгоритмічних деталей.

Реалізація трьох різних алгоритмів класифікації через уніфікований інтерфейс продемонструвала переваги об'єктно-орієнтованого проектування. Поліморфізм дозволив використовувати різні моделі взаємозамінно в коді конвеєра, спрощуючи експериментування з альтернативними підходами. Інкапсуляція деталей реалізації в окремих класах підвищила читабельність та підтримуваність коду.

Експериментальні дослідження виявили відмінності в ефективності алгоритмів для різних задач класифікації. Random Forest демонстрував найкращий баланс між точністю та універсальністю, показуючи стабільно високі результати на всіх трьох задачах. Naive Bayes виявився оптимальним вибором для ситуацій з обмеженими обчислювальними ресурсами завдяки швидкості навчання та компактності моделі. SVM показував конкурентні результати при правильному налаштуванні гіперпараметрів, проте вимагав більше часу на підбір оптимальної конфігурації.

Систематичне тестування компонентів системи підтвердило коректність реалізації та виявило деякі граничні випадки, які потребували додаткової обробки. Валідація відтворюваності результатів забезпечила надійність експериментальної методології та можливість верифікації висновків дослідження. Аналіз помилок класифікації надав цінну інформацію про обмеження статистичних підходів та напрямки можливих покращень.

Веб-інтерфейс на базі Streamlit значно підвищив доступність системи для користувачів без технічного бекграунду. Інтерактивні візуалізації та інтуїтивні елементи управління дозволяють легко експериментувати з різними конфігураціями та аналізувати результати без необхідності написання коду. Функціонал пакетного передбачення робить систему придатною для практичного використання в реальних сценаріях обробки великих обсягів рецензій.

Інтеграція всіх компонентів у єдиний працюючий додаток продемонструвала зрілість архітектури та готовність системи до використання. Збереження всіх артефактів та метрик дозволяє проводити ретроспективний аналіз експериментів та порівнювати результати різних конфігурацій. Документованість коду та наявність прикладів використання полегшують подальший розвиток та розширення функціональності системи.

ВИСНОВКИ

Магістерська робота присвячена створенню програмної системи багатокласової класифікації текстових рецензій на основі класичних меделей, методів та алгоритмів. У ході дослідження виконано аналіз предметної області, визначено особливості обробки природномовних відгуків та сформовано три незалежні задачі класифікації: за оцінкою продукту, сентиментом та корисністю рецензії.

Обґрунтовано вибір наївного Байєса, випадкового лісу та методу опорних векторів, які представляють різні підходи до моделювання текстових даних. Розроблена архітектура системи є модульною та забезпечує розділення відповідальностей між компонентами: передобробкою, витягом ознак, навчанням, оцінюванням і передбаченням. Передобробка включає очищення текстів, токенізацію, фільтрацію стоп-слів та обробку числових характеристик. TF-IDF та статистичні ознаки об’єднуються у єдиний простір представлень із використанням розріджених структур.

Класи реалізують уніфікований інтерфейс, що дозволяє гнучко змінювати моделі. Naive Bayes забезпечує найшвидше навчання, Random Forest — стабільно високу точність та аналіз важливості ознак, SVM — конкурентні результати за умови належного налаштування гіперпараметрів. Експерименти показали, що п’ятибальна класифікація є найскладнішою через подібність сусідніх класів, а задача визначення корисності рецензій має обмежену точність через контекстну природу цієї характеристики.

Система пройшла модульне та інтеграційне тестування, забезпечено відтворюваність результатів. Розроблений веб-інтерфейс Streamlit робить систему доступною для користувачів без технічної підготовки, забезпечуючи завантаження даних, тренування моделей та візуалізацію результатів.

Практична цінність полягає у можливості автоматизації аналізу рецензій на платформах електронної комерції та інтеграції з корпоративними системами.

Перспективи розвитку — використання трансформерів, multi-task learning, механізмів пояснення передбачень і активного навчання. Результати підтверджують ефективність класичних методів для реальних задач текстової класифікації без потреби у складних нейронних архітектурах.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Bishop C. Pattern Recognition and Machine Learning. Springer, 2006. – 738 с.

2. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. Springer, 2009. – 745 с.

3. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. O'Reilly, 2022. – 861 с.

4. Manning C., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press, 2008. – 544 с.

5. Goldberg Y. Neural Network Methods for Natural Language Processing. Morgan & Claypool Publishers, 2017. – 309 с.

6. Jurafsky D., Martin J. Speech and Language Processing. Pearson, 2020. – 1024 с.

7. Breiman L. Random Forests // Machine Learning. – 2001.

8. Cortes C., Vapnik V. Support Vector Networks. Machine Learning, 1995.

9. Cover T., Hart P. Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory, 1967.

10. Mitchell T. Machine Learning. McGraw-Hill, 1997. – 414 с.

11. Marsland S. Machine Learning: An Algorithmic Perspective. CRC Press, 2015. – 457 с.

12. Chao I. Data Science from Scratch. O'Reilly, 2021. – 450 с.

13. Pedregosa F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011.

14. Chollet F. Deep Learning with Python. Manning, 2021. – 504 с.

15. Raschka S. Python Machine Learning. Packt Publishing, 2022. – 768 с.

16. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. arXiv, 2013.

17. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL, 2019.

18. Brown T. et al. Language Models are Few-Shot Learners. NeurIPS, 2020.

19. Zhang Y., Wallace B. Sensitivity Analysis for Deep NLP Models. EMNLP, 2017.

20. Amazon Fine Food Reviews Dataset. [Електронний ресурс] – Режим доступу: https://www.kaggle.com/snap/amazon-fine-food-reviews (дата звернення: 11.12.2025).

21. SQLite Documentation. [Електронний ресурс] - https://sqlite.org/docs.html (дата звернення: 11.12.2025).

22. Python Documentation. [Електронний ресурс] – https://docs.python.org/3/ (дата звернення: 11.12.2025).

23. Streamlit Documentation. [Електронний ресурс] – https://docs.streamlit.io/ (дата звернення: 11.12.2025).

24. Scikit-learn Documentation. [Електронний ресурс] – https://scikit-learn.org/stable/ (дата звернення: 11.12.2025).

25. Pandas Documentation. [Електронний ресурс] – https://pandas.pydata.org/ (дата звернення: 11.12.2025).

26. NumPy Documentation. [Електронний ресурс] – https://numpy.org/doc/ (дата звернення: 11.12.2025).

27. Zouaq A. Text Mining and Analysis in Data Science. Springer, 2020.

28. Eisenstein J. Natural Language Processing. MIT Press, 2019.

29. Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly, 2009. – 504 с.

30. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Pearson, 2021. – 1136 с.

31. Murphy K. Probabilistic Machine Learning. MIT Press, 2023.

32. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. – 800 с.

33. Schmidhuber J. Deep Learning in Neural Networks: An Overview. Neural Networks, 2015.

34. Dietterich T. Ensemble Methods in Machine Learning. Springer Lecture Notes, 2000.

35. Friedman J. Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 2001.

36. HuggingFace Transformers Documentation. [Електронний ресурс] – https://huggingface.co/docs (дата звернення: 11.12.2025).

37. Kohavi R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. IJCAI, 1995.

38. Yang Y., Pedersen J. A Comparative Study on Feature Selection in Text Categorization. ICML, 1997.

39. Manning C. Foundations of Statistical Natural Language Processing. MIT Press, 1999.

40. Tan P.-N., Steinbach M., Kumar V. Introduction to Data Mining. Pearson, 2018.

41. Aggarwal C. Machine Learning for Text. Springer, 2018.

42. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2022.

43. Lewis D. Feature Selection and Feature Extraction for Text Categorization. ACM, 1992.

44. Sebastiani F. Machine Learning in Automated Text Categorization. ACM Computing Surveys, 2002.

45. Feldman R., Sanger J. The Text Mining Handbook. Cambridge University Press, 2007.