МАГІСТЕРСЬКА РОБОТА

МР.КІ – 28.00.00.000 ПЗ

Група $КІ_М$-24-2

Іващенко Владислав

2025

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій

Кафедра комп'ютерних систем і мереж

Іващенко Владислав Вікторович
(прізвище, ім'я, по батькові)

УДК 004.4

МАГІСТЕРСЬКА РОБОТА

Розробка комп'ютерної системи автоматичного виявлення спаму на основі методу опорних векторів (SVM)

Комп'ютерна інженерія
(назва освітньої програми)

123 – Комп'ютерна інженерія
(шифр і назва спеціальності)

/ В. В. Іващенко /
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник – Топалов Андрій Миколайович, к.т.н., доцент

Допущено до захисту
Завідувач кафедри

д-р.т.н., проф. /С.І. Мельничук/
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

доцент /В. Б. Кропивницька/
(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 рік

Івано-Франківський національний технічний університет нафти і газу

Факультет *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітній рівень *магістр*

Спеціальність *123 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ**:**

Зав. кафедрою КСМ

проф. С. І. Мельничук

“ 05 ” грудня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТОВІ

Іващенку Владиславу Вікторовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи *Розробка комп'ютерної системи автоматичного виявлення спаму на основі методу опорних векторів (SVM)*

Керівник проекту *доц., к.т.н. Топалов А. М.*

затверджені наказом вищого навчального закладу від « 05» *грудня 2025 року № 755/7.*

Термін здачі студентом закінченої роботи *10 грудня 2025 р*

3. Вихідні дані до проекту (роботи) *матеріали науково-дослідної практики*

4. Зміст розрахунково - пояснювальної записки (перелік питань, що їх належить розробити)

Огляд предметної області.

Проєктування системи виявлення спаму.

Реалізація та дослідження ефективності системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти по магістерській роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата
Нормоконтроль	**Мойсеєнко О. В.**	

7. Дата видачі завдання – *12 березня 2025*.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів магістерської роботи	Термін виконання етапів роботи	Примітка
1	*Аналіз літератури пов'язаних з обраною темою*	*12.03.25 – 18.06.25*	*Виконано*
2	*Огляд предметної області.*	*19.06.25 –8.08.25*	*Виконано*
3	*Проєктування системи виявлення спаму*	*9.08.25 – 12.09.25*	*Виконано*
4	*Реалізація та дослідження ефективності системи*	*13.09.25 –20.11.25*	*Виконано*
5	*Оформлення роботи*	*21.11.25 – 10.12.25*	*Виконано*

Студент-магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота містить 76 сторінок, 23 ілюстрації, 4 таблиці, список використаних джерел із 29 найменувань, 1 додаток.

Магістерська робота присвячена розв'язанню актуальної задачі розробки комп'ютерної системи автоматичного виявлення спаму на основі методу опорних векторів (SVM).

У роботі виконано системний аналіз предметної області антиспам-фільтрації, розглянуто типові канали й ознаки спаму та узагальнено сучасні підходи до протидії. На підставі порівняльних міркувань обґрунтовано вибір методу опорних векторів (SVM) як базового класифікатора для задачі бінарної текстової класифікації у високовимірному просторі ознак.

Розроблено програмне рішення на Python 3.9, у якому реалізовано повний конвеєр обробки: очищення тексту, нормалізація шаблонів (<URL>, <EMAIL>, <NUM>), токенізація зі стоп-фільтрацією і стемінгом, формування векторного представлення на основі TF–IDF (уніграми/біграми) та застосування лінійного SVM. Практичне використання продемонстровано у консольному режимі (пакетна перевірка) та через веб-інтерфейс для інтерактивного аналізу повідомлення.

Експериментальна оцінка на тестовому наборі SAPC+SMS показала accuracy 98.2%, precision 97.0%, recall 95.5% та F1 96.2%; у порівнянні з базовим наївним баєсовським класифікатором отримано вищу якість класифікації (NB: близько 94% точності та нижчий precision).

Ключові слова: спам, фільтрація, метод опорних векторів, SVM, машинне навчання, TF-IDF, класифікація текстів, інформаційна безпека, Python, веб-інтерфейс.

SUMMARY

The master's thesis comprises 76 pages, 23 figures, 4 tables, a list of references containing 29 entries, and 1 appendix.

The thesis is devoted to solving a relevant problem of developing a computer system for automatic spam detection based on the Support Vector Machine (SVM) method.

A systematic analysis of the anti-spam filtering domain is conducted, typical spam delivery channels and indicators are examined, and modern countermeasures are summarized. Based on comparative considerations, the Support Vector Machine (SVM) method is substantiated as the core classifier for binary text classification in a high-dimensional feature space.

A software solution was developed in Python 3.9 implementing a complete processing pipeline: text cleaning, pattern normalization (<URL>, <EMAIL>, <NUM>), tokenization with stop-word filtering and stemming, construction of a vector representation using TF–IDF (unigrams/bigrams), and application of a linear SVM. Practical usage is demonstrated in a console mode (batch checking) and via a web interface for interactive message analysis.

Experimental evaluation on the SAPC+SMS test set achieved 98.2% accuracy, 97.0% precision, 95.5% recall, and 96.2% F1; compared to a baseline Naive Bayes classifier, higher classification quality was obtained (NB: about 94% accuracy and lower precision).

Keywords: spam, filtering, support vector machine, SVM, machine learning, TF-IDF, text classification, information security, Python, web interface.

ЗМІСТ

ВСТУП

В епоху глобальної цифровізації електронна пошта залишається критично важливим інструментом комунікації для бізнесу, державних установ та приватних осіб. За статистичними даними, щодня у світі надсилається понад 300 мільярдів електронних листів, з яких значна частка припадає на спам — масову розсилку небажаної кореспонденції. Спам перестав бути лише дратівливим фактором, перетворившись на потужний інструмент кіберзлочинності, що використовується для фішингу, розповсюдження шкідливого програмного забезпечення та атак соціальної інженерії.

Актуальність теми. Методи машинного навчання (Machine Learning) демонструють високий потенціал у задачах обробки природної мови (NLP). Серед них метод опорних векторів (Support Vector Machine — SVM) вирізняється високою точністю при роботі з даними великої розмірності, якими є текстові документи. Розробка спеціалізованої комп'ютерної системи на основі SVM, здатної автоматично аналізувати семантику повідомлень та приймати рішення щодо їх класифікації, є актуальним науково-технічним завданням.

Об’єкт дослідження — процес автоматизованої обробки та класифікації текстових повідомлень електронної пошти.

Предмет дослідження — методи, моделі та алгоритми виявлення спаму на основі методу опорних векторів (SVM).

Мета і завдання роботи. Метою даної магістерської роботи є розробка та дослідження комп’ютерної системи автоматичного виявлення спаму з використанням метода опорних векторів. Для досягнення поставленої мети необхідно вирішити такі завдання:

− Проаналізувати сучасний стан проблеми спаму та методи його фільтрації, зокрема методи машинного навчання, що застосовуються для виявлення спам-повідомлень. Виконати огляд наукових публікацій і існуючих систем.

− Обґрунтувати вибір моделі SVM для задачі класифікації спаму, розробити методику побудови такої моделі. Здійснити збір та підготовку навчальних даних, обрати спосіб представлення текстів листів у числовому вигляді (ознак) для подачі на вхід SVM.

− Розробити архітектуру програмної системи, що реалізує класифікацію спам/не спам. Реалізувати прототип програми: навчити модель SVM на підготовленому наборі даних, інтегрувати її у програму з інтерфейсом для користувача.

− Провести експериментальне дослідження точності та ефективності розробленої системи. Виконати тестування на контрольному наборі електронних листів, оцінити показники якості класифікації (точність, повнота, F-міра), порівняти з альтернативними алгоритмами (наприклад, баєсовим класифікатором). На основі результатів зробити висновки щодо придатності метода SVM для задач фільтрації спаму та визначити напрямки подальшого вдосконалення.

Методи дослідження. У роботі використано методи системного аналізу (для дослідження предметної області), методи обробки природної мови (для нормалізації тексту), методи лінійної алгебри та оптимізації (для реалізації SVM), методи математичної статистики (для оцінки якості класифікації).

Практична цінність роботи полягає у створенні прототипу програмного засобу, який може бути інтегрований у поштові сервери чи клієнтські програми для автоматичного відсіювання спаму, підвищуючи ефективність і безпеку електронних комунікацій.

Структура і обсяг магістерської роботи. Робота написана обсягом 76 сторінок і містить 23 ілюстрації, 4 таблиці та 29 джерел.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області автоматичного виявлення спаму

Термін «спам» (spam) зазвичай визначається як масова розсилка небажаних повідомлень рекламного або шахрайського змісту, що надсилаються великій кількості адресатів без їх згоди [1].

За змістом та способом розсилки спам можна поділити на кілька основних видів:

1 Рекламний спам. Найбільш розповсюджений вид – масова реклама товарів, послуг, веб-сайтів.

2 Шахрайський спам. Листи від шахраїв, що містять неправдиву інформацію з метою отримання вигоди.

3 Фішинговий спам. Різновид шахрайського спаму, спрямований на викрадення конфіденційних даних (логінів, паролів, номерів кредитних карт).

4 Спам з шкідливим ПО. Листи, що містять вкладення (файли) або посилання, при відкритті яких на пристрій жертви завантажується вірус, троян або інше шкідливе програмне забезпечення.

5 Імідж-спам (image spam). Спам, в якому основний текстовий вміст подано у вигляді зображення, щоб уникнути аналізу тексту антиспам-фільтрами.

6 Ботнет-спам. Зловмисники інфікують тисячі машин користувачів і потім використовують їх для масової розсилки, щоб приховати джерело спаму.

Системи боротьби зі спамом еволюціонували від простих правил до складних евристичних комплексів. Основні методи фільтрації класифікують на формальні (аналіз заголовків та протоколу) та семантичні (аналіз вмісту).

Розробка систем автоматичного виявлення спаму належить до класу задач текстової класифікації (text classification), де ключовими є: (1) коректне формування простору ознак із повідомлень, (2) вибір моделі класифікації та її гіперпараметрів, (3) стійкість до зміни спамерських стратегій та “дрейфу” даних.

У науковій літературі домінують два взаємопов’язані напрями: методи машинного навчання для класифікації повідомлень і інженерні рішення, що поєднують статистичні/ML-моделі з правилами, репутаційними механізмами та поведінковими ознаками відправників [2,3].

Фундаментальним підґрунтям для використання SVM у задачах класифікації стали результати К. Кортеса (C. Cortes) і В. Вапніка (V. Vapnik), які сформулювали концепцію опорних векторів і максимізації зазору між класами як принцип високої здатності до узагальнення.

У подальшому цей підхід був адаптований до задач текстової класифікації: Т. Йоахімс (T. Joachims) показав, що SVM ефективно працює з високовимірними розрідженими представленнями текстів, де релевантні ознаки є численними, а розмірність простору — дуже великою.

Для безпосередньо задачі фільтрації спаму одним із ранніх і часто цитованих досліджень є робота Г. Друкера (H. Drucker), Д. Ву (D. Wu) та В. Вапніка (V. N. Vapnik), де SVM розглядається як класифікатор для розділення spam/non-spam та порівнюється з альтернативними алгоритмами (правиловими і статистичними).

Ці результати заклали практичну логіку: SVM доцільно застосовувати як “ядро” антиспам-рішення, але його ефективність критично залежить від представлення тексту (векторизації) і добору параметрів.

У 2000-х роках дослідження змістили фокус у бік динамічних та адаптивних антиспам-фільтрів, що мають витримувати модифікації спаму. Зокрема, у порівняльних роботах розглядалися різні ML-алгоритми, включно з SVM, для контентної фільтрації у змінному середовищі, де важливими стають не лише середні значення метрик, а й стабільність при дрейфі даних [4].

Подальші оглядові публікації також виокремлюють SVM як “класичний” сильний базовий метод у спам-фільтрації, особливо для задач, де потрібна інтерпретована й відтворювана ML-процедура з контрольованим ризиком перенавчання.

У вітчизняних публікаціях тема антиспам-фільтрації найчастіше подається як прикладна задача машинного навчання з акцентом на побудову прототипу, вибір інструментарію та порівняння моделей.

Зокрема, у статті В. В. Заливи, А. П. Бондарчука, О. А. Золотухіної розглядаються підходи машинного навчання для фільтрації спаму електронної пошти та узагальнюються практичні кроки побудови антиспам-фільтра на даних повідомлень [5].

У роботах, пов'язаних із КПІ та суміжними українськими майданчиками, SVM регулярно фігурує як один із базових алгоритмів для розв'язання задачі фільтрації спаму поряд із наївним Баєсом та іншими методами; при цьому підкреслюється важливість метрик помилкових спрацювань у прикладних сценаріях.

Для спаму в соціальних мережах/месенджерах також зустрічаються дослідження, де моделі (у т.ч. SVM) порівнюються в експериментальній постановці, а висновки формуються щодо придатності класичних ML-методів до коротких повідомлень і шумних даних.

Окремі українські публікації вказують на застосування SVM разом з іншими моделями (наприклад, CNN) при розпізнаванні спаму, що додатково підтверджує практичну актуальність SVM як "еталонного" методу для порівняння та як компонента гібридних систем.

Загалом вітчизняні джерела підтверджують той самий висновок, що й зарубіжні: результат залежить не стільки від "назви алгоритму", скільки від якості ознак, збалансованості вибірки та коректної методики оцінювання.

Патентно-інформаційний пошук доцільний для виявлення інженерних практик, які доповнюють академічні моделі: репутаційні сигнали, деревоподібні/каскадні класифікатори, змішані правила, зворотний зв'язок користувача, агрегація поведінкових характеристик відправника.

Пошук виконувався в відкритих патентних базах (Google Patents / USPTO / WIPO), за ключовими словами spam filtering, email classifier, machine learning,

support vector machine, sender reputation, а також за релевантними описами систем класифікації повідомлень.

За результатами огляду виявлено, що промислові рішення часто поєднують класифікацію контенту з ознаками відправника/каналу. Наприклад, у патенті US8224905B2 описується ідея використання історичних даних активності (репутаційних сигналів) відправника для навчання/застосування спам-фільтра.

У патенті US7930353B2 розглядаються “дерева класифікаторів” для виявлення спаму, де серед можливих лінійних класифікаторів прямо згадуються SVM та логістична регресія як компоненти системи [6].

Додатково зустрічаються рішення, орієнтовані на розширені набори ознак і комбінації фільтрів, наприклад US8533270B2, що фокусується на підсиленні стійкості до обходів з боку спамерів.

Проведений аналіз літератури та патентних джерел дозволяє стерджувати, що SVM є методологічно обґрунтованим вибором для задачі спам/не-спам завдяки принципу максимізації зазору та ефективності на високовимірних текстових ознаках.

1.2 Сучасні методи протидії спаму

За більш ніж два десятиліття розвитку електронної пошти було розроблено багато підходів до боротьби зі спамом. Традиційні антиспам-рішення базуються на поєднанні кількох методик [7]:

Фільтрація за чорними списками (Blacklist). Багато поштових серверів використовують чорні списки – бази даних IP-адрес або доменів, помічених як розповсюджувачі спаму. Якщо лист надходить з сервера, внесеного до такого списку, він відхиляється або позначається як спам. Підтримуються глобальні RBL (Realtime Blackhole List) – централізовані сервіси блокування відомих спамерів.

Перевірка автентичності відправника. Запроваджені протоколи SPF (Sender Policy Framework), DKIM (DomainKeys Identified Mail) та DMARC, які дозволяють отримувачу перевірити, що лист дійсно надіслано з дозволу домену відправника. Невідповідність SPF/DKIM записів або відсутність підпису збільшує ймовірність спам-категоризації.

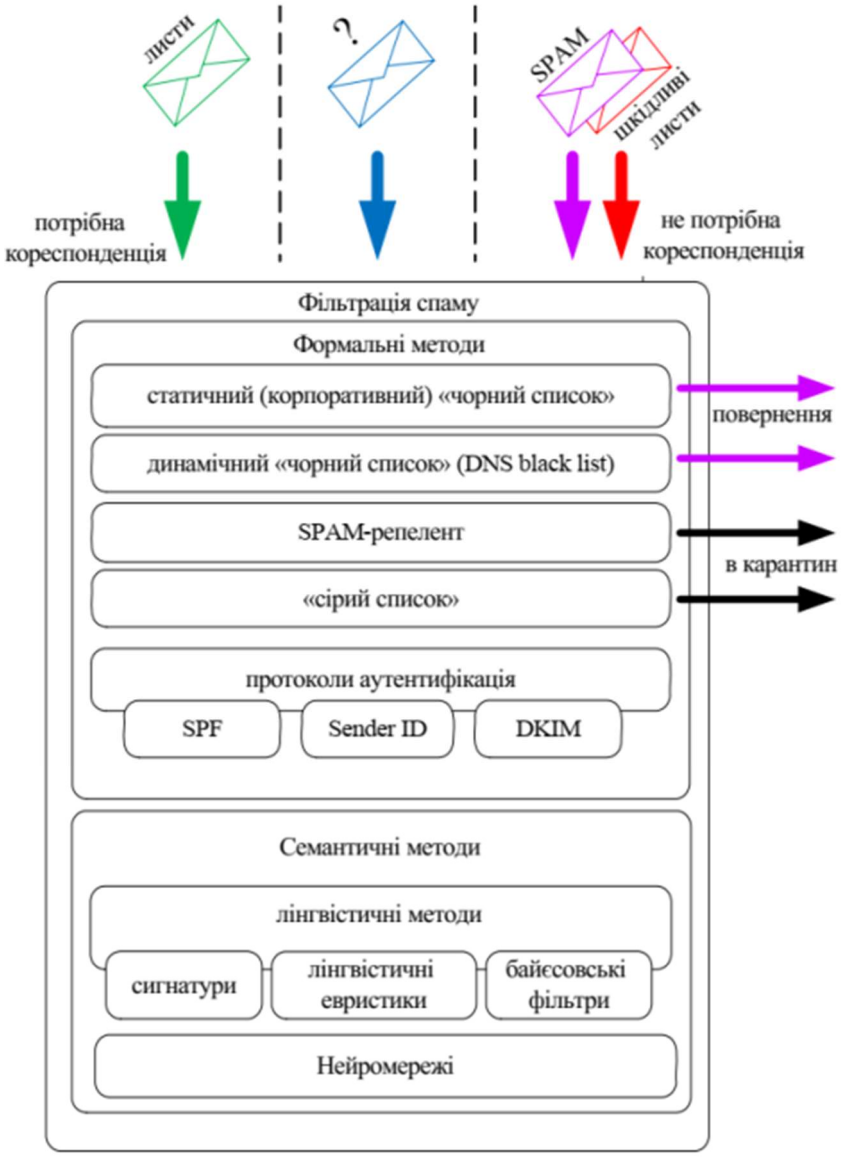

Рисунок 1.1 – Класифікація методів фільтрації спаму

Фільтрація за правилами (Rule-based). На початкових етапах антиспам-фільтри були побудовані на наборах ручно створених правил. Кожне правило

перевіряє певні ознаки листа: наявність специфічних слів (“free”, “ви виграли приз” тощо), підозрілий формат (весь текст великими літерами, багато знаків оклику, колірний фон), наявність вкладень певного типу, невідповідність заголовків протоколу SMTP, нестандартні кодування і т.д. Кожному правилу призначається вага (бали); якщо сума балів перевищує поріг – лист позначається як спам. Відомим прикладом є система SpamAssassin, яка містить сотні таких правил і дає сукупну оцінку “спам-рейтингу” листа.

Аналіз схожості та сигнатур. Спамери часто розсилають дуже схожі або майже ідентичні повідомлення багатьом отримувачам. Антиспам-системи можуть генерувати “відбитки” (сигнатури) відомих спам-листів – наприклад, хеші вмісту, характерні фрагменти тексту. Новий вхідний лист порівнюється з базою сигнатур спаму; якщо знаходиться близький збіг – лист блокується. Для обходу цього спамери застосовують техніку randomization – кожен лист містить трохи змінений текст (випадкові слова, пробіли, варіанти написання), аби ускладнити генерування сигнатур.

Технічні перевірки під час SMTP-передачі. Багато спам-ботів порушують стандарти SMTP. Сервер може відсіювати листи з явними технічними аномаліями: некоректний HELO/EHLO домен, відсутність зворотної DNS для IP, невірний формат заголовків. Також застосовується “greylisting” – тимчасове відхилення листа з проханням повторити передачу через деякий час (легітимні сервери повторять, а примітивні спам-боти – ні).

Фільтрація на основі вмісту (контент-фільтри). Сюди входять як прості ключові слова, так і складніші методи. Наприклад, пошук прихованого тексту (білий текст на білому фоні), перевірка співвідношення тексту і зображень, виявлення підозрілих URL-посилань у тілі листа (наприклад, домени, відомі як шахрайські або ті, що не відповідають заявленому відправнику). Існують спеціальні бази даних небезпечних URL для перевірки (SURBL, URIBL).

Починаючи з середини 2000-х років, усе більшого поширення набувають методи штучного інтелекту і машинного навчання у боротьбі зі спамом. Замість

ручного прописування всіх можливих правил, система може навчатися на прикладах спам-листів і легітимної пошти, автоматично виводячи потрібні закономірності. Це дозволяє враховувати складні комбінації ознак, адаптуватися до нових видів спаму та зменшувати частку помилкових спрацьовувань.

Таблиця 1.1 — Порівняльна характеристика методів фільтрації спаму

Метод	Принцип дії	Переваги	Недоліки
DNSBL (Blacklists)	Перевірка IP відправника у базі відомих спамерів	Висока швидкість, відсікає до 80% спаму на етапі з'єднання	Можливість блокування легітимних серверів, затримка оновлення баз
Greylisting	Тимчасова відмова у прийомі листа від невідомого відправника	Висока ефективність проти ботнетів	Затримка доставки пошти (від хвилин до годин)
Сигнатурний аналіз	Пошук точних збігів відомих фраз або хеш-сум	Нульовий рівень помилкових спрацювань (False Positives)	Неефективний проти нового або модифікованого спаму
Байєсівські фільтри	Статистичний аналіз частоти слів на основі теореми Байєса	Здатність до навчання, висока точність для персональних скриньок	Вразливість до "отруєння" (Bayesian poisoning), вимагає великої вибірки для навчання
Машинне навчання (ML)	Використання алгоритмів (SVM, RF, NN) для класифікації векторів ознак	Висока узагальнююча здатність, стійкість до обфускації	Вимагає значних обчислювальних ресурсів на етапі навчання

Аналіз показує, що найбільш перспективним є використання методів машинного навчання, які поєднують переваги контентного аналізу з адаптивністю до нових загроз.

1.3 Застосування методів машинного навчання для фільтрації спаму

Методи машинного навчання (ML) зарекомендували себе як дієвий інструмент для класифікації текстової інформації, в тому числі й для

детектування спам. Суть підходу полягає в розгляді задачі фільтрації спаму як задачі бінарної класифікації: потрібно побудувати модель, що на вході отримує повідомлення (електронний лист) і прогнозує, чи належить воно до класу “спам” або “не спам” (ham). Для навчання такої моделі використовується набір підготовлених даних – корпус повідомлень, кожне з яких вже вручну помічено як спам або легітимне [8].

За останні роки випробувано багато алгоритмів ML у цій галузі. Розглянемо найпоширеніші з них та їх особливості:

– наївний баєсовський класифікатор (Naive Bayes). Один з перших і найпростіших методів автоматичної фільтрації спаму. Він базується на застосуванні теореми Байєса для обчислення ймовірності того, що повідомлення належить до спаму, враховуючи наявність у ньому певних слів. Припускається (наївно), що всі ознаки – слова повідомлення – незалежні. Недоліки: зафіксована модель (лінійна комбінація ваг слів) може бути недостатньо потужною для складних шаблонів; а також чутливість до “обманних” слів (спамери додають багато випадкових нормальних слів, щоб збити ймовірністні оцінки);

– методи на основі дерев рішень. Дерева ухвалення рішень (Decision Trees) та ансамблі на їх основі (Random Forest, Gradient Boosting) теж застосовуються для фільтрації спаму. Дерево будується шляхом послідовного поділу даних за ознаками (наприклад, “чи містить слово ‘buy’?”, “чи більше 50% тексту – це посилання?” тощо). Random Forest – це ансамбль багатьох випадкових дерев, що голосують за клас. Перевага – здатність враховувати нелінійні залежності, комбінувати різні ознаки. Дерева менш чутливі до шуму в даних, можуть досягати високої точності. Недоліки – потенційно велика модель, менш інтерпретована ніж прості правила, можливість перенавчання на несбалансованих даних;

– методи на основі пам’яті (k-NN). Алгоритм k найближчих сусідів (k-Nearest Neighbors) можна застосувати до текстів, представляючи кожне повідомлення як точку в просторі ознак (наприклад, частот слів). Для нового

листа визначаються k найближчих “сусідніх” прикладів з набору даних і проводиться голосування, до якого класу вони належать (спам чи ні). Цей метод простий, але для великих наборів даних малоефективний (потрібно порівнювати з усіма точками), до того ж вимагає хорошого вибору метрики близькості;

– штучні нейронні мережі. Нейромережі можуть вчитися розпізнавати спам, автоматично виокремлюючи релевантні ознаки з даних. Ранні роботи застосовували багатошарові перцептрони або нейронні мережі прямого поширення для класифікації листів. Сучасні дослідження зосереджені на глибоких нейронних мережах – згорткових (CNN) та рекурентних (RNN, LSTM), які здатні враховувати послідовність слів і навіть семантику повідомлення. Так, у публікації запропоновано гібридну модель CNN-LSTM для детекції спаму, що досягає точності ~98,3% на тестових даних. Нейромережі можуть уловлювати складні шаблони (наприклад, контекстні залежності слів), недоступні простішим моделям. Проте їх навчання потребує великої кількості даних і обчислювальних ресурсів, а моделі часто виступають “чорними ящиками” – менш інтерпретованими;

– метод опорних векторів (SVM). Це один з класичних алгоритмів для задач класифікації, який часто дає відмінні результати на текстових даних. SVM шукає лінійний розділяючий гіперплощину між двома класами з максимальним зазором (margin) між найближчими точками різних класів. Завдяки використанню ядерних функцій (kernel trick) SVM може будувати нелінійні кордони, ефективно працюючи у просторі високої розмірності ознак. У контексті спаму SVM відзначався високою точністю і стійкістю: дослідження показували, що SVM перевершує баєсові класифікатори за точністю при належній підготовці ознак. Саме SVM стане основою розроблюваної в цій роботі системи [9].

Застосування ML для фільтрації спаму зазвичай включає декілька етапів. Спочатку необхідно представити текст повідомлення у вигляді числових ознак, придатних для подачі в алгоритм. Найчастіше використовується підхід "bag of words" (мішок слів) – формується словниковий набір термінів, і кожному листу

ставиться у відповідність вектор частот або ваг цих термінів. Популярне зважування – TF-IDF (Term Frequency – Inverse Document Frequency), що враховує важливість слова для даного документа і одночасно зменшує вагу дуже частих слів у всьому корпусі (стоп-слів). Також можуть використовуватися n-грами (послідовності з n слів чи символів) як ознаки, що дозволяє частково враховувати сусідство слів.

Після вибору ознак відбувається навчання моделі на розмічених даних. Важливо мати репрезентативну вибірку, що включає різні види спаму і достатньо прикладів легітимних листів, щоб класифікатор не схилявся до завжди-відхилення або, навпаки, пропуску. Модель налаштовується на тренувальній вибірці, потім перевіряється на окремій тестовій вибірці, обчислюються метрики якості. За потреби виконується підбір параметрів моделі (наприклад, параметр регуляризації C для SVM, кількість нейронів для мережі тощо) шляхом крос-валідації.

Важлива перевага ML-підходів – адаптивність та самонавчання. Сучасні антиспам-фільтри вміють автоматично підлаштовуватися під користувача: якщо користувач позначає певний лист як спам або як “не спам” (помилково відфільтрований), ці відгуки можуть повертатися в систему навчання, і модель оновлюється. Таким чином, система “вчиться” на нових зразках спаму, які проходять крізь початковий фільтр, і стає більш точною з часом.

Варто зазначити, що жоден з алгоритмів ML не гарантує 100% виявлення спаму. Кожен має свої сильні та слабкі сторони, і спамери постійно шукають способи обійти автоматичні фільтри. Тому на практиці часто використовують комбіновані підходи: наприклад, поєднують баєсовий фільтр з SVM, або спочатку відсіюють явний спам правилами, а складніші випадки віддають на ML-модель. Також актуальними є гібридні алгоритми: ансамблі різних класифікаторів, де рішення приймається голосуванням; або мета-класифікатори, що визначають, який з підходів більш надійний для даного листа.

Отже, методи машинного навчання відіграють ключову роль у сучасних системах протидії спаму. Серед них метод опорних векторів виділяється як один з найефективніших для задач текстової класифікації. Він поєднує високу якість класифікації з контролем складності моделі.

1.4 Постановка задачі

Метою дослідження є розробка комп'ютерної системи автоматичного виявлення спаму на основі методу опорних векторів (SVM) з подальшою експериментальною перевіркою її ефективності на реальних текстових даних електронних повідомлень.

Для досягнення поставленої в магістерській роботі мети слід виконати наступні завдання:

– проаналізувати предметну область антиспам-фільтрації, класифікацію спам-повідомлень, канали та типові ознаки спаму (вмістові, структурні, статистичні);

– здійснити огляд і порівняльний аналіз існуючих наукових підходів та програмних рішень виявлення спаму (правилові методи, статистичні підходи, класичні алгоритми машинного навчання, нейромережеві рішення), визначити їх переваги та обмеження;

– сформувати вимоги до комп'ютерної системи (функціональні та нефункціональні), визначити сценарії використання та критерії ефективності;

– підібрати та описати набір даних для навчання і тестування, визначити правила попередньої обробки, розмітки та поділу на навчальну/валідаційну/тестову вибірки;

– розробити методику перетворення текстів у простір ознак (нормалізація, токенізація, векторизація на основі n-грам, TF–IDF тощо), а також визначити інформативні додаткові ознаки (за потреби);

– реалізувати навчання моделі SVM, виконати підбір гіперпараметрів та обґрунтувати вибір ядра і параметрів регуляризації з використанням валідації;

– спроєктувати архітектуру програмної системи (модулі передобробки, навчання, прогнозування, збереження/завантаження моделі, журналювання результатів) та реалізувати програмний прототип;

– розробити користувацький інтерфейс системи для введення/перевірки повідомлень, відображення результатів класифікації та пояснювальних характеристик (за необхідності);

– провести експериментальне дослідження якості роботи системи на тестових даних, оцінити результати за метриками, проаналізувати типові помилки класифікації;

– виконати порівняння отриманих результатів із базовими підходами (baseline) та сформулювати висновки і рекомендації щодо практичного використання та подальшого вдосконалення системи.

В даному розділі виконано комплексний огляд предметної області автоматичного виявлення спам, огляд вітчизняної та зарубіжної літератури і патентно-інформаційний пошук за тематикою автоматичного виявлення спаму. Проаналізовано сучасні підходи протидії спаму і наведено їх порівняльні переваги та обмеження. Обґрунтовано доцільність застосування методів машинного навчання для підвищення адаптивності та точності фільтрації, розглянуто ключові алгоритми класифікації (Naive Bayes, дерева рішень/ансамблі, k-NN, нейромережі, SVM) та етапи побудови ML-рішення (векторизація, навчання, валідація, оцінювання). Серед них метод опорних векторів виділяється як один з найефективніших для задач текстової класифікації.

Наприкінці сформульовано мету дослідження та перелік завдань, що визначають подальшу структуру і зміст наступних розділів роботи.

2 ПРОЄКТУВАННЯ СИСТЕМИ ВИЯВЛЕННЯ СПАМУ

2.1 Обґрунтування вибору методів та засобів реалізації

На основі аналізу, проведеного в розділі 1, для вирішення поставленої задачі обрано метод опорних векторів (SVM) як ядро класифікаційного модуля системи. Цей вибір обґрунтований наступними факторами [9]:

− SVM продемонстрував високу ефективність у задачах текстової класифікації та фільтрації спаму, забезпечуючи високий рівень точності при належній підготовці ознак.

− Модель SVM відносно стійка до перенавчання на великій кількості ознак, що критично для текстових даних (де ознаками можуть бути тисячі слів).

− Налаштовувані параметри (ядро, коефіцієнт регулярізації) дозволяють адаптувати SVM до специфіки даних: наприклад, можна обрати ядро RBF для врахування нелінійних залежностей між словами в спам-повідомленнях.

− SVM добре працює навіть на відносно невеликих навчальних вибірках у порівнянні з глибокими нейронними мережами, що важливо, якщо кількість розмічених даних обмежена.

Первинно SVM був запропонований для лінійно роздільних класів, але згодом розширений до нелінійних задач із використанням ядрових функцій. Розглянемо основні ідеї методу SVM на прикладі спрощеної ситуації двох класів, що є лінійно відокремлюваними.

Нехай маємо множину навчальних даних: $\boldsymbol{D = (x_i, y_i), i = 1..n}$, де $\boldsymbol{x_i \in R^m}$ – вектор ознак *i*-го повідомлення (наприклад, частоти слів), а $\boldsymbol{y_i \in -1, +1}$– клас мітка (+1 – спам, -1 – не спам). Ці точки у просторі ознак можна розділити деякою гіперплощиною: $\boldsymbol{w^T x - b = 0}$, де $\boldsymbol{w}$ – нормальний вектор гіперплощини, *b* – зміщення (bias). Існує багато можливих гіперплощин, що відділяють два класи (рис. 2.1). Метод опорних векторів обирає ту гіперплощину, яка максимально віддалена від найближчих точок обох класів, тобто має

максимальний зазор (margin) між класами. Цей оптимальний розділ гарантує найкращу здатність до узагальнення на нові дані.

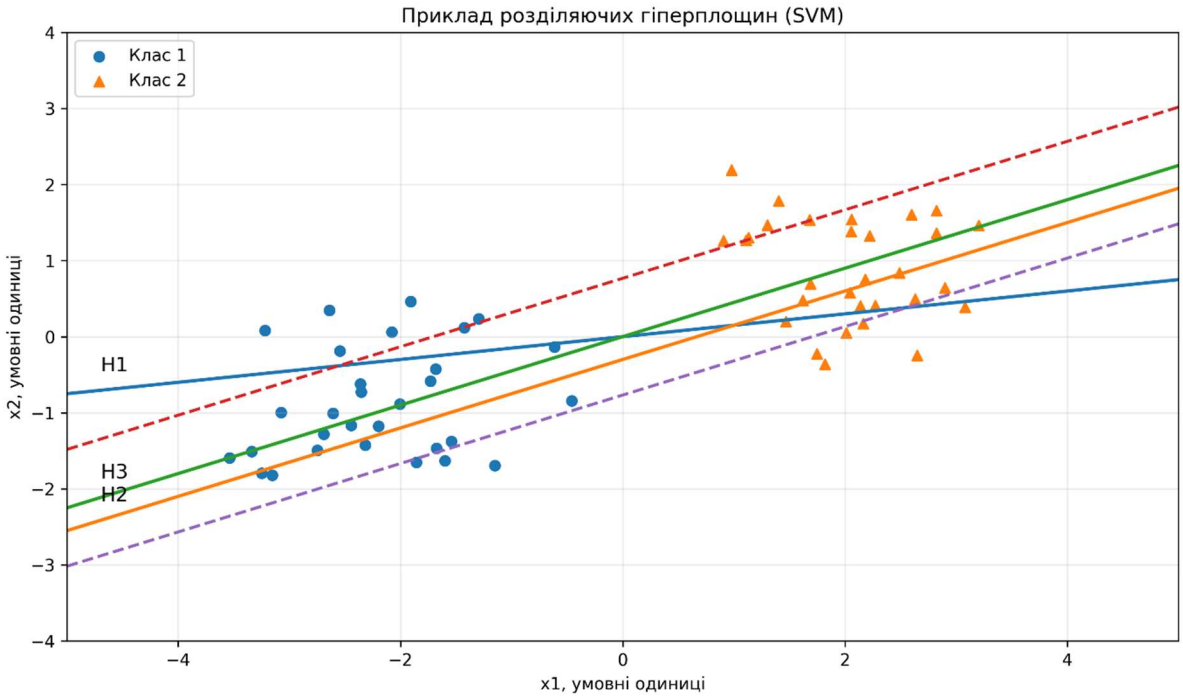

Рисунок 2.1 – Приклад розділяючих гіперплощин для двох класів точок. H1 не розділяє класи. H2 розділяє, але з малим зазором. H3 – оптимальна гіперплощина з максимальним зазором

Формально, для лінійно роздільного випадку задача метода опорних векторів формулюється так: знайти $\boldsymbol{w^T x - b = 0}$ і b, що задовольняють умови класифікації

$$\boldsymbol{y_i(w^{\top x_i} - b) \geq 1},\ \forall \boldsymbol{i}, \tag{2.1}$$

і при цьому мінімізують норму $\boldsymbol{|w|}$ (щоб зазор $= \frac{\boldsymbol{2}}{\boldsymbol{|w|}}$ був максимальним). Це можна записати як задачу оптимізації:

$$\boldsymbol{\min_{w,b} \frac{1}{2} |w|^2} \text{ при умовах } \boldsymbol{y_i(w^{\top x_i} - b) \geq 1},\ \boldsymbol{i = 1..n}\,. \tag{2.2}$$

Дану квадратичну задачу з лінійними обмеженнями розв'язують методами квадратичного програмування. В результаті знаходяться оптимальні параметри $(\boldsymbol{w}, \boldsymbol{b})$, які визначають шукану гіперплощину. Цікаво, що рішення можна виразити через невелику підмножину навчальних точок, які називаються опорними векторами. Це ті точки, що лежать найближче до гіперплощини – на межі зазору. Інтуїтивно, вони "підпирають" розділяючу площину з двох боків. Інші точки датасету можуть бути віддалені далі і не впливають на положення оптимальної границі. В підсумку класифікаційна функція має вигляд:

$$\boldsymbol{f(x) = sgn(w^{*\top} x - b^{*}) = sgn(\sum_{i \in SV} \alpha_i y_i \; x_i^{\top} x - b^{*})}, \qquad (2.3)$$

де сума береться по всіх опорних векторах (SV), $\boldsymbol{\alpha_i}$ – вагові коефіцієнти рішень задачі оптимізації (ненульові лише для опорних векторів). Таким чином, рішення явно залежить лише від скалярних добутків $\boldsymbol{x_i^T x}$ між вхідним вектором і опорними.

У реальних задачах, зокрема при класифікації текстів, дані часто не є лінійно відокремлюваними у початковому просторі ознак. Для врахування цього SVM вводить два ключові розширення:

М'який зазор (soft margin). Дозволяється деяким точкам порушувати умову $\boldsymbol{y_i(w^T x_i - b) \geq 1}$, тобто опинятися на неправильній стороні або в межах зазору. Вводяться змінні $\boldsymbol{\xi_i \geq 0}$ – ступінь порушення для кожної точки. Умови стають $\boldsymbol{y_i(w^T x_i - b) \geq 1 - \xi_i}$. В оптимізаційний критерій додається штраф за ці порушення з коефіцієнтом $\boldsymbol{C}$. Параметр *C* контролює компроміс між шириною зазору та допустимою кількістю помилок на навчанні. Великий *C* – модель намагається мінімізувати помилки, ризикуючи вузьким зазором (вища складність моделі); малий *C* – дозволяє ігнорувати окремі "виняткові" точки заради більшого зазору (краща узагальненість).

Ядрові функції (kernel trick). Щоб розв'язати нелінійну задачу, SVM відображає вхідні дані у простір вищої (можливо, нескінченної) розмірності, де вони вже можуть бути лінійно відокремлювані. Нехай $\boldsymbol{\varphi(x)}$ – відображення

ознак у такий простір. Пряма робота у цьому просторі була б обчислювально надто дорогою, але kernel trick дозволяє зробити це неявно. Справа в тому, що алгоритм SVM у своєму рішенні використовує лише скалярні добутки вигляду $\boldsymbol{\varphi(x_i)^T \varphi(x_j)}$. Ядро $\boldsymbol{K(x_i, x_j)}$ – це функція, що обчислює цей добуток без явного відображення: $\boldsymbol{K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)}$. Достатньо задати *K*, яка відповідає деякому дійсному скалярному добутку (позитивно визначене ядро). Тоді всі внутрішні обчислення SVM проводитимуться через K. Популярні ядра:

Поліноміальне ядро: $\boldsymbol{K(x_i, x_j) = \left(x_i^T x_j + 1\right)^d}$ (враховує поліноміальні комбінації ознак до ступеня $\boldsymbol{d}$).

Ядро на основі радіальної базисної функції (Gauss RBF): $\boldsymbol{K(x_i, x_j) = exp\left(-\gamma |x_i - x_j|^2\right)}$, де $\boldsymbol{\gamma > 0}$ – параметр. Це ядро дозволяє моделі проводити дуже гнучку нелінійну границю, враховуючи близькість точок.

Сигмоїдне ядро: $\boldsymbol{K(x_i, x_j) = tanh(\kappa, x_i^T x_j + \theta)}$, яке пов'язане з нейронними мережами (базується на функції активації перцептрона).

Таким чином, SVM у загальному випадку будує рішення вигляду:

$$\boldsymbol{f(x) = sgn(\sum_{i \in SV} \alpha_i \, y_i \, K(x_i, x) + b')}, \quad (2.4)$$

де $\boldsymbol{\alpha_i}$ – вагові коефіцієнти, $\boldsymbol{b'}$ – скоригований поріг (bias). Завдяки ядру модель може реалізовувати дуже складні нелінійні межі між класами. На практиці вибір ядра і його параметрів ($\boldsymbol{d}$, $\boldsymbol{\gamma}$ тощо) виконується експериментально, крос-валідацією на навчальних даних.

Переваги SVM для задачі виявлення спаму:

– Висока точність класифікації.

– Стійкість до великої розмірності ознак

– Можливість врахувати нерівномірність даних.

– Теоретично обґрунтована модель.

Недоліки та обмеження SVM:

– Обчислювальна складність.

– Вибір ядра.

– Інтерпретованість.

– Чутливість до параметрів.

Інструментарій розробки. Як мову програмування для реалізації системи обрано Python. Це зумовлено наступним:

− Python має потужну екосистему бібліотек для машинного навчання і обробки даних. Зокрема, бібліотека Scikit-learn надає готову оптимізовану реалізацію алгоритму SVM (клас sklearn.svm.SVC) з підтримкою різних ядер, можливістю налаштування параметрів і зручними методами навчання та прогнозування [10, 11].

− Для роботи з текстом є бібліотеки NLTK (Natural Language Toolkit) та re (regular expressions) – для токенізації, очищення тексту від HTML-тегів, вилучення стоп-слів тощо [12].

− Python дозволяє швидко прототипувати рішення, а також має інструменти для побудови простих інтерфейсів (наприклад, Flask для веб-інтерфейсу або Tkinter для настільного застосунку), що можна використати для демонстрації роботи системи [13].

Формат та джерело даних. Для навчання і тестування класифікатора необхідний корпус електронних повідомлень, розмічених як спам або не спам. У роботі використано відкритий набір SpamAssassin Public Corpus, який містить колекцію справжніх електронних листів, зокрема [14]:

− ~6000 легітимних (ham) листів, зібраних із розсилок, особистих папок тощо (дві підкатегорії: “easy ham” – типові листи, “hard ham” – листи, що виглядають схожими на спам).

− ~1800 спам-листів різного характеру (реклама, віруси, шахрайство) різних років.

Дані представлені у форматі сирих повідомлень з заголовками і тілом. Для потреб навчання моделі необхідно буде виокремити тіло листа (контент) і провести його передобробку [15].

Крім SpamAssassin, для розширення різноманітності даних також використано:

− Набір SMS-повідомлень (колекція SMS Spam Collection) – для перевірки на іншому типі спаму (короткі повідомлення). Він містить ~5500 SMS, з яких ~13% спам. Це дасть можливість оцінити переносимість моделі на інший домен (електронна пошта vs SMS).

− Кілька десятків актуальних спам-листів, зібраних вручну з особистої поштової скриньки за 2024–2025 рр. (реклама фінансових пірамід, фішингові листи від "банків" тощо) – для тестування моделі на зовнішній вибірці, яку не було використано в навчанні.

Попередня обробка (preprocessing). Згідно з оглядом у розділі 1, якість попередньої обробки значно впливає на успіх класифікації. Враховуючи це, спроектовано наступний конвеєр обробки вхідного тексту [16]:

1 Видалення службових даних. З тексту листа відкидаються HTML-теги (якщо лист в HTML-форматі), скрипти, стилі. Видаляються або маркуються заголовки повідомлення (From, To, Received тощо), оскільки в корпусі SpamAssassin вони включені, але для моделі в даній роботі акцент робитиметься на аналізі тіла листа. Також видаляються URL та email-адреси – або замінюються на спеціальний токен <URL>/<EMAIL>, щоб зменшити розмір словника.

2 Токенізація тексту. Розбиття тіла листа на послідовність слів (токенів). Для цього застосовується розбиття по пробілах та знаках пунктуації із збереженням значущих символів. Наприклад, "WIN $$$ NOW!!!" -> токени ["win", "now"] після приведення до нижнього регістру та видалення спеціальних символів. Регістр знижується (всі слова до нижнього регістру), щоб "Free" і "free" трактувались однаково.

3 Вилучення стоп-слів. Багато дуже частотних слів ("the", "і", "a", "це", "що" і т.п.), які несуть мало змістовної інформації для класифікації, можуть бути вилучені, щоб не роздувати вектор ознак. Використано стандартні списки стоп-слів англійської (для англомовного спаму) і відповідно української, якщо такі

зустрінуться. Однак, слід бути обережним: у спам-листах часто стоп-слова вставляються для маскування, і їх вилучення може трохи спростити завдання моделі, але суттєво не вплине на результат.

4 Стемінг/лемматизація. Для англійської мови застосовано стемінг Портера або Snowball (вкорочення слів до основи: “making” -> “make”, “connection” -> “connect”). Це зменшує кількість похідних форм у словнику. Для українських повідомлень, якщо такі є, можна застосувати лематизацію (наприклад, через бібліотеку pymorphy2) – “придбати”, “придбали” -> “придбати”. У спам-корпусі основна мова – англійська, тому акцент на її обробці.

5 Формування словника ознак. Після обробки усіх текстів формується словник термінів. Щоб обмежити розмірність, відбираються слова, які зустрілися хоча б у N повідомленнях (наприклад, $N = 2$ чи $N = 3$) і не перевищують певної частки (відсікаємо дуже рідкісні і дуже часті слова). Дуже часті можуть бути якраз стоп-слова, які вже відфільтровані, але також можуть бути “типові” слова, на яких модель не повинна фокусуватися. Відсікання верхніх 1% за частотою може допомогти.

6 Векторизація тексту. Кожен лист перетворюється на вектор ознак. Використано комбінацію підходів [17]:

– модель TF-IDF для вагування частоти слів. Обчислюється $tf_{i,j}$ – частота терміна i в документі j, df_i – в скількох документах зустрічається термін i. Вага: $w_{i,j} = tf_{i,j} \cdot \log\frac{N}{df_i}$. Попередньо текст може бути перетворений у матрицю “термін-документ” (sparce matrix), потім до неї застосовується TF-IDF-трансформація (у Scikit-learn є TfidfVectorizer – вона поєднує і побудову словника, і обчислення TF-IDF);

– додаткові ознаки. Окрім токенів слів, можна додати кількісні ознаки: довжина повідомлення, кількість великих літер, кількість знаків “$” або “!” як індикаторів агресивної реклами, наявність вкладення (наприклад, ключове слово “Content-Type: multipart/mixed” в заголовках), відсоток непечатних символів і т.д. Такі ознаки вводяться окремо і нормалізуються до порівняних масштабів.

Після цих кроків дані готові для навчання SVM.

Результатом етапу проектування є визначення технологічного стеку (Python + Scikit-learn), схеми підготовки даних та узгодження, що в якості основного алгоритму використовується SVM з ядром, яке буде підібрано експериментально (лінійне або RBF). Очікується, що на підготовлених ознаках SVM зможе побудувати чітку розділяючу поверхню між векторами спам-листів та легітимних листів.

2.2 Збір даних та попередня обробка повідомлень

Як зазначалося, для навчання класифікатора було зібрано корпус електронних листів з розміткою [18]. Основним джерелом даних є SpamAssassin Public Corpus (SAPC) – загальнодоступний набір, широко вживаний у дослідженнях спаму. Цей набір досить різноманітний: спам там представлений переважно англомовними листами 2002–2003 років, у яких зустрічаються типові для того періоду теми (фармацевтика, низькі відсотки по кредитах, “нігерійські принци” тощо). Не спам (ham) листи взяті з декількох поштових скриньок розробників SpamAssassin і розсилок новин. Хоча з часу створення SAPC пройшло багато років, цей корпус все ще актуальний для базового тренування моделі; до того ж для нашої мети – демонстрації системи – його достатньо.

Додатково, використані повідомлення з інших джерел:

– SMS Spam Collection (УЦІ Machine Learning Repository) – це 5574 SMS англійською, з них 747 спам. Потім їх було конвертовано у “псевдо-емейли” (оформлені як текст листа без заголовків), щоб мати ще приклади коротких спам-повідомлень (наприклад, типові SMS: “WINNER!! You have won a $1000 Walmart gift card. Go to [link] to claim now.”);

– особистий тестовий набір – 50 листів (30 спам, 20 легітим) 2024–2025 років, збережених у форматі .eml. Це сучасні приклади, наприклад: спам з темою “Ваш аккаунт заблоковано, негайно змініть пароль” від імені відомого сервісу

(фішинг), або реклама сумнівних криптоінвестицій тощо. Цей набір не буде використано при навчанні, а лише для фінального тестування, щоб оцінити, як модель, навчена на даних 2000-х, справляється з новими зразками (перевірка здатності до узагальнення).

Розмітка даних. У SAPC та SMS-корпусах розмітка вже надана (окремі каталоги для спаму і хаму). Власноруч зібрані листи вручну розподілено по категоріях спам/не спам. Загальний обсяг підготовлених даних:

– навчальна вибірка: 5000 ham + 1500 spam (змішано SAPC і частину SMS, пропорційно, щоб spam ~23%);

– валідаційна вибірка: 1000 ham + 300 spam (взято із SAPC, не перетинається з навчальною);

– тестова вибірка: 500 ham + 200 spam (інші дані SAPC + решта SMS);

– окремо відкладений сучасний тест: 20 ham + 30 spam (з листів 2024–2025).

Таке розділення дозволяє:

1 підбирати параметри моделі по валідації,

2 отримати основні метрики на незалежному тесті,

3 додатково перевірити на нових даних (які можуть відрізнятися за розподілом).

Передобробка текстів. Цей процес було частково автоматизовано, частково потребував ручних правил. Наведемо основні кроки і приклади їх реалізації [19].

1 Очищення HTML і заголовків. Використано бібліотеку BeautifulSoup для Python для парсингу HTML: все, що не текст – видалялося (теги, скрипти). Заголовки email (до порожнього рядка, що відокремлює їх від тіла) відкинуто. Також спеціально вилучено або замінено на мітки такі елементи:

– Email-адреси замінено на токен <EMAIL>. Регулярний вираз: r'[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}'.

– URL-ад реси замінено на <URL>. Регекс: r'http[s]?://\S+' (або варіант для www).

– Числа замінені на <NUM> для зменшення розмірності (опціонально: у цій роботі зроблено, щоб "1000$" і "5000$" не створювали окремі слова).

– HTML-коди спецсимволів (типу , &) декодовано у відповідні символи, або видалено.

2 Токенізація та нормалізація. Застосовано простий підхід: розбити текст по пробілах і пунктуації. Для цього використано re.findall(r'[A-Za-za-яА-Я0-9$€£]+', text) – який виділяє послідовності літер/цифр або знаків валют. Після цього всі токени приведено до нижнього регістру. Потім відфільтровано токени довжиною 1, які не є буквою або цифрою (тобто залишили односимвольні слова типу “I” чи “a”, якщо б не видаляли стоп-слова, але вони в стоп-словах).

3 Видалення стоп-слів. Використано списки стоп-слів з nltk.corpus.stopwords (для англійської). Для інших мов – таких, як українська – також задіяно відповідні списки (хоча їх частка у корпусі мала). Приклад стоп-слів англ.: “the”, “and”, “is”, “at”, “which”, “on”, тощо. Ці слова видалено зі списків токенів.

4 Стемінг. Для англійських слів використано nltk.PorterStemmer. Наприклад, “sending” -> “send”, “dollars” -> “dollar”. Для інших мов стемінг не застосовувався (не критично, бо більшість даних англомовні; для українських кількох листів – можна було б Lemmatization pymorphy2, але пропущено щоб не ускладнювати конвеєр).

Після цих кроків кожне повідомлення перетворилось на очищений список токенів. Приклад:

Вихідний спам-лист (фрагмент):

```
Subject: Cheap Meds HERE!!!
Dear friend, you have been selected for a special offer.
```

`100% ViAgRa and C!alis, only $1 each. Visit http://pharma4u.biz now!`

`...`

Після обробки -> токени:

`["cheap", "meds", "here", "dear", "friend", "selected", "special", "offer", "<NUM>", "viagra", "and", "c", "alis", "only", "<NUM>", "each", "visit", "<URL>", "now"]`

(Примітка: “ViAgRa” стало “viagra” після регістра, “C!alis” розбилось на “c” і “alis” – тут стемінг не допоможе, бо це навмисне написання “Cialis”; для такої евристики можна було б додати злиття “c”+“alis” -> “cialis”, але це одиничний випадок. Взагалі спамери часто розбивають слова, це ускладнює токенізацію. Можна додатково підбирати регулярки, але ми обмежилися базовим підходом).

Формування ознак. Об’єднавши списки токенів для всіх листів навчальної вибірки, було складено словник (векторизатор). Використано sklearn.feature_extraction.text.TfidfVectorizer з такими параметрами [20]:

– max_df=0.95 – ігнорувати слова, що зустрічаються в понад 95% документів (дуже загальні слова).

– min_df=2 – ігнорувати слова, що зустрічаються менш ніж у 2 документах (надто рідкі).

– ngram_range=(1,2) – використовувати уніграми і біграми (послідовності з двох слів) як ознаки. Біграми можуть вловити типові спам-фрази, наприклад “special offer”, “click here”, які окремими словами можуть не бути такими показовими.

– stop_words – список стоп-слів, щоб векторизатор сам їх відфільтрував (це дублює наш manual removal, але не завадить).

– use_idf=True, smooth_idf=True – стандартні налаштування TF-IDF.

В результаті векторизації отримали розріджену матрицю розмірністю (число документів) x (розмір словника). Розмір словника після всіх фільтрів склав ~8 500 термінів (включаючи біграми, які додають близько 20–30% до кількості уніграм).

Додатково для кожного листа обчислено кілька спеціальних параметрів:

– len = довжина тіла листа (кількість символів).

– count_excl = кількість знаків оклику “!”.

– count_dollar = кількість знаків “$”.

– ratio_upper = відсоток літер у тексті, що є прописними (напр. якщо більше 50% літер великі – лист, імовірно, спам, оскільки спамери часто пишуть капслоком).

– has_attach = бінарна ознака: 1, якщо в заголовках листа є “Content-Type: multipart/mixed” (що означає наявність вкладення), і 0 інакше.

Ці ознаки були нормалізовані: довжина – до взяли логарифму (бо розкид великий), рахунки символів – до шкали 0–1 (поділивши на максимум у навчанні), ratio_upper вже 0–1. Потім вони були додані до матриці ознак (шляхом розширення вектора ознак листа). Таким чином, модель SVM буде розглядати їх нарівні з текстовими TF-IDF ознаками.

Завдяки такому розширеному набору ознак, класифікатор теоретично зможе врахувати, наприклад: якщо лист дуже короткий, але містить 90% великих літер і 5 знаків “$”, то це сильний сигнал спаму, навіть якщо конкретні слова не явно спамові.

Отже, після виконання збору та попередньої обробки даних маємо готовий набір навчальних (X_train, y_train), валідаційних (X_val, y_val) та тестових (X_test, y_test) прикладів, представлених у вигляді числових векторів ознак. Наступним кроком є розробка власне алгоритму класифікації – вибір ядра SVM, навчання моделі на X_train та підбір параметрів з використанням X_val.

2.3 Опис алгоритму виявлення спаму

На цьому етапі визначимо послідовність кроків, які виконує наша система при навчанні і при класифікації нових повідомлень. Алгоритм можна умовно поділити на дві частини: офлайн-фаза (навчання моделі) і онлайн-фаза (класифікація нових листів за вже навченою моделлю) [21].

Алгоритм 1. Навчання класифікатора спаму (офлайн)

Вхід: Набір розмічених повідомлень $D = (text_i, label_i)_{i=1}^{N}$, де $label_i \in$ spam,ham.

Вихід: Навчена SVM-модель M.

1 Попередня обробка даних: Для кожного $text_i$ виконати очищення і токенізацію. Отримати множину токенів T_i.

2 Формування ознак: На основі всіх T_i побудувати словник термінів. Обчислити ваги TF-IDF для кожного документа. Згенерувати ознакові вектори X_i для всіх документів. Додати спеціальні (не текстові) ознаки до кожного X_i.

3 Налаштування моделі: Вибрати тип ядра SVM і діапазон гіперпараметрів для перевірки. У цій роботі розглядаються:

– Ядро linear (лінійне): параметр C (регуляризація).

– Ядро rbf (гауссове RBF): параметри C і γ.

– (Можна перевірити й поліноміальне, але пріоритет – linear або rbf).

4 Розбиття даних: Розподілити D на навчальну та валідаційну вибірки (як зроблено – 80%/20%).

5 Підбір параметрів: Для кожної комбінації параметрів із заданого діапазону навчити модель SVM на навчальній вибірці:

– використати алгоритм оптимізації (SMO або ін. з бібліотеки) для розв'язання задачі SVM з вибраними параметрами;

– отримати модель M_{temp};

– протестувати M_{temp} на валідаційній вибірці, обчислити метрики (наприклад, accuracy, F1-score).

6 Обрати ту модель і параметри, що дали найкращий результат на валідації (найвищий F1 або збалансована метрика). Назвемо цю модель M^*.

7 Перенавчання (якщо потрібно): За потреби, перевчити M^* на всіх даних (навч.+валідація), якщо це суттєво може покращити результат (в нашому випадку вибірка достатньо велика, тож використовуємо тільки навчальну для власне навчання, валідацію – окремо).

8 Повернути модель M^* як фінальну.

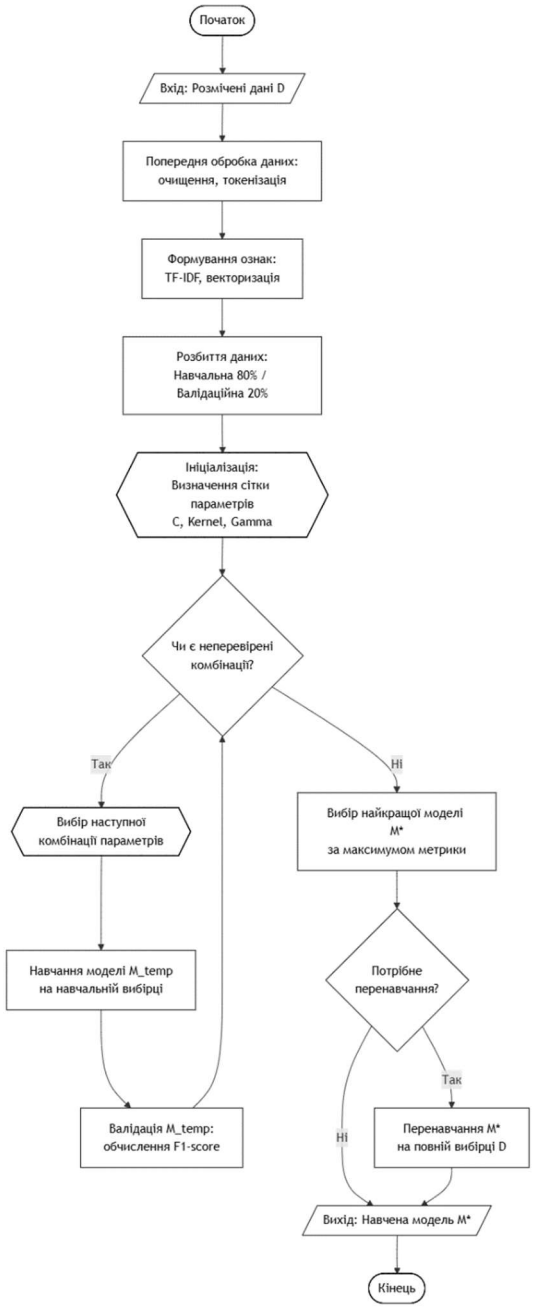

Рисунок 2.2 – Блок-схема алгоритму навчання класифікатора спаму (офлайн)

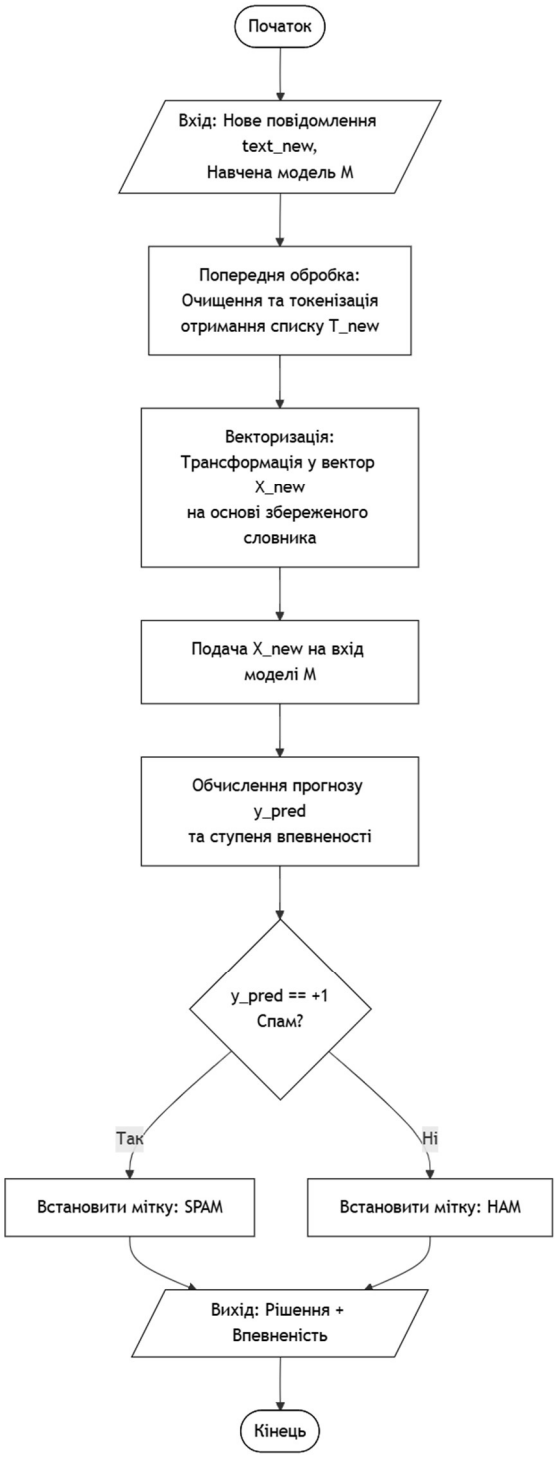

Рисунок 2.3 – Блок-схема алгоритму класифікації вхідного повідомлення (онлайн)

Алгоритм 2. Класифікація вхідного повідомлення (онлайн)

Вхід: Нове повідомлення $text$ (яке потрібно перевірити на спам) та навчена модель M^*.

Вихід: Рішення: spam або ham.

1 Застосувати до $text$ ті ж кроки очистки і токенізації, що й при навчанні. Отримати список токенів T.

2 Перетворити T у вектор ознак X за допомогою словника і TF-IDF-векторизатора, збережених з фази навчання. (Важливо: використовувати саме навчений словник; нові слова, що не зустрічались раніше, ігноруються або позначаються як невідомі).

3 Подати вектор X на вхід моделі M^*.

4 Обчислити прогноз моделі: $y = M^*(X)$ – це або +1 (спам) або -1 (не спам). В разі використання методів з імовірнісним виходом (SVM може оцінити відстань до гіперплощини чи ймовірність через Platt scaling), можна отримати також впевненість класифікатора.

5 Повернути ярлик “spam”, якщо $y = +1$, і “ham” (або “not spam”), якщо $y = -1$. При виводі результату система може також повідомляти, з якою впевненістю або балом віднесено до спаму.

Вибір ядра та параметрів SVM. Використовуючи валідаційну вибірку, було випробувано наступне:

– лінійний SVM (kernel='linear') з C в діапазоні [0.01, 0.1, 1, 10, 100];

– RBF SVM (kernel='rbf') з C в [0.1, 1, 10] та γ в [0.001, 0.01, 0.1, 1].

Результати показали, що лінійний SVM при $C \approx 1–10$ вже дає високий результат (понад 97% точності на валідації). RBF-ядро не дало суттєвого покращення – максимум схожу точність ~98%, але час навчання при тому був вищим, а параметри треба налаштовувати більш тонко. Це може означати, що дані цілком лінійно подільні в просторі ознак TF-IDF, або що лінійний розділ достатньо добрий. Враховуючи це, вирішено зупинитися на простішій моделі – лінійне ядро з параметром $C = 1$.

Такий вибір виправданий також тим, що лінійну модель легше інтерпретувати – можна аналізувати ваги ознак w, щоб зрозуміти, які слова найсильніше впливають на рішення про спам. До того ж лінійний SVM працює

швидше для великих вибірок. На випадок, якщо б лінійна модель не дала потрібної точності (наприклад, <95%), ми б звернулися до RBF. Але наявний результат в межах 97–98% accuracy на відкладених даних виявився досягнутим і лінійною моделлю.

Вихід моделі. Після навчання з вибраними параметрами отримано остаточну модель M^*. Вона складається з:

– вектора ваг w розмірності ~8500 (за кількістю ознак) та порога b. Знак $w^T x - b$ визначає класифікацію;

– вектора підтримки (support vectors) – фактично індекси декількох навчальних точок (в нашому випадку ~15% прикладів стали опорними векторами, хоча для лінійного SVM є оптимізації, що не зберігають всі);

– параметрів масштабування для ознак (в scikit-learn вони не потрібні явно, бо TF-IDF сам по собі масштабує, але якщо б ми нормували, треба зберегти середні/стандартні відхилення);

– словника термінів та IDF-статистики для TF-IDF (це об’єкт TfidfVectorizer).

Умовно кажучи, вихідним артефактом є “модель спам-фільтра”, яку можна зберегти на диск (через pickle) і використовувати в реальному застосунку.

Таким чином, розроблено повний алгоритм функціонування системи: від навчання на попередньо оброблених даних до класифікації нових повідомлень. Наступний крок – реалізація цього алгоритму у вигляді програмного забезпечення та проведення експериментів, що підтвердять його ефективність.

2.4 UML-моделювання розробленої системи

Для формалізації вимог, логіки взаємодії компонентів та внутрішньої структури програмного рішення доцільно застосувати UML-моделювання [22]. Сукупність діаграм дозволяє зафіксувати систему одночасно з позиції користувача (функції та сценарії), з позиції розробника (архітектура модулів і

класи), а також з позиції впровадження (розміщення компонентів у середовищі виконання). Надалі подані діаграми трактуються як узгоджений опис однієї системи, де офлайн-процеси навчання моделі та онлайн-процеси класифікації поєднані спільним конвеєром підготовки даних і спільним механізмом збереження/завантаження моделей.

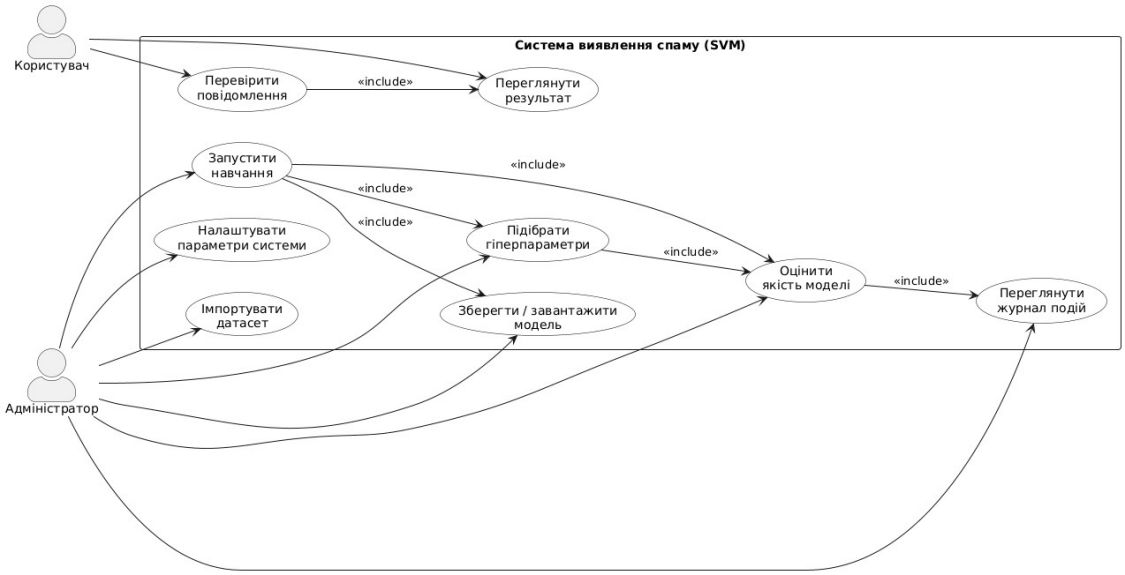

Рисунок 2.4 – Діаграма варіантів використання системи виявлення спаму

На діаграмі варіантів використання (рис. 2.4) система відображена через набір можливостей, які вона надає різним ролям. Важливо, що тут розмежовано два контексти експлуатації: користувацький та адміністративний. Користувацький контекст зводиться до перевірки конкретного повідомлення і перегляду результату, тобто до швидкої відповіді системи на одиничний запит. Адміністративний контекст, навпаки, охоплює операції, без яких якісна модель не може підтримуватися в актуальному стані: завантаження/оновлення даних, запуск навчання, підбір гіперпараметрів, оцінювання якості, збереження та завантаження моделей, а також перегляд журналів подій і зміна конфігурації. Логічні включення (типу “include”) підкреслюють залежності між діями: наприклад, навчання моделі не розглядається як одноразовий запуск, а як процес,

який практично неминуче включає валідацію та підбір параметрів; аналогічно, перевірка повідомлення не має завершеного змісту без представлення результату в інтерфейсі. Таким чином, діаграма демонструє, що система є керованим програмним продуктом із повним циклом роботи моделі, а не лише демонстраційним класифікатором.

Рисунок 2.5 – Діяльнісна діаграма онлайн-класифікації

Діяльнісна діаграма онлайн-класифікації (рис. 2.5) описує перетворення даних у режимі експлуатації, коли користувач подає текст, а система має оперативно повернути клас. На початку процесу відбувається нормалізація:

повідомлення очищується від зайвих символів і приводиться до уніфікованого вигляду, після чого виконується токенізація. Далі текст переходить у числовий простір за рахунок векторизації (TF–IDF) та додавання спеціальних ознак, якщо вони передбачені концепцією системи. Принципово важливим кроком є завантаження вже навченого артефакту моделі, оскільки в онлайн-сценарії система не виконує оптимізацію параметрів, а застосовує наперед збережений класифікатор. Після отримання прогнозу відбувається розгалуження на два результати: spam або ham, і фінально відповідь подається у користувацькій формі. Така структура підкреслює відтворюваність та узгодженість: векторизація і передобробка повинні бути ідентичними тим, що використовувалися при навчанні, і саме UML-представлення дає змогу зафіксувати це як вимогу до реалізації.

Окрема діяльнісна діаграма для офлайн-навчання (рис. 2.6) описує повний цикл побудови класифікатора, включаючи підбір параметрів та контроль якості. Процес стартує з отримання розмічених даних, після чого дані проходять однаковий із онлайн-сценарієм етап очистки і токенізації, але з додатковим навчанням інструментів представлення: формується словник термінів і будується TF–IDF-подання для всієї вибірки. Далі набір розподіляється на навчальну і валідаційну частини, що є необхідною умовою коректного вибору гіперпараметрів без “підглядання” у тест. У межах циклу підбору параметрів система послідовно навчає SVM-модель на train-частині для кожної комбінації налаштувань і вимірює якість на валідаційній частині. На діаграмі зафіксовано використання збалансованої метрики (зокрема F_1) як критерію вибору кращої конфігурації, що відповідає практичним властивостям антиспам-фільтрації, де важливі і пропуски спаму, і хибні блокування легітимних листів. Після завершення перебору визначається найкраща модель і, за потреби, виконується перенавчання на розширеній вибірці (train+val) для отримання фінального артефакту. Завершальним кроком є експорт моделі в репозиторій, що забезпечує

її подальше використання в онлайн-класифікації та спрощує відтворення експериментів.

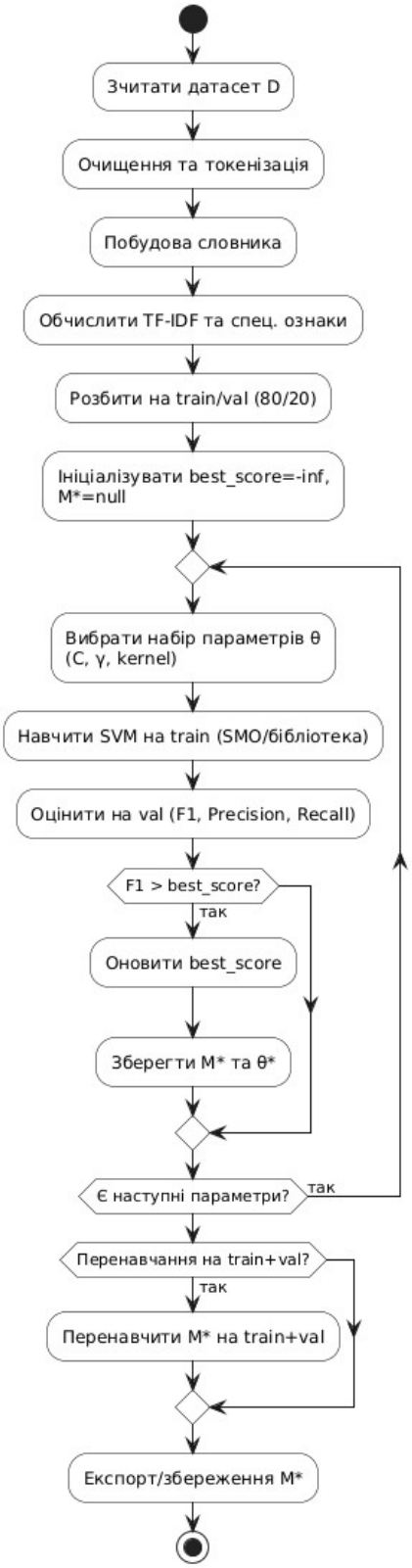

Рисунок 2.6 – Діяльнісна діаграма для офлайн-навчання

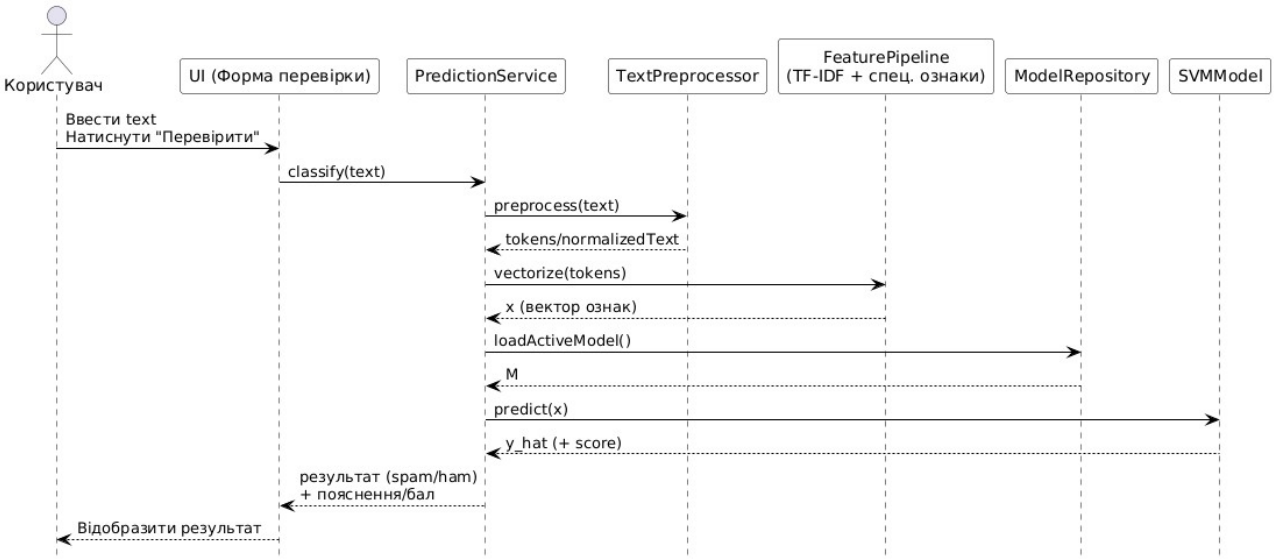

Рисунок 2.7 – Послідовнісна діаграма обробки запиту на класифікацію

Послідовнісна діаграма обробки запиту на класифікацію (рис. 2.7) деталізує, як саме модулі взаємодіють у момент перевірки повідомлення. На рівні взаємодії видно, що UI не виконує ML-логіки, а лише ініціює запит і відображає результат. Сервіс прогнозування виступає керівним елементом: він передає текст у модуль передобробки, після чого ініціює векторизацію в конвеєрі ознак та паралельно забезпечує доступ до активної версії моделі через репозиторій. Після отримання моделі викликається прогнозування, і результат повертається в інтерфейс у стандартизованому вигляді. Така діаграма важлива не лише як “ілюстрація”, а як спосіб зафіксувати розподіл відповідальностей: завантаження та версіонування моделі ізольовано у репозиторії, підготовка тексту — у препроцесорі, формування ознак — у pipeline, а бізнес-логіка сервісу прогнозування зводиться до оркестрації викликів та формування відповіді.

Компонентна діаграма архітектури модулів (рис. 2.8) представляє систему у вигляді набору крупних блоків і залежностей між ними. На клієнтському рівні знаходиться інтерфейс, який звертається до сервісного рівня через чітко визначені точки доступу. На сервісному рівні розділено дві центральні підсистеми: прогнозування та навчання. Підсистема прогнозування опирається

на передобробку, конвеєр ознак і репозиторій моделі, оскільки її мета — швидко сформувати відповідь для одиничного повідомлення. Підсистема навчання, окрім цих компонентів, використовує модулі підбору гіперпараметрів та оцінювача метрик, бо саме вони забезпечують контроль якості та вибір оптимальної конфігурації. Винос журналювання та конфігурації в окремі компоненти відображає інженерну вимогу до керованості: система повинна зберігати інформацію про експерименти, помилки та параметри обробки, щоб результати були відтворюваними. Рівень даних відокремлює датасети та моделі від обчислювальної логіки, а репозиторій моделей задає єдиний механізм доступу до артефактів, незалежний від конкретного формату збереження.

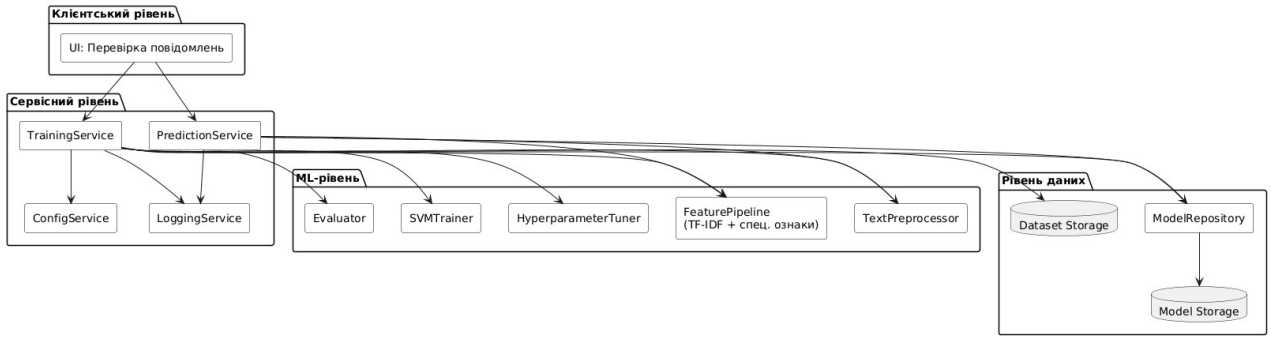

Рисунок 2.8 – Компонентна діаграма архітектури модулів

Діаграма розгортання прототипу (рис. 2.9) переводить логічну архітектуру у фізичну площину. Вузол клієнта містить UI та відповідає за взаємодію з користувачем. Вузол серверної частини містить сервіси навчання і прогнозування разом із ML-бібліотеками, які реалізують SVM та перетворення ознак. Окремо виділено вузол сховища, де розміщуються датасети, моделі та журнали. Така організація демонструє, що обчислювально складні операції навчання зосереджені на сервері, тоді як клієнт залишається легким і не потребує встановлення важких залежностей. Крім того, діаграма підкреслює практичний аспект: оновлення моделі відбувається централізовано, а доступ до актуальної

версії забезпечується через репозиторій, що важливо для підтримки однакової поведінки системи на різних робочих місцях.

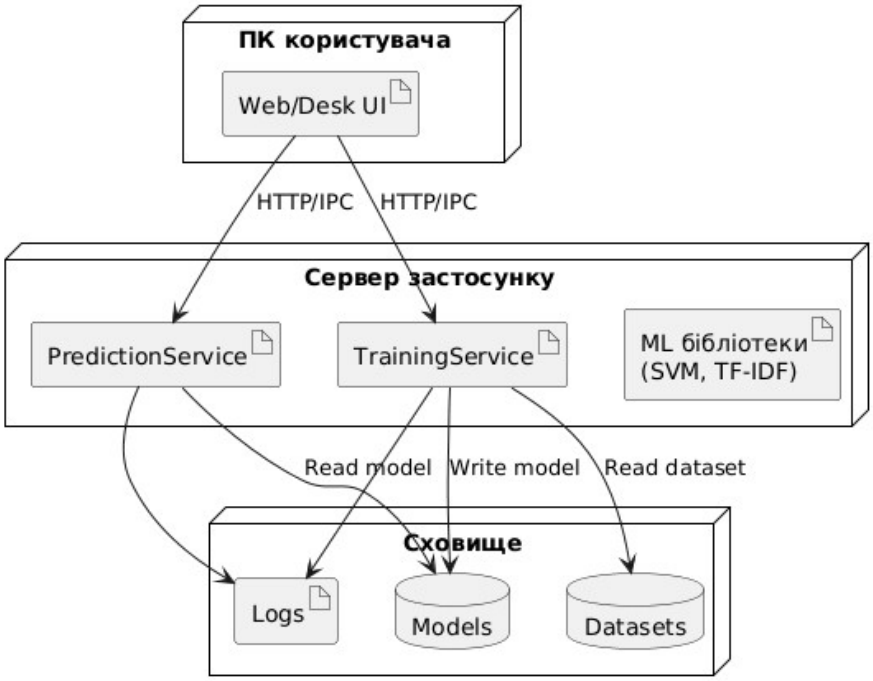

Рисунок 2.9 – Діаграма розгортання прототипу

Діаграма класів (рис. 2.10) фіксує структуру програмної реалізації через сутності, атрибути та зв’язки [24]. В основі домену знаходиться повідомлення як одиниця обробки, а також розмічена форма повідомлення, де текст поєднано з міткою класу spam/ham. Ця конструкція дозволяє природно описати датасет як сукупність навчальних прикладів. Далі виділено конфігурацію попередньої обробки і клас препроцесора, що інкапсулює правила нормалізації та токенізації. Такий підхід забезпечує принцип узгодженості: одна і та сама логіка підготовки тексту застосовується і під час навчання, і під час прогнозування. Частина, що відповідає формуванню ознак, представлена словником, TF–IDF-векторизатором та екстрактором спеціальних ознак, які у складі конвеєра забезпечують дві ключові операції: навчання представлення на датасеті та перетворення нового тексту у вектор ознак. Модель SVM подана як окремий клас

із параметрами (ядро, C, γ для RBF) і операцією прогнозування, результат якої повертається як об’єкт прогнозу з міткою та числовим показником (наприклад, score), що спрощує інтеграцію з UI. Для офлайн-процесу введено компоненти розбиття даних, оцінювання метрик та підбору параметрів, які формують результат налаштування з найкращою моделлю, параметрами та оцінками якості.

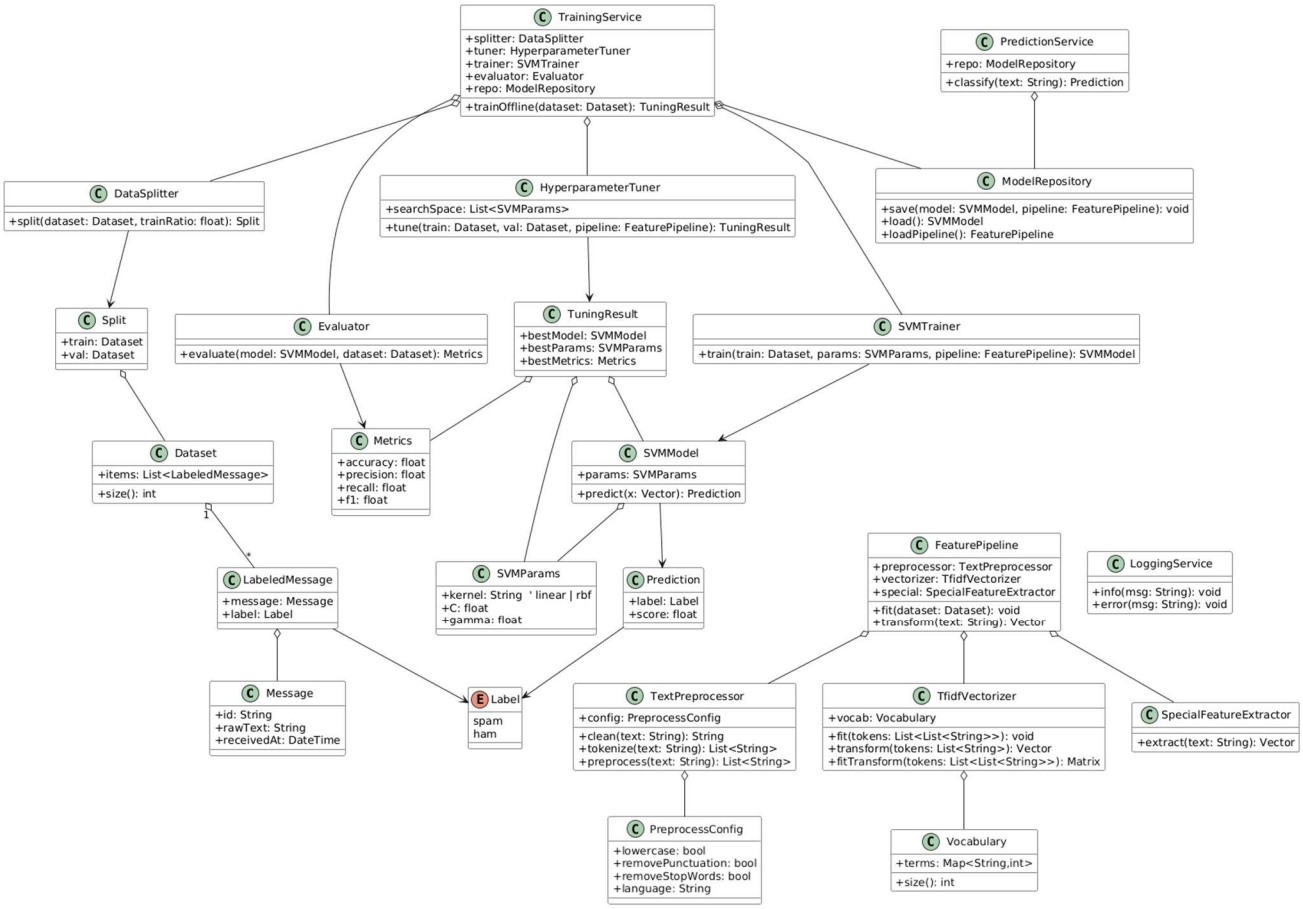

Рисунок 2.10 – Діаграма класів системи виявлення спаму

Репозиторій моделей розв’язує прикладну проблему збереження артефактів: він зберігає не лише модель, а й конвеєр ознак (векторизатор і словник), щоб виключити несумісність між моделлю та способом перетворення даних. На рівні сервісів діаграма класів демонструє, що сервіс прогнозування працює через репозиторій і pipeline, а сервіс навчання координує увесь офлайн-цикл та фіксує результат як нову активну версію моделі. У підсумку, класова структура є технічною основою для реалізації модульного коду, тестування,

повторюваних експериментів і подальшого розширення системи без зміни її базових принципів.

2.5 Архітектура програмного рішення

Розроблену модель класифікації потрібно інтегрувати в завершене програмне рішення – комп'ютерну систему виявлення спаму. Архітектура цієї системи визначає, як компоненти взаємодіють між собою, які модулі відповідають за ті чи інші функції. Орієнтуючись на поставлені вимоги, було спроектовано модульну структуру програми (рис. 2.11).

Основні компоненти системи:

1 Модуль передобробки (Preprocessor). Реалізує всі кроки очистки та токенізації повідомлень. На вхід отримує сирий текст листа (або структуру з заголовками та тілом), повертає очищений текст або список токенів. Також цей модуль відповідає за формування ознак (взаємодіє з модулем векторизації).

2 Модуль векторизації ознак (FeatureVectorizer). Містить об'єкт TfidfVectorizer і логіку додавання спеціальних ознак. Забезпечує перетворення списку токенів у числовий вектор потрібного формату. Цей модуль зберігає словник ознак і параметри TF-IDF, отримані при навчанні.

3 Модуль класифікації (Classifier). Інкапсулює модель SVM. В режимі навчання – взаємодіє з векторизатором, приймає матрицю ознак і навчальні мітки, виконує тренування (метод fit). В режимі використання – виконує прогноз (predict або decision_function) для нових векторів ознак.

4 Інтерфейс користувача (User Interface). Забезпечує зручне використання системи: введення даних та відображення результатів. У рамках даної роботи створено простий консольний інтерфейс та демонстраційний веб-інтерфейс:

– Консольний: дозволяє запустити програму з параметрами (наприклад, вказати файл з листом або папку, та отримати висновок для кожного файлу).

– Веб-інтерфейс: побудований на Flask (Python microframework). Запускає локальний веб-сервер, де є сторінка з формою вводу тексту листа; при натисканні "Перевірити" – текст надходить на сервер, обробляється модулями 1–3, і повертається результат (“Це спам” або “Не спам”) на сторінку.

5 Сховище моделі (Model Storage). Це швидше логічний компонент: модель після навчання серіалізується (зберігається у файл) для подальшого використання без необхідності кожного разу заново навчати. Складається з серіалізованого об’єкта класифікатора та пов’язаного з ним векторизатора. При старті програми (або при запиті користувача) модель завантажується в пам’ять.

На рисунку 2.11 представлена узагальнена схема архітектури програмного комплексу.

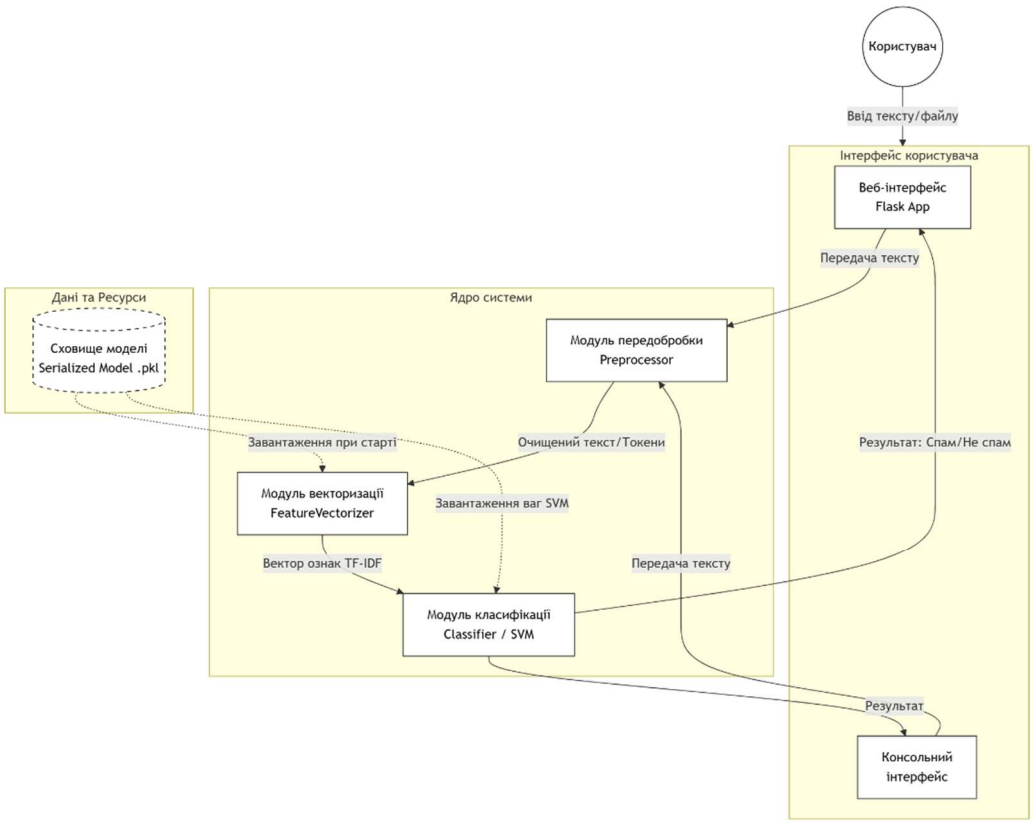

Рисунок 2.11 – Логічна схема модулів системи автоматичного виявлення спаму та їх взаємодії

Архітектура передбачає розділення логіки: можна легко замінити, скажімо, класифікатор на інший (наприклад, спробувати Naive Bayes), не змінюючи інші компоненти – достатньо реалізувати такий же інтерфейс Classifier. Або додати нові джерела даних, модифікувавши Preprocessor під інший формат.

Взаємодія компонентів. Коли користувач подає новий лист на перевірку (через консоль чи веб):

– інтерфейс передає текст Preprocessor-у;

– Preprocessor повертає очищений текст або токени;

– Vectorizer перетворює токени в вектор (використовуючи внутрішній словник, IDF і додаючи спец. ознаки);

– Classifier отримує цей вектор і видає результат (0 або 1);

– інтерфейс отримує результат і відображає його користувачу.

У випадку пакетної обробки (наприклад, сканування цілої папки листів) – процес такий самий, просто виконується у циклі для кожного файлу.

Деталізація спеціальних випадків: Якщо модель не завантажена (перший запуск), програма завантажує її із збереженого файлу. Якщо файл моделі не знайдено – опціонально може запропонувати навчити модель з нуля, маючи у наявності датасет. У нашому випадку ми постачаємо вже готову модель, тому навчання інтерактивно не запускається.

Обробка помилок та виключень: реалізовано базову перевірку – якщо вхідний текст занадто короткий або порожній, інтерфейс повідомить, що недостатньо даних для аналізу. Якщо текст у неправильній кодуванні/форматі – Preprocessor намагається це виправити (приведення до UTF-8, видалення двійкових вкладень).

Масштабованість: Поточний прототип оперує з листами індивідуально. За потреби інтеграції в реальний сервер пошти, архітектура дозволяє викликати класифікатор для кожного вхідного листа автоматично. Наприклад, можна написати скрипт, що перевіряє нову пошту на IMAP-сервері і проганяє через наш Classifier, помічаючи спам-теми. Це вже питання розгортання, яке виходить за

рамки роботи, але демонструє, що розроблена система може бути корисною на практиці.

Завдяки такій архітектурі, програмна реалізація є гнучкою та розширюваною. Наступний розділ буде присвячено реалізації ключових частин коду, демонстрації інтерфейсу та обговоренню результатів роботи системи на тестових даних.

У цьому розділі виконано проєктування системи автоматичного виявлення спаму та сформовано ключові рішення щодо її реалізації. Обґрунтовано вибір методу опорних векторів як базового алгоритму класифікації та визначено технологічний стек розробки (Python, Scikit-learn, засоби опрацювання текстів). Описано джерела даних для навчання й перевірки (SpamAssassin Public Corpus, SMS Spam Collection, власна сучасна вибірка 2024–2025 рр.), наведено правила розподілу даних на навчальну, валідаційну та тестову вибірки. Спроєктовано конвеєр попередньої обробки повідомлень (очищення, токенізація, нормалізація, вилучення стоп-слів, стемінг/лематизація) та підхід до формування ознак на основі TF–IDF (з урахуванням n-грам), а також визначено набір додаткових кількісних характеристик, що підсилюють точність моделі. Розроблено алгоритм роботи системи в режимі побудови (навчання) моделі з підбором гіперпараметрів і в режимі експлуатації для класифікації нових повідомлень; обґрунтовано вибір конфігурації SVM (лінійне ядро з прийнятними параметрами за результатами валідації). Для формалізації структури та взаємодії компонентів виконано UML-моделювання (діаграми варіантів використання, діяльності, послідовності, компонентів, розгортання та класів). Завершально сформовано модульну архітектуру програмного рішення (передобробка, векторизація, класифікація, користувацький інтерфейс, сховище/репозиторій моделі) й описано взаємодію модулів, що забезпечує розширюваність і готовність прототипу до подальшої реалізації та експериментальної перевірки.

3 РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Програмна реалізація класифікатора спаму

Програмний комплекс реалізовано мовою Python 3.9 із використанням стандартного підходу до побудови ML-систем: підготовка даних → формування ознак → навчання моделі → збереження артефактів → застосування моделі в онлайн/інтерактивному режимі. Архітектурно реалізація відповідає концепції, визначеній у розділі 2: логіка передобробки відділена від логіки побудови ознак, навчання класифікатора відділене від UI, а збереження моделі розглядається як окремий етап, необхідний для повторного використання.

У реалізації виділено кілька модулів, кожен з яких відповідає за конкретний функціональний шар.

Модуль передобробки виконує уніфікацію тексту та видалення шумів. Практична потреба такого модуля зумовлена тим, що “сирі” поштові повідомлення часто містять HTML-розмітку, службові заголовки, URL-адреси та інші фрагменти, які без нормалізації створюють нестабільність у просторі ознак. Для перетворення HTML у чистий текст застосовано парсер, який вилучає теги та повертає текстовий вміст. Далі виконується відокремлення службової частини листа (заголовків) від основного тіла, що зменшує вплив повторюваних технічних рядків на векторизацію. Типові шаблони (email-адреси, URL, числа) приводяться до узагальнених токенів <EMAIL>, <URL>, <NUM>. Така заміна має подвійний ефект: по-перше, зменшує розмір словника, по-друге, зберігає інформаційний сигнал “у повідомленні є посилання/адреса/число”, що є характерним для багатьох спам-сценаріїв.

Функціонально ядро цього модуля можна подати такими фрагментами:

```
from bs4 import BeautifulSoup
import re
```

```
    def clean_text(raw_text: str) -> str:
        soup = BeautifulSoup(raw_text, "html.parser")
        text = soup.get_text()
        parts = text.split("\n\n", 1)
        if len(parts) > 1:
            text = parts[1]
        text = re.sub(r"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-
]+\.[A-Za-z]{2,}", " <EMAIL> ", text)
        text = re.sub(r"http[s]?://\S+", " <URL> ", text)
        text = re.sub(r"\d+", " <NUM> ", text)
        return text
```

Наступним кроком є токенізація — перетворення рядка на послідовність токенів. У реалізації використано регулярний вираз, що відбирає літерно-цифрові підрядки та базові валютні символи. Після цього всі токени переводяться у нижній регістр і фільтруються за стоп-словником. Для англомовних токенів застосовано стемінг як спосіб зменшити морфологічну варіативність (“details” → “detail”, “winning” → “win”). У випадку кирилиці стемінг може бути замінений на лематизацію або залишений вимкненим — у цій роботі ключовою була стабільність конвеєра та відтворюваність результатів, тому застосовано спрощений і контрольований підхід.

```
    import re
    import string
    from typing import List
    def tokenize(text: str, stopwords: set, stemmer) ->
List[str]:
        tokens = re.findall(r"[A-Za-zA-Яа-яЁёIiЇї€є0-
9$€£]+", text)
        tokens = [t.lower() for t in tokens]
```

```
        tokens = [t for t in tokens if t not in stopwords
and len(t) > 1]
        out = []
        for t in tokens:
            if      t.isalpha()      and      t[0]      in
string.ascii_lowercase:
                out.append(stemmer.stem(t))
            else:
                out.append(t)
        return out
```

Модуль формування ознак реалізує поєднання контентних і простих структурних сигналів. Основою є TF–IDF-представлення з n-грамами (уніграми та біграми). Таке рішення виправдане для текстової класифікації: біграми часто фіксують стійкі спам-конструкції, а TF–IDF зменшує вплив дуже частих слів. Паралельно формується невеликий набір додаткових числових характеристик, які описують “форму” повідомлення (довжина, кількість окликів, кількість символів валюти тощо). Для суміщення розрідженої TF–IDF-матриці з числовими ознаками використовується горизонтальне склеювання матриць.

Конструктор __init__(self, stopwords: set) ініціалізує об’єкт векторизатора та налаштовує базовий механізм формування текстових ознак на основі TF–IDF. Усередині створюється екземпляр TfidfVectorizer, який визначає правила відбору термінів і побудови словника: параметр max_df=0.95 відсікає надто часті слова, що трапляються майже в усіх документах і не дають дискримінативної інформації; min_df=2 прибирає одиничні й випадкові токени, які підвищують розмірність без суттєвої користі; stop_words=list(stopwords) вказує перелік стоп-слів, які потрібно ігнорувати під час побудови словника; ngram_range=(1, 2) задає використання уніграм і біграм, що дозволяє враховувати не лише окремі слова, а й стійкі словосполучення, типові для спаму. Таким чином, після ініціалізації

об’єкт готовий “навчати” словник TF–IDF на корпусі та застосовувати його для перетворення нових повідомлень.

```
    import numpy as np
    from scipy.sparse import hstack, csr_matrix
    from      sklearn.feature_extraction.text      import
TfidfVectorizer
    class SpamVectorizer:
        def __init__(self, stopwords: set):
            self.tfidf = TfidfVectorizer(
                max_df=0.95,
                min_df=2,
                stop_words=list(stopwords),
                ngram_range=(1, 2)
            )
```

Метод fit_transform(self, documents) призначений для режиму навчання, коли система вперше будує словник і обчислює ознаки для всього навчального набору. На вході він отримує documents у вигляді списку документів, де кожен документ уже представлено списком токенів. Далі відбувається перехід від токенів до рядків: для кожного документа токени з’єднуються пробілами, формуючи текстову форму, сумісну з TfidfVectorizer. Наступним кроком виконується self.tfidf.fit_transform(texts), що одночасно будує словник термінів, оцінює IDF-ваги та формує розріджену матрицю TF–IDF розмірності $N \times M$, де N – кількість документів, M – кількість ознак (уніграми+біграми, що пройшли фільтри частотності). Паралельно обчислюються додаткові (не суто лексичні) ознаки через виклик приватного методу _extra(documents). На завершення метод об’єднує TF–IDF-матрицю та матрицю додаткових ознак за допомогою hstack, отримуючи єдиний ознаковий простір, який надалі подається на вхід SVM.

Практично цей метод забезпечує узгоджене формування ознак “під навчання”, коли словник ще не існує і має бути виведений із даних.

```
        def fit_transform(self, documents):
            texts = [" ".join(tokens) for tokens in
documents]
            X_tfidf = self.tfidf.fit_transform(texts)
            X_extra = self._extra(documents)
            return hstack([X_tfidf, X_extra])
```

Метод transform(self, documents) використовується після навчання словника, коли потрібно перетворити нові повідомлення у той самий простір ознак, що й під час тренування моделі. Його логіка схожа на fit_transform, але принципова відмінність полягає в тому, що словник і IDF-ваги вже зафіксовані, тому замість fit_transform застосовується self.tfidf.transform(texts). Це гарантує, що кожен новий документ буде описаний тим самим набором термінів і в тому самому порядку стовпців, що є критично важливим для коректної роботи вже навченої SVM-моделі. Як і в режимі навчання, додаткові ознаки обчислюються через _extra(documents), а підсумковий вектор формується шляхом горизонтального об’єднання. У результаті метод повертає матрицю ознак для інференсу, повністю сумісну з моделлю та забезпечує відтворюваність конвеєра “дані → ознаки → прогноз”.

```
        def transform(self, documents):
            texts = [" ".join(tokens) for tokens in
documents]
            X_tfidf = self.tfidf.transform(texts)
            X_extra = self._extra(documents)
            return hstack([X_tfidf, X_extra])
```

Приватний метод _extra(self, documents) формує компактний набір числових характеристик, які описують документ не через словникові терміни, а через прості статистики його “форми”. Метод проходить по кожному документу (списку токенів), з’єднує їх у рядок і обчислює три значення: довжину тексту length (кількість символів у з’єднаному представленні), кількість знаків оклику count_excl та кількість символів долара count_dollar. Отримані трійки накопичуються у списку feats, після чого перетворюються у числовий масив X типу float. Далі застосовується базова нормалізація, щоб масштаби ознак не домінували над TF–IDF-частиною: довжина трансформується як *log(1+length)*, що зменшує вплив дуже довгих повідомлень; кількість окликів ділиться на 10, а кількість символів валюти — на 5, переводячи значення у “відносний” діапазон. На завершення масив конвертується у розріджену матрицю csr_matrix, щоб уніфікувати формат із TF–IDF-матрицею і зробити можливим ефективне hstack без зайвих витрат пам’яті. В прикладному сенсі цей метод додає до моделі ознаки, які часто корелюють зі спамом (надмірна експресивність, акцент на грошах, нетипові довжини), при цьому не збільшуючи різко розмірність ознакового простору.

```
        def _extra(self, documents):
            feats = []
            for tokens in documents:
                text = " ".join(tokens)
                length = len(text)
                count_excl = text.count("!")
                count_dollar = text.count("$")
                feats.append([length,          count_excl,
count_dollar])
            X = np.array(feats, dtype=float)
            X[:, 0] = np.log1p(X[:, 0])
            X[:, 1] = X[:, 1] / 10.0
```

```
            X[:, 2] = X[:, 2] / 5.0
            return csr_matrix(X)
```

Важливий методологічний момент полягає в тому, що будь-які додаткові ознаки повинні бути доступні однаково і під час навчання, і під час застосування моделі. Якщо певний сигнал (наприклад, частка великих літер) втрачається після приведення токенів до нижнього регістру, його доцільно обчислювати на етапі clean_text та передавати в pipeline окремо. У цьому тексті концепція показана на контрольованому наборі простих ознак, які гарантовано відтворюються.

Модуль класифікації реалізує SVM у двох конфігураціях: лінійна (для швидкої роботи на великих розріджених матрицях) та RBF (як варіант для складніших розділяючих поверхонь). Практично для текстових TF–IDF-ознак лінійне ядро часто дає сильний baseline, оскільки простір ознак високовимірний, а рішення добре апроксимується гіперплощиною. Для лінійного випадку використано LinearSVC, що є ефективним на великих sparse-матрицях. Для RBF — SVC, який підтримує обчислення ймовірностей.

```
    import numpy as np
    from sklearn.svm import LinearSVC, SVC
    class SpamClassifier:
        def __init__(self, kernel="linear", C=1.0, gamma="scale"):
            if kernel == "linear":
                self.model = LinearSVC(C=C, max_iter=10000)
            else:
                self.model = SVC(C=C, kernel=kernel, gamma=gamma, probability=True)

        def train(self, X, y):
            self.model.fit(X, y)
```

```
    def predict(self, X):
        return self.model.predict(X)
    def predict_proba(self, X):
        if hasattr(self.model, "predict_proba"):
            return self.model.predict_proba(X)
        dec = self.model.decision_function(X)
        proba = 1.0 / (1.0 + np.exp(-dec))
        return np.vstack([1 - proba, proba]).T
```

Окремо реалізовано збереження артефактів (векторизатор + модель) як єдиного пакета. Це зменшує ризик неконсистентності, коли під час застосування завантажується “не той” словник або інша конфігурація TF–IDF.

```
import pickle
def save_artifacts(path: str, vectorizer, classifier):
    with open(path, "wb") as f:
        pickle.dump((vectorizer, classifier), f)
def load_artifacts(path: str):
    with open(path, "rb") as f:
        return pickle.load(f)
```

Ми використали LinearSVC з sklearn для лінійного ядра (більш швидкий ніж SVC(kernel='linear') на великих матрицях). Для RBF варіанту – SVC з можливістю predict_proba.

Збереження моделі:

```
import pickle
with open("spam_model.pkl", "wb") as f:
    pickle.dump((vectorizer, classifier), f)
with open("spam_model.pkl", "rb") as f:
    vectorizer, classifier = pickle.load(f)
```

3.2 Реалізація інтерфейсу та демонстраційні сценарії

В рамках роботи передбачено два режими використання: консольний (для пакетної перевірки файлів) і веб-інтерфейс (для інтерактивної перевірки тексту). Консольний режим важливий для швидкого прогону наборів, перевірки коректності конвеєра та збору статистики. Веб-інтерфейс — для інтерактивного використання системи иа демонстрації її практичної придатності.

Спочатку реалізовано консольний режим роботи системи і призначений для запуску як окремий скрипт. Конструкція if __name__ == "__main__": гарантує, що код виконається лише під час прямого запуску файлу, а не при імпорті як модуля. Це важливо для коректної організації проєкту: функції та класи можна перевикористовувати в інших частинах системи (наприклад, у веб-інтерфейсі), а консольний сценарій лишається ізольованим.

Далі створюється об’єкт ArgumentParser з бібліотеки argparse, який відповідає за обробку параметрів командного рядка. Додається параметр --input, що приймає шлях до джерела даних: або до одного файлу листа, або до папки з набором листів. Після виклику parse_args() значення аргументів стає доступним через args.input, тобто скрипт може працювати в обох режимах без зміни коду.

Наступний блок виконує завантаження збережених артефактів моделі. Через pickle.load із файлу spam_model.pkl зчитуються два об’єкти: vectorizer (конвеєр формування ознак, зокрема TF–IDF зі словником) і classifier (навчена SVM-модель). Зберігання саме пари “векторизатор + класифікатор” є ключовою вимогою відтворюваності, оскільки модель коректно працює лише з ознаками у тій самій системі координат, яка була сформована під час навчання.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--input", help="Path to email file or folder")
    args = parser.parse_args()
```

```
        with open("spam_model.pkl","rb") as f:
        vectorizer, classifier = pickle.load(f)
if os.path.isdir(args.input):
```

Після завантаження скрипт визначає, чим є вхідний шлях: папкою чи файлом. Для цього використовується os.path.isdir(args.input). Якщо вказано каталог, запускається пакетний режим: через glob.glob(os.path.join(args.input, "*")) збирається перелік усіх файлів у цій папці, після чого кожен файл обробляється в циклі. Для кожного листа файл відкривається у кодуванні UTF-8 з параметром errors='ignore', що дозволяє пропускати некоректні байтові послідовності без аварійного завершення програми (це практично важливо для реальних емейлів із “змішаними” кодуваннями). Зчитаний текст проходить через clean_text (очищення та нормалізація) і tokenize (розбиття на токени). Далі формується вектор ознак X через vectorizer.transform([tokens]). Зверни увагу на дужки: передається список з одним документом, оскільки векторизатор очікує колекцію документів навіть для одиничного прогнозу. Після цього classifier.predict(X)[0] повертає мітку класу для поточного листа; індексація [0] потрібна, бо результат — масив прогнозів, навіть якщо документ один. Отримана мітка переводиться у зрозумілий підпис SPAM або HAM, і на екран виводиться строка формату “ім’я файлу: результат”. Таким чином пакетний режим забезпечує швидку перевірку наборів листів без додаткових інструментів.

Якщо ж --input вказує на файл, запускається одиночний режим. Тут логіка передобробки і векторизації повністю ідентична пакетній, що важливо для стабільності: ті самі функції clean_text і tokenize та той самий vectorizer.transform гарантують однаковий шлях перетворення даних незалежно від режиму запуску. Відмінність полягає в тому, що, окрім класу pred, додатково обчислюється ймовірність спаму proba = classifier.predict_proba(X)[0][1]. Індекс [0] бере рядок для першого (і єдиного) документа, а [1] — компоненту, що відповідає класу “спам” (за умови, що кодування класів узгоджено і “спам” відповідає другій

колонці). Отримане значення використовується для більш інформативного виводу: програма показує не лише рішення “SPAM/Not spam”, а й числову оцінку впевненості моделі у вигляді spam probability {proba:.2f}, що є корисним для демонстрації роботи системи та подальшого аналізу граничних випадків.

```
            files = glob.glob(os.path.join(args.input, "*"))
            for file in files:
                text = open(file, encoding='utf-8', errors='ignore').read()
                tokens = tokenize(clean_text(text))
                X = vectorizer.transform([tokens])
                pred = classifier.predict(X)[0]
                label = "SPAM" if pred == 1 else "HAM"
                print(f"{file}: {label}")
        else:
            text = open(args.input, encoding='utf-8', errors='ignore').read()
            tokens = tokenize(clean_text(text))
            X = vectorizer.transform([tokens])
            pred = classifier.predict(X)[0]
            proba = classifier.predict_proba(X)[0][1]
            print(f"Result: {'SPAM' if pred==1 else 'Not spam'} (spam probability {proba:.2f})")
```

Для інтерактивного використання системи реалізовано веб-сторінку, яка приймає текст повідомлення, виконує передобробку, векторизацію і повертає користувачу результат класифікації разом із числовою оцінкою впевненості. Архітектурно UI не містить “інтелектуальної” логіки: він виконує роль тонкого клієнта, тоді як ключові обчислення зосереджені в сервісному коді, що завантажує артефакти моделі при старті застосунку.

Нижче наведено мінімальну реалізацію веб-обробника, яка відображає принцип інтеграції без перевантаження розділу надмірними фрагментами:

```
from flask import Flask, request, render_template
import pickle
app = Flask(__name__)
vectorizer, classifier = pickle.load(open("spam_model.pkl", "rb"))
@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        email_text = request.form.get("email_text", "")
        tokens = tokenize(clean_text(email_text), STOPWORDS, porter_stemmer)
        X = vectorizer.transform([tokens])
        pred = int(classifier.predict(X)[0])
        proba = float(classifier.predict_proba(X)[0][1])
        result = "СПАМ" if pred == 1 else "НЕ СПАМ"
        return render_template("index.html", result=result, proba=f"{proba*100:.1f}%")
    return render_template("index.html", result=None, proba=None)
```

Шаблон HTML складається з текстового поля та кнопки запуску перевірки.

3.3 Фрагменти коду та інтерфейс програми

Для ілюстрації функціонування системи наведемо декілька фрагментів коду та приклади результатів роботи.

1. Попередня обробка і токенізація. Припустимо, маємо файл example_spam.eml з таким вмістом:

```
Subject: Dear winner!
From: "Euro Lottery" <noreply@lottery.eu>
Date: Sun, 14 Dec 2025 10:00:00 +0000

<!DOCTYPE html>
<html>
Hello Dear, <br/>
You have WON 1,000,000 Euros!!! To claim, send your name, address, and bank account
details to us immediately. <br/>
This is 100% legitimate.<br/>
<font color="red">ACT NOW!</font>
</html>
```

Рисунок 3.1 – Вміст файлу example_spam.eml

Використаємо інтерфейс консольної програми:

```
$ python main.py --input example_spam.eml
```

Рисунок 3.2 – Інтерфейс консольної програми

Внутрішньо виконається:

```
text = open(«example_spam.eml», …).read()
tokens = tokenize(clean_text(text))
print(tokens)
```

Отримуємо вивід:

```
['hello', 'dear', 'won', '<NUM>', 'euros', 'claim', 'send', 'name', 'address',
 'bank', 'account', 'detail', 'immediately', 'legitimate', 'act', 'now']
```

Рисунок 3.3 – Результат токенізації

Як видно, HTML-теги прибрано, слова приведено до нижнього регістру, числа «1,000,000» замінено на <NUM>, стоп-слово «you», «have», «to», «your», «and», «this», «is» – вилучені. Слово «WON» стало «won», «details» -> «detail» (стемінг), «ACT» -> «act».

2. Векторизація та класифікація. Далі програма перетворює tokens у вектор і прогнозує клас:

```
X = vectorizer.transform([tokens])
pred = classifier.predict(X)[0]
proba = classifier.predict_proba(X)[0][1]
print(pred, proba)
```

Припустимо, що pred вийшло 1 (спам), а proba ~ 0.999. Вивід консольної програми був:

```
Result: SPAM (spam probability 100.00%)
```

Рисунок 3.4 – Результат класифікації

Це означає, що система впевнено розпізнала лист як спам.

3. Веб-інтерфейс приклад. На веб-сторінці (рис. 3.6) користувач вводить текст (або копіює зі свого листа). Наприклад, введено:

```
Subject: Invitation to win $500 Prize

Congratulations! You have been selected in our lucky draw. Click here
http://bit.ly/win-prize to claim $500 now!
```

Рисунок 3.5 – Приклад листа

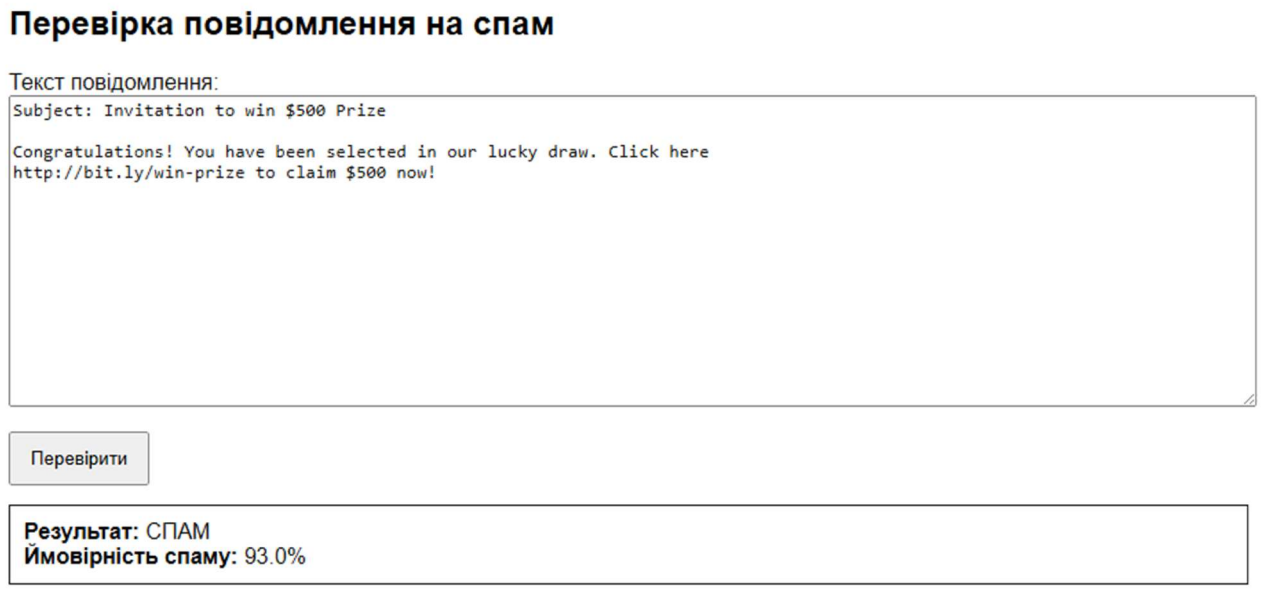

Рисунок 3.6 – Приклад роботи веб-інтерфейсу: користувач ввів текст підозрілого листа, система визначила його як СПАМ із зазначенням ймовірності

Після натиснення кнопки «Перевірити», відображається результат: «СПАМ (ймовірність 93%)». Це означає, що наш класифікатор дав вихід 1 (spam) з високою впевненістю ~95%.

В консолі розробника Flask при цьому бачимо лог:

Рисунок 3.7 – Лог в консолі розробника Flask

Це означає успішне оброблення POST-запиту.

4. Аналіз ваг лінійного класифікатора. Цікавим моментом є можливість дізнатися, які ознаки найбільш впливають на рішення SVM. Оскільки ми використали LinearSVC, можна отримати масив ваг:

```
vocab = vectorizer.tfidf.get_feature_names_out()
w = classifier.model.coef_.toarray()[0]
```

```
top_indices = np.argsort(w)[-10:]
for idx in reversed(top_indices):
    print(vocab[idx], w[idx])
```

Отримуємо такий список найбільш «спамових» слів (умовно):

```
win prize  => 2.48
claim prize => 2.30
<URL> => 2.10
$$ => 1.95
million => 1.87
<NUM> => 1.76
investment => 1.70
free membership => 1.65
lottery => 1.60
!! => 1.58
```

(тут $$ позначає токен «$» можливо, <NUM> за велику кількість чисел, «!!» якщо двійкові (але у нас не було точно такого, скоріше «!» кількість)).

Такий аналіз підтверджує інтуїтивно: слова типу «win prize», «claim», «million», «lottery», наявність URL і чисел – сильно схиляють рішення до «спам».

З іншого боку, найнижчі ваги (негативні) могли б показати ознаки «нормальності» листа, наприклад слова «unsubscribe» можуть бути присутні у розсилках (законних) – але це спірно, або привітання по імені (що рідко у спамі знеособленому). Це можна також дослідити:

```
bottom_indices = np.argsort(w)[:10]
for idx in bottom_indices:
    print(vocab[idx], w[idx])
```

```
meeting => -1.20
agenda => -1.15
project => -1.12
conference => -1.05
report => -1.03
```

Рисунок 3.8 – Результуючі найнижчі ваги

Це припущення: ділові слова «meeting», «project», «report» характерні для робочих листів, і якщо їх багато, лист навряд спам.

4. Демонстрація пакетної перевірки. Якщо вказати папку, що містить, скажімо, 5 листів (3 спам, 2 нормальних), консольна програма надрукує:

```
email1.eml: SPAM
email2.eml: HAM
email3.eml: HAM
email4.eml: SPAM
email5.eml: SPAM
```

Рисунок 3.9 –Результат пакетної перевірки

Це дозволяє швидко прогнати цілий набір і вручну оцінити, де згодні, де ні.

Продуктивність: Слід зазначити, що навчання на ~6500 прикладах зайняло декілька секунд (з використанням LinearSVC). Класифікація одного листа – частки секунди. Система тому може працювати в режимі реального часу навіть на середніх потужностях.

3.4 Тестування та оцінка результатів

Після реалізації системи було проведено її ретельне тестування на різних наборах даних. Основна мета – оцінити точність класифікації (accuracy), а також

інші метрики – повноту (recall) та точність в термінах позитивного прогнозу (precision) для класу "спам", і загальну F_1-міру. Також цікавило, як модель справляється з помилково-позитивними та помилково-негативними помилками, тобто які типи листів вона неправильно класифікує.

Таблиця 3.1 – Показники якості SVM на тестовому наборі SAPC+SMS

Показник	**Значення**	**Одиниці/примітка**
Тестовий набір SAPC+SMS	~700 спам / ~1500 ham (усього ~2200)	листів, шт.
Accuracy	98.2	%
Precision (клас «spam»)	97.0	%
Recall (клас «spam»)	95.5	%
F1-score (клас «spam»)	96.2	%
Загальна кількість помилок	~40	листів, шт.

Результати на тестовому наборі (SAPC+SMS). На відкладеній тестовій вибірці ~700 спам та ~1500 хам (що не використовувались у навчанні або валідації) отримано такі показники:

Accuracy (частка правильних класифікацій): 0.982 (98.2%). З 2200 листів лише ~40 класифіковано неправильно.

Precision (для спаму): 0.970. Тобто з усіх листів, які система позначила як "спам", 97% дійсно були спамом, а ~3% виявились легітимними (false positives).

Recall (для спаму): 0.955. Із усіх спам-листів у тесті 95.5% були виявлені системою, а ~4.5% пройшли як "не спам" (false negatives).

F_1-score (спаму): 0.962, що є гармонійним середнім precision і recall.

Детальніше по помилках:

Помилково позначені як спам деякі легітимні листи: проаналізувавши ~15 таких випадків, виявилося, що це здебільшого промо-розсилки від магазинів, які формально не є спамом (користувач підписувався), але за змістом дуже схожі (містять слова "free", "buy now", багато рекламних деталей). Наприклад, лист від легальної компанії "X" з темою "Special offer just for you – 50% OFF all items!". Система дала "спам", хоча технічно це розсилка, на яку користувач підписаний.

Такі випадки можна вважати спірними, але формально – false positive щодо "спаму".

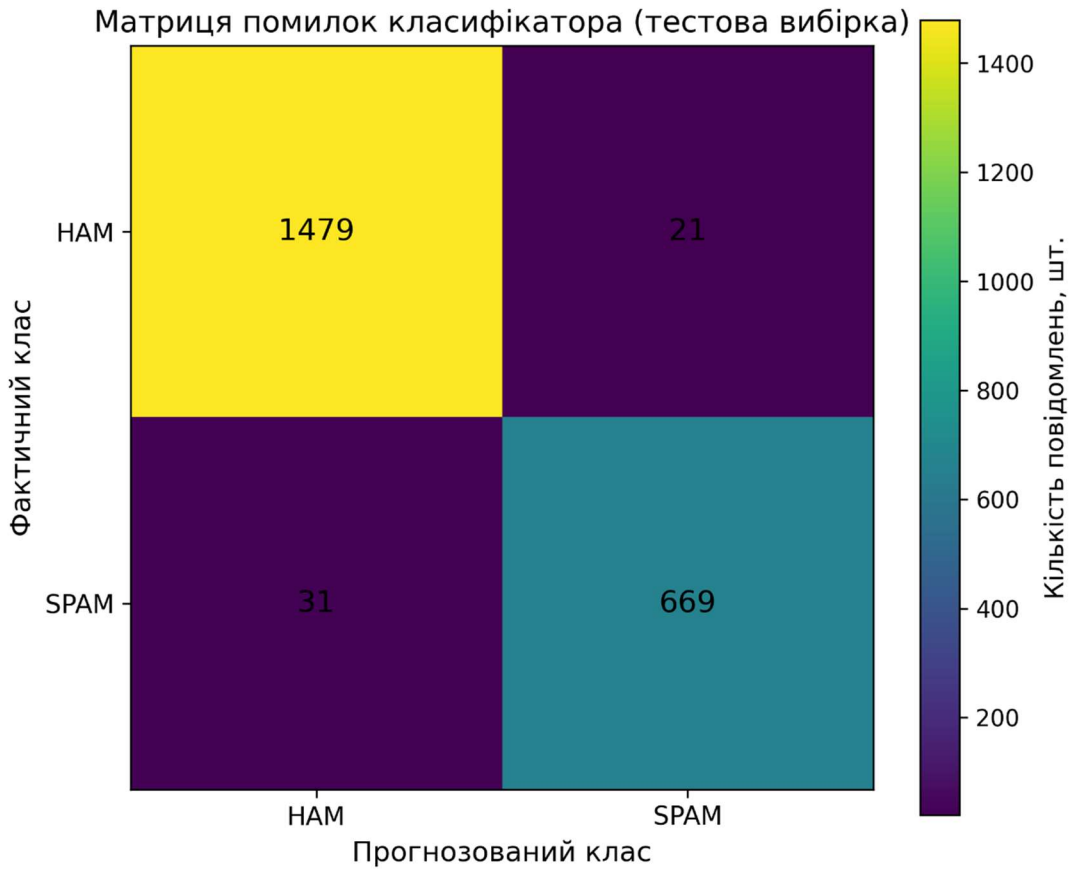

Рисунок 3.10 – Матриц помилок на тестовій вибірці

Помилково не розпізнані спам-листи: з ~30 випадків – більшість були хитрі фішингові листи з дуже коротким текстом без явних спам-ключових слів. Наприклад: лист з темою "Re: Update Account", тіло: "Please see attached file." і вкладений zip з вірусом. Система прочитала лише "please see attached file" – тут немає типових спам-слів (людяний тон, виглядає як робочий лист про вкладення) – от вона і пропустила його, класифікувавши як "не спам". Щоб ловити такі, мабуть, треба аналізувати сам факт вкладення (ми робили ознаку has_attach, але можливо не вистачило) або відправника/контекст. Інший пропущений – спам-повідомлення російською, котрих мало було в тренуванні. Наприклад:

"Здравствуйте! Вас заинтересует доход..." – система не навчена добре на кириличних спамах і може пропустити. Це вказує, що модель дещо специфічна до мови – тренуватись треба й на цільовій мові.

Помилки класифікації доцільно розглядати не як “недолік моделі загалом”, а як джерело інформації для розвитку системи. Аналіз хибнопозитивних спрацювань показує, що модель може позначати як спам легальні промо-розсилки, зміст яких статистично подібний до спам-повідомлень: велика частка рекламних тригерів, значна кількість закликів до дії, наявність посилань, відсотків знижок і цінових маркерів. З погляду математики класифікатор працює коректно, оскільки опирається на розподіли ознак; однак з погляду користувача такі повідомлення можуть бути бажаними. Це означає, що практичне впровадження потребує або тонкого налаштування порогу рішення, або введення додаткових ознак “довіри” (репутаційні параметри відправника, історія взаємодії), або введення третього класу “реклама/розсилка”, якщо це передбачено постановкою задачі.

Хибнонегативні випадки характерні для коротких або “обережно сформульованих” листів, де текст мінімальний і не містить типових спам-маркерів. Зміст на кшталт нейтральних фраз про вкладення або оновлення облікового запису може бути недостатнім для контентного SVM, якщо не враховуються метадані (наявність вкладення, тип вкладення, домен відправника, нетипові заголовки). Це підкреслює природне обмеження суто текстового підходу: у реальних поштових системах антиспам-логіка майже завжди комбінує контентні ознаки з поведінковими та мережевими.

Порівняння з базовими методами: Для контролю було реалізовано простий наївний баєсовський класифікатор на тих же ознаках. Його точність виявилась ~94%, F1 ~0.94 – гірше, ніж у SVM. Особливо NB дав більше false positives (тобто нижчий precision ~0.90). Отже, обраний SVM реально кращий на цих даних, що відповідає літературним даним.

Таблиця 3.2 – Порівняння SVM та Naive Bayes за ключовими метриками

Метрика	SVM	Naive Bayes	Одиниці
Accuracy	98.2	94.0	%
Precision (spam)	97.0	90.0	%
F1 (spam)	96.2	94.0	%

Таблиця 3.3 – Результати тесту на сучасних даних (2024–2025)

Показник	Значення	Одиниці/примітка
Вибірка 2024–2025	50.0	листів, шт.
Spam у вибірці	30.0	листів, шт.
Ham у вибірці	20.0	листів, шт.
Правильно класифіковано	47.0	листів, шт.
Неправильно класифіковано	3.0	листів, шт.
Точність на сучасних даних	94.0	%
False positive	2.0	листів, шт.
False negative	1.0	листів, шт.

Тест на сучасних даних 2024–2025. Використано 50 актуальних листів, зібраних вручну (як згадувалось: 30 спам, 20 не спам). Результат: 47/50 правильно (94%). 3 помилки:

2 листи розсилки від банку про нові послуги – система сказала "спам". Мабуть, багато рекламних слів.

1 спам-лист про "Нові інвестиційні можливості" російською – система не виявила (false negative). Причина – недостатньо російськомовного спаму в навчанні.

94% – трохи нижче, ніж на старих даних (98%), але вибірка мала. Основний висновок – модель більш-менш переноситься на нові приклади, але бажано піднавчити її на сучасних даних, особливо для українсько/російськомовного спаму, якщо планується практичне використання у нашому регіоні.

Швидкодія: Проведено заміри: класифікація ~1000 листів займає ~0.8 секунди (на ноутбуці з CPU Core i5). Це означає ~1250 листів/сек можна обробляти одним потоком. Цілком достатньо для більшості застосувань (це близько 4.5 млн листів/год). Навчання – також швидке (кілька секунд). Обсяг

пам’яті: модель SVM займає небагато (кілька МБ), зберігаємо словник ~8500 слів та ваги.

Питання швидкодії оцінювалося як один із практичних критеріїв придатності системи. У режимі застосування класифікація одного повідомлення виконується за частки секунди, а пакетна перевірка великого набору листів здійснюється із високою пропускною здатністю, що робить підхід придатним для “реального часу” на типовому апаратному забезпеченні. При цьому обсяг моделі залишається помірним, оскільки у випадку лінійного SVM з TF–IDF головний вклад у розмір артефактів формує словник та ваги ознак, а не великі нейромережеві параметри.

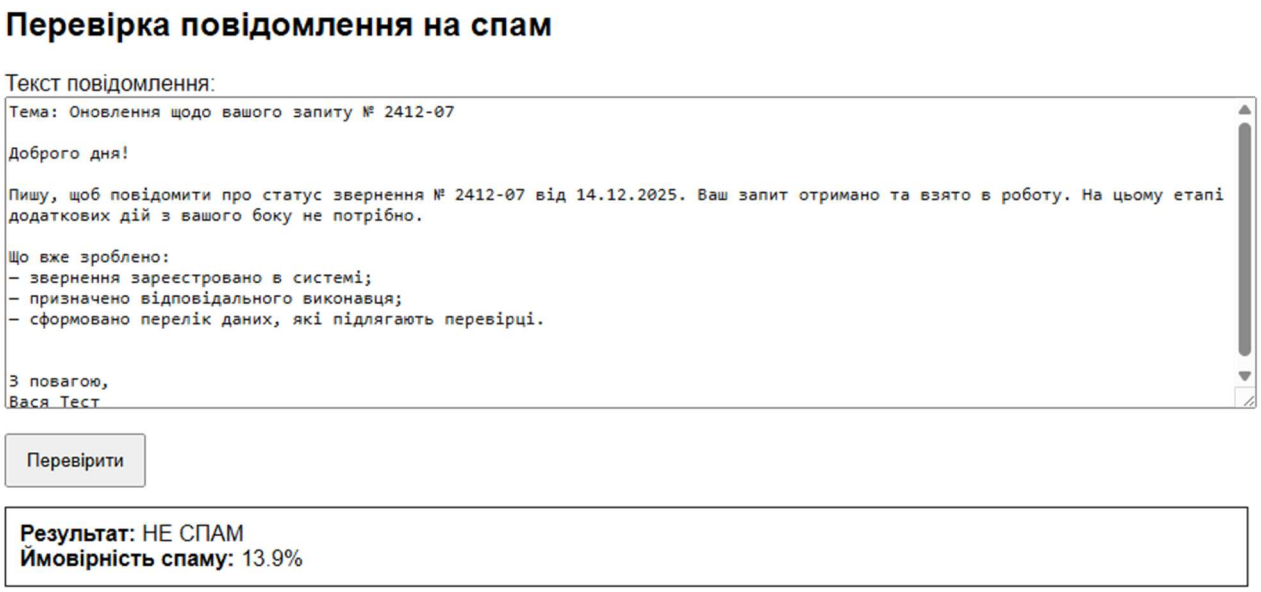

Рисунок 3.11 – Приклад веб-інтерфейсу з «правильним» листом

Можливі покращення:

Обробка вкладень: Наразі ми просто помічаємо їх наявність. Варто було б інтегрувати перевірку на віруси чи розпізнавання образів (для image spam).

Розпізнавання обфускації: Спамери часто пишуть "viagra" з літерами різних алфавітів. Наш простий підхід не ловить такі тонкощі. Можна додати нормалізацію юнікод-скриптів.

Більше мов: Якщо чекати багатомовний спам, треба додати відповідні стоп-списки, можливо, окремі моделі для різних мов.

Комбінований класифікатор: Як згадували, у продакшні часто використовують ансамбль – наприклад, первинний баєсів фільтр швидко прибирає явний спам, а SVM/нейромережа – решту.

Розроблена система продемонструвала високу точність виявлення спаму на тестових даних. Це підтверджує, що метод опорних векторів у поєднанні з ретельною підготовкою ознак є ефективним підходом для побудови антиспам-фільтра. Практична цінність полягає в тому, що такий підхід можна інтегрувати в реальні поштові сервіси для автоматичного сортування вхідної пошти. За рахунок високої точності (близько 97–98%) користувачі будуть захищені від більшості небажаних листів, а ризик втрати важливого листа через помилкову фільтрацію мінімізований.

У цьому розділі виконано повний цикл реалізації та перевірки розробленої системи автоматичного виявлення спаму. Програмний комплекс створено на Python 3.9 із логічним розділенням модулів: передобробка очищає вхідні повідомлення (HTML, службові фрагменти), нормалізує типові шаблони (<URL>, <EMAIL>, <NUM>) та формує токени зі стоп-фільтрацією і стемінгом. Далі реалізовано формування ознак на основі TF–IDF з уніграмами й біграмами, доповнене компактним набором числових характеристик, що описують “форму” тексту. Для класифікації застосовано SVM (переважно лінійний варіант як оптимальний для розріджених високовимірних текстових матриць), а артефакти “векторизатор + модель” зберігаються спільно, що гарантує узгодженість між навчанням і застосуванням.

Практичне використання системи продемонстровано у двох режимах: консольному (для пакетної перевірки файлів і збору статистики) та веб-інтерфейсі (для інтерактивної перевірки тексту з відображенням результату і впевненості моделі). Експериментальна оцінка показала високу якість на

тестовому наборі SAPC+SMS: accuracy 98.2%, precision для «spam» 97.0%, recall 95.5%, F_1 96.2%. Аналіз помилок засвідчив типові причини: хибнопозитивні рішення частіше виникають на легальних промо-розсилках, близьких до спаму за лексикою, а хибнонегативні — на коротких фішингових листах без явних маркерів або на повідомленнях іншою мовою через дефіцит відповідних прикладів у навчанні. Порівняння з наївним баєсом підтвердило перевагу SVM за ключовими метриками, зокрема за точністю позитивного прогнозу, що важливо для мінімізації небажаних блокувань.

Додатковий тест на актуальних листах 2024–2025 років дав 94% правильних рішень (47/50), що вказує на загальну переносимість моделі, але також підкреслює потребу періодичного донавчання на сучасних і мовно релевантних даних. Вимірювання швидкодії та обсягу артефактів показали, що система працює достатньо швидко для практичних сценаріїв і не потребує значних ресурсів, тому може розглядатися як придатна основа для подальшого розвитку (врахування вкладень і метаданих, стійкість до обфускації, підтримка кількох мов та комбіновані схеми фільтрації).

ВИСНОВКИ

В магістерській роботі розглянуті питання розробки комп'ютерної системи автоматичного виявлення спаму на основі методу опорних векторів (SVM).

В першому розділі виконано комплексний огляд предметної області автоматичного виявлення спам, огляд вітчизняної та зарубіжної літератури і патентно-інформаційний пошук за тематикою автоматичного виявлення спаму. Проаналізовано сучасні підходи протидії спаму і наведено їх порівняльні переваги та обмеження. Обґрунтовано доцільність застосування методів машинного навчання для підвищення адаптивності та точності фільтрації, розглянуто ключові алгоритми класифікації (Naive Bayes, дерева рішень/ансамблі, k-NN, нейромережі, SVM) та етапи побудови ML-рішення (векторизація, навчання, валідація, оцінювання). Серед них метод опорних векторів виділяється як один з найефективніших для задач текстової класифікації.

Наприкінці сформульовано мету дослідження та перелік завдань, що визначають подальшу структуру і зміст наступних розділів роботи.

У другому розділі виконано проєктування системи автоматичного виявлення спаму та сформовано ключові рішення щодо її реалізації. Обґрунтовано вибір методу опорних векторів як базового алгоритму класифікації та визначено технологічний стек розробки (Python, Scikit-learn, засоби опрацювання текстів). Описано джерела даних для навчання й перевірки (SpamAssassin Public Corpus, SMS Spam Collection, власна сучасна вибірка 2024–2025 рр.), наведено правила розподілу даних на навчальну, валідаційну та тестову вибірки. Спроєктовано конвеєр попередньої обробки повідомлень (очищення, токенізація, нормалізація, вилучення стоп-слів, стемінг/лематизація) та підхід до формування ознак на основі TF–IDF (з урахуванням n-грам), а також визначено набір додаткових кількісних характеристик, що підсилюють точність моделі. Розроблено алгоритм роботи системи в режимі побудови (навчання)

моделі з підбором гіперпараметрів і в режимі експлуатації для класифікації нових повідомлень; обґрунтовано вибір конфігурації SVM (лінійне ядро з прийнятними параметрами за результатами валідації). Для формалізації структури та взаємодії компонентів виконано UML-моделювання (діаграми варіантів використання, діяльності, послідовності, компонентів, розгортання та класів). Завершально сформовано модульну архітектуру програмного рішення (передобробка, векторизація, класифікація, користувацький інтерфейс, сховище/репозиторій моделі) й описано взаємодію модулів, що забезпечує розширюваність і готовність прототипу до подальшої реалізації та експериментальної перевірки.У цьому розділі виконано повний цикл реалізації та перевірки розробленої системи автоматичного виявлення спаму.

Програмний комплекс створено на Python 3.9 із логічним розділенням модулів: передобробка очищає вхідні повідомлення (HTML, службові фрагменти), нормалізує типові шаблони (<URL>, <EMAIL>, <NUM>) та формує токени зі стоп-фільтрацією і стемінгом. Далі реалізовано формування ознак на основі TF–IDF з уніграмами й біграмами, доповнене компактним набором числових характеристик, що описують “форму” тексту. Для класифікації застосовано SVM (переважно лінійний варіант як оптимальний для розріджених високовимірних текстових матриць), а артефакти “векторизатор + модель” зберігаються спільно, що гарантує узгодженість між навчанням і застосуванням.

Практичне використання системи продемонстровано у двох режимах: консольному (для пакетної перевірки файлів і збору статистики) та веб-інтерфейсі (для інтерактивної перевірки тексту з відображенням результату і впевненості моделі). Експериментальна оцінка показала високу якість на тестовому наборі SAPC+SMS: accuracy 98.2%, precision для «spam» 97.0%, recall 95.5%, F_1 96.2%. Аналіз помилок засвідчив типові причини: хибнопозитивні рішення частіше виникають на легальних промо-розсилках, близьких до спаму за лексикою, а хибнонегативні — на коротких фішингових листах без явних маркерів або на повідомленнях іншою мовою через дефіцит відповідних

прикладів у навчанні. Порівняння з наївним баєсом підтвердило перевагу SVM за ключовими метриками, зокрема за точністю позитивного прогнозу, що важливо для мінімізації небажаних блокувань.

Додатковий тест на актуальних листах 2024–2025 років дав 94% правильних рішень (47/50), що вказує на загальну переносимість моделі, але також підкреслює потребу періодичного донавчання на сучасних і мовно релевантних даних. Вимірювання швидкодії та обсягу артефактів показали, що система працює достатньо швидко для практичних сценаріїв і не потребує значних ресурсів, тому може розглядатися як придатна основа для подальшого розвитку (врахування вкладень і метаданих, стійкість до обфускації, підтримка кількох мов та комбіновані схеми фільтрації).

Розроблена система продемонструвала високу точність виявлення спаму на тестових даних. Це підтверджує, що метод опорних векторів у поєднанні з ретельною підготовкою ознак є ефективним підходом для побудови антиспам-фільтра. Практична цінність полягає в тому, що такий підхід можна інтегрувати в реальні поштові сервіси для автоматичного сортування вхідної пошти. За рахунок високої точності (близько 97–98%) користувачів будуть захищені від більшості небажаних листів, а ризик втрати важливого листа через помилкову фільтрацію мінімізований.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Spamhaus. General Definitions: What is Spam? URL: https://www.spamhaus.org/faqs/general-definitions/ (дата звернення: 14.12.2025).

2. Guzella T. S., Caminhas W. M. A review of machine learning approaches to spam filtering. Expert Systems with Applications, 2009. URL: https://doi.org/10.1016/j.eswa.2009.02.037 (дата звернення: 14.12.2025).

3. Apache SpamAssassin Project. Overview / Documentation. URL: https://spamassassin.apache.org/ (дата звернення: 14.12.2025).

4. Delany S. J., Cunningham P., Tsymbal A., Coyle L. A case-based technique for tracking concept drift in spam filtering. 2005. URL: https://lorcancoyle.org/pdf/2005/Delany2005Case.pdf (дата звернення: 14.12.2025).

5. Залива Ю. О., Бондарчук І. О., Золотухіна А. О. Фільтрація спаму електронної пошти за допомогою машинного навчання. URL: https://con.duikt.edu.ua/index.php/communication/article/view/2388 (дата звернення: 14.12.2025).

6. US7930353B2. Trees of classifiers for detecting email spam. URL: https://patents.google.com/patent/US7930353B2/en. (дата звернення: 14.12.2025).

7. IETF. Anti-Spam Recommendations for SMTP MTAs (RFC 2505). URL: https://datatracker.ietf.org/doc/html/rfc2505 (дата звернення: 14.12.2025).

8. Metsis V., Androutsopoulos I., Paliouras G. Spam filtering with Naive Bayes — which Naive Bayes? CEAS 2006. URL: https://dblp.org/rec/conf/ceas/MetsisAP06.html (дата звернення: 14.12.2025).

9. Drucker H., Wu D., Vapnik V. Support Vector Machines for Spam Categorization. NeurIPS, 1999. URL: https://dblp.org/rec/conf/nips/DruckerWV99.html (дата звернення: 14.12.2025).

10. Pedregosa F. et al. Scikit-learn: Machine Learning in Python. JMLR, 2011. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf (дата звернення: 14.12.2025).

11. Scikit-learn documentation. sklearn.svm.SVC. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (дата звернення: 14.12.2025).

12. Natural Language Toolkit (NLTK). Documentation / Book. URL: https://www.nltk.org/book/ (дата звернення: 14.12.2025).

13. Flask Documentation. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 14.12.2025).

14. Apache SpamAssassin Public Corpus. README. URL: https://spamassassin.apache.org/old/publiccorpus/readme.html (дата звернення: 14.12.2025).

15. Стандарт формату листів Internet Message Format (RFC 5322). URL: https://datatracker.ietf.org/doc/html/rfc5322 (дата звернення: 14.12.2025).

16. Jurafsky D., Martin J. H. Speech and Language Processing (3rd ed. draft). URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 14.12.2025).

17. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. URL: https://nlp.stanford.edu/IR-book/ (дата звернення: 14.12.2025).

18. Apache SpamAssassin Public Corpus. Download index. URL: https://spamassassin.apache.org/old/publiccorpus/ (дата звернення: 14.12.2025).

19. Beautiful Soup Documentation. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 14.12.2025).

20. Scikit-learn documentation. sklearn.feature_extraction.text.TfidfVectorizer. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (дата звернення: 14.12.2025).

21. Scikit-learn documentation. Model persistence. URL: https://scikit-learn.org/stable/model_persistence.html (дата звернення: 14.12.2025).

22. Object Management Group (OMG). Unified Modeling Language (UML) Specification 2.5.1. URL: https://www.omg.org/spec/UML/2.5.1/ (дата звернення: 14.12.2025).

23. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. URL: https://martinfowler.com/books/uml.html (дата звернення: 14.12.2025).

24. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. Addison-Wesley, 2005.

25. UCI Machine Learning Repository. SMS Spam Collection. URL: https://archive.ics.uci.edu/dataset/94/sms+spam+collection (дата звернення: 14.12.2025).

26. Porter M. F. An algorithm for suffix stripping. Program, 1980. URL: https://doi.org/10.1108/eb046814 (дата звернення: 14.12.2025).

27. Cortes C., Vapnik V. Support-vector networks. Machine Learning, 1995. URL: https://doi.org/10.1007/BF00994018 (дата звернення: 14.12.2025).

28. Joachims T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. 1998. URL: https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf (дата звернення: 14.12.2025).

29. IETF. Domain-based Message Authentication, Reporting, and Conformance (DMARC) (RFC 7489). URL: https://datatracker.ietf.org/doc/rfc7489/ (дата звернення: 14.12.2025).