Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

# Кваліфікаційна робота магістра

на тему: «Розроблення моделі машинного навчання для класифікації вхідних повідомлень»

Виконав: студент групи К23-2М

Спеціальність 122 Комп'ютерні науки

Ковзиков О.Ю.
(прізвище та ініціали)

Керівник к.е.н. Яковенко Т. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Дніпровський державний технічний університет
(місце роботи)

доцент кафедри математичного моделювання та системного аналізу
(посада)

к.т.н., доц. Волосова Н.М.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

# АНОТАЦІЯ

Ковзиков О.Ю. Розроблення моделі машинного навчання для класифікації вхідних повідомлень.

Дипломна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп'ютерні науки» – Університет митної справи та фінансів, Дніпро, 2025.

У кваліфікаційній роботі розглянуто проблему автоматизації класифікації текстових повідомлень шляхом створення моделі машинного навчання. Завдання класифікації текстів передбачає автоматичне розподілення повідомлень на категорії, що дозволяє значно знизити навантаження на людські ресурси, підвищити точність обробки даних та скоротити час, необхідний для прийняття рішень.

У роботі проведено аналіз сучасних підходів до класифікації текстів, включаючи такі методи, як наївний баєсів класифікатор, метод опорних векторів (SVM), дерева рішень та глибинні нейронні мережі. Кожен із цих підходів оцінено з точки зору їх ефективності, точності, швидкості обробки та можливостей до адаптації у різних умовах.

Для забезпечення якості класифікації було впроваджено алгоритми попередньої обробки тексту, такі як видалення стоп-слів, стемінг, лематизація та векторизація тексту з використанням методів TF-IDF і Word2Vec. Ці етапи дозволили знизити шум у даних, зменшити розмірність задачі та забезпечити підготовку текстів до подальшого аналізу. Практична частина роботи включає розробку програмної моделі, яка забезпечує високу точність класифікації повідомлень на основі експериментально визначених параметрів.

Ключові слова: класифікація тексту, машинне навчання, попередня обробка тексту, глибинні нейронні мережі, метод опорних векторів, векторизація тексту, видалення стоп-слів, лематизація.

# ABSTRACT

Kovzykov O.Yu. Development of a machine learning model for classifying incoming messages.

Diploma thesis for obtaining a master's degree in specialty 122 «Computer Science» – University of Customs and Finance, Dnipro, 2025.

The master's thesis deals with the problem of automating the classification of text messages by creating a machine learning model. The task of text classification involves the automatic categorization of messages, which can significantly reduce the burden on human resources, increase the accuracy of data processing and reduce the time required for decision-making.

This paper analyzes modern approaches to text classification, including methods such as naive Bayesian classifier, support vector machine (SVM), decision trees, and deep neural networks. Each of these approaches was evaluated in terms of their efficiency, accuracy, processing speed, and adaptability to different conditions.

To ensure the quality of the classification, we implemented text preprocessing algorithms, such as stop word removal, stemming, lemmatization, and text vectorization using TF-IDF and Word2Vec methods. These steps allowed us to reduce noise in the data, reduce the dimensionality of the problem, and prepare the texts for further analysis. The practical part of the work includes the development of a software model that ensures high accuracy of message classification based on experimentally determined parameters.

Keywords: text classification, machine learning, text preprocessing, deep neural networks, support vector machine, text vectorization, stop word removal, lemmatization.

ЗМІСТ

# ВСТУП

У сучасному світі, що стрімко розвивається, кількість цифрових даних, зокрема текстових повідомлень, зростає експоненційно. Всі ці дані мають величезний потенціал для покращення різних процесів, але для їх ефективного використання необхідні інноваційні технології, які дозволяють автоматизувати аналіз та класифікацію вхідної інформації. Особливо актуальним це завдання є в сфері обробки текстових повідомлень, що надходять до різних систем – від електронної пошти до соціальних мереж і чат-ботів. Виявлення та розподіл цих повідомлень за категоріями за допомогою моделей машинного навчання дозволяє значно підвищити ефективність обробки даних та прийняття рішень, що, в свою чергу, сприяє значному зниженню витрат часу та ресурсів.

Однією з ключових задач у цій сфері є створення точних та ефективних моделей класифікації, які дозволяють автоматично розподіляти повідомлення на категорії за певними ознаками. Це є надзвичайно важливим для широкого кола застосувань, таких як автоматизація технічної підтримки, розподіл електронної пошти по папках, виявлення спаму, а також у системах обробки звернень клієнтів. Технології машинного навчання відкривають безліч можливостей для автоматизації цих процесів, надаючи організаціям інструменти для підвищення ефективності та зниження кількості помилок, які виникають при ручному обробленні інформації.

Актуальність обраної теми підтверджується також тим, що з кожним роком збільшується кількість вхідних повідомлень, які потребують швидкої та точної обробки. Класичні методи аналізу тексту вже не здатні справлятися з такою великою кількістю інформації, що зумовлює потребу у розробці нових моделей та алгоритмів, які забезпечать точну та швидку класифікацію текстових повідомлень. Це вимагає поглибленого вивчення існуючих методів

машинного навчання, а також пошуку нових підходів до вирішення проблеми класифікації повідомлень.

Мета роботи полягає у розробці та впровадженні моделі машинного навчання для класифікації вхідних повідомлень, що забезпечить автоматизовану обробку та точний розподіл текстових даних на відповідні категорії. Для досягнення цієї мети необхідно вирішити низку важливих завдань.

Завдання роботи:

– провести аналіз існуючих методів машинного навчання, що використовуються для класифікації текстових повідомлень, зокрема розглянути основні алгоритми, такі як наївний баєсів класифікатор, метод опорних векторів, дерева рішень та глибинні нейронні мережі;

– розробити структуру та архітектуру моделі машинного навчання, що буде здатна забезпечити високу точність класифікації текстових повідомлень;

– провести експерименти з різними підходами до попередньої обробки тексту, включаючи методи видалення стоп-слів, лематизації та стемінгу;

– оцінити ефективність розробленої моделі за допомогою стандартних метрик якості, таких як точність, повнота, F1-міра та час обробки;

– розробити рекомендації щодо застосування створеної моделі в реальних умовах для автоматизації процесів обробки текстових повідомлень.

Об’єктом дослідження є процес класифікації текстових повідомлень з використанням алгоритмів машинного навчання, а також самі текстові повідомлення, які є різноманітними за змістом, темами та форматом.

Предметом дослідження є методи та алгоритми машинного навчання, що використовуються для класифікації текстових даних, а також процес побудови та оптимізації моделей для цього завдання.

Методи дослідження включають:

– методи машинного навчання. для створення класифікаційних моделей та їх навчання на основі вибраних даних;

– методи статистичного аналізу. для оцінки якості моделі та порівняння результатів різних алгоритмів;

– методи попередньої обробки тексту. для підготовки даних, зокрема методи лексичного та семантичного аналізу тексту;

– експериментальні методи. для порівняння різних підходів до вирішення задачі класифікації.

Практична значимість роботи полягає у тому, що розроблена модель машинного навчання для класифікації вхідних повідомлень може бути успішно застосована в різних сферах, де необхідна автоматизація обробки текстових даних. Це включає застосування в автоматичних системах підтримки користувачів, виявленні спаму в електронній пошті, а також у багатьох інших бізнес-процесах, де необхідно обробляти великі обсяги вхідної текстової інформації. Результати роботи можуть бути використані як основа для подальших досліджень та розробок в області обробки природної мови та машинного навчання.

Наукова новизна роботи полягає в розробці нової моделі класифікації вхідних повідомлень, яка поєднує в собі як традиційні, так і сучасні методи машинного навчання. В рамках роботи пропонується оптимізація існуючих підходів до класифікації текстових повідомлень шляхом інтеграції різних алгоритмів та методів попередньої обробки тексту. Зокрема, буде розглянуто застосування глибинних нейронних мереж для задачі класифікації, що дозволяє підвищити точність моделі на складних даних. Крім того, пропонується методика адаптації моделі до специфічних умов різних сфер застосування, що підвищує її гнучкість і універсальність.

Структура кваліфікаційної роботи. Кваліфікаційна робота магістра складається з трьох розділів. Обсяг кваліфікаційної роботи – 85 сторінок. Робота містить 11 рисунків. Список використаних джерел складається з 16 посилань.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

## 1.1 Наївний баєсів класифікатор для класифікації текстових повідомлень

Наївний баєсів класифікатор є одним з найбільш популярних методів для розв’язання задач класифікації текстових повідомлень, зокрема в контексті обробки природної мови. Він належить до класу статистичних методів і ґрунтується на застосуванні теореми Байєса для прогнозування ймовірності належності певного тексту до одного з класів [1]. Хоча його назва містить слово «наївний», це не знижує ефективності методу, оскільки він забезпечує досить точні результати в ряді практичних застосувань, зокрема при класифікації текстових даних.

Наївний баєсів класифікатор спрощує задачу класифікації, зробивши важливе припущення, яке полягає в незалежності ознак (в даному випадку слів чи термінів) одне від одного в межах одного класу. Це припущення є наївним, оскільки в реальності слова в текстах часто взаємопов’язані. Однак, навіть за таких спрощень, метод виявляється досить потужним і широко застосовуваним у різних областях, таких як фільтрація спаму, класифікація новинних статей, визначення настроїв у текстах та багатьох інших. Основою наївного баєсів класифікатора є застосування теореми Байєса, яка дає змогу обчислити ймовірність того, що певне повідомлення належить до певного класу, виходячи з наявних даних про розподіл слів у тексті для кожного класу. Для цього використовуються дві основні компоненти: ймовірність класу і ймовірність спостереження конкретного слова в межах цього класу.

Підхід наївного баєсів класифікатора полягає у використанні окремих умовних ймовірностей для кожного слова в тексті, тобто ймовірність появи кожного слова у конкретному класі розглядається незалежно від інших слів (рис. 1.1). Це спрощує обчислення ймовірності, оскільки замість того, щоб

враховувати всі взаємозв’язки між словами в тексті, розрахунок ймовірності кожного слова відбувається окремо. Така незалежність є наївною, адже в реальних текстах слова часто мають взаємозалежності, однак, за рахунок спрощення обчислень цей метод є достатньо ефективним і швидким.

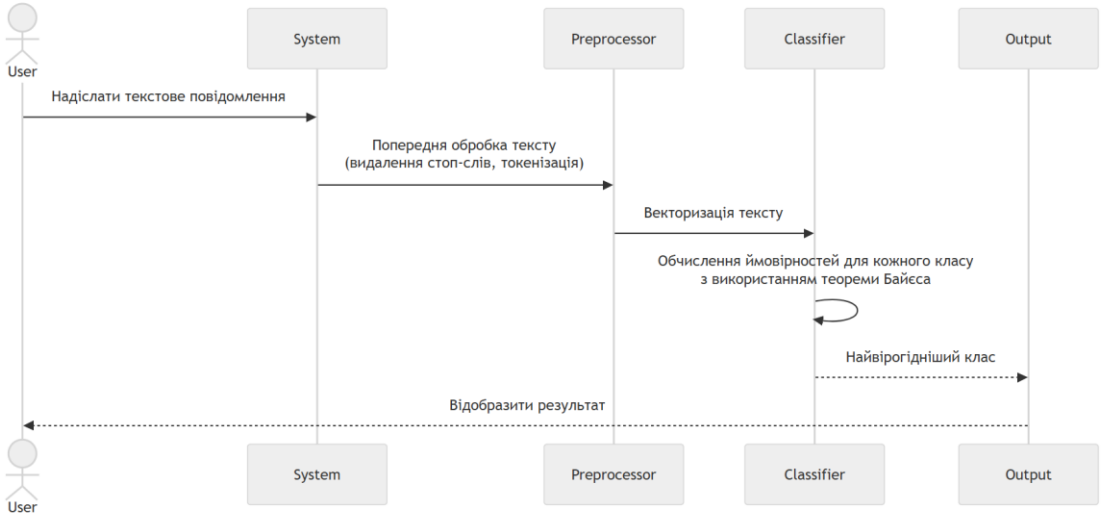


Рисунок 1.1 – Послідовність використання наївного класифікатора Байєса

Кроки для побудови наївного баєсів класифікатора для текстових повідомлень полягають у наступному: спочатку необхідно зібрати дані для навчання, що включають тексти, що вже мають відповідні мітки класів [2]. Далі ці тексти потрібно обробити, що включає в себе попереднє очищення даних – видалення непотрібних символів, таких як пунктуація чи стоп-слова, а також перетворення текстів у формат, який підходить для подальшої роботи з ними, наприклад, шляхом розбиття на окремі слова (токенізація). Наступним кроком є визначення ймовірностей для кожного слова в текстах для кожного класу, що дозволяє створити статистичну модель для класифікації нових текстів. Після цього, коли модель навчена, наївний баєсів класифікатор може використовувати ймовірності для кожного класу, щоб визначити, до якого класу належить новий текст. Ключовим моментом є вибір правильного класу:

для кожного тексту на основі навченої моделі обчислюються ймовірності того, що цей текст належить до кожного з класів, і вибирається клас з найбільшою ймовірністю.

Попри те, що наївний баєсів класифікатор має свої обмеження, зокрема через припущення про незалежність слів, він показує високу ефективність при класифікації текстових повідомлень в умовах реальних задач. Однією з переваг методу є його швидкість і ефективність, оскільки модель навчання є відносно легкою і потребує мінімальних ресурсів для обробки навіть великих обсягів даних. Крім того, метод є стійким до надмірної складності даних, таких як великі розміри словників, оскільки кожен клас описується лише ймовірностями для кожного слова окремо, що значно знижує складність навчання.

Завдяки своїй простоті наївний баєсів класифікатор є популярним вибором для ряду практичних застосувань, зокрема в задачах фільтрації спаму. У цьому випадку метод використовується для визначення того, чи є вхідне повідомлення спамом або ні, на основі ознак, таких як наявність певних слів чи виразів, які часто зустрічаються в спам-повідомленнях [2]. Важливим моментом є також використання наївного баєсів класифікатора для аналізу настроїв текстів, наприклад, у випадку оцінки відгуків користувачів на продукти чи послуги. У цьому випадку класифікація відбувається за двома класами – позитивний чи негативний відгук, що дозволяє швидко отримувати загальну оцінку настроїв користувачів.

Варто зазначити, що наївний баєсів класифікатор має кілька варіантів. Один із них – це бінарна класифікація, де існують лише два можливі класи, і кожне повідомлення має бути віднесене до одного з них. В іншому випадку класифікація може бути багатокласовою, коли повідомлення повинне бути віднесене до одного з кількох можливих класів, таких як класи новин (політика, спорт, культура тощо). При багатокласовій класифікації наївний баєсів класифікатор обчислює ймовірності для кожного класу і вибирає той

клас, для якого ймовірність є найвищою. Незважаючи на свою простоту, наївний баєсів класифікатор має низку недоліків, зокрема через припущення про незалежність слів. У реальних текстах, як правило, існують складні зв’язки між словами, що може знижувати точність моделі. У таких випадках інші методи класифікації, як-от методи на основі нейронних мереж, можуть продемонструвати кращі результати, оскільки вони здатні враховувати контекст слів і їх взаємозв’язки. Проте наївний баєсів класифікатор залишається одним з найбільш ефективних і простих інструментів для швидкої класифікації текстових повідомлень, особливо коли важливо мати модель, яка швидко навчається і має низькі вимоги до обчислювальних ресурсів. Його застосування не обмежується лише текстовою класифікацією, оскільки він також може бути використаний для прогнозування ймовірностей в інших областях, таких як фінанси, медицина, біоінформатика та багато інших.

## 1.2 Метод опорних векторів

Метод опорних векторів став популярним завдяки своїй здатності ефективно працювати з високими розмірами простору ознак і навіть у випадку, коли дані не можуть бути лінійно відокремлені. Основна ідея, яка лежить в основі SVM, це пошук гіперплощини, яка розділяє простір так, щоб відстань між цією площиною і найближчими точками кожного класу була максимальною [1-3]. Ці точки, які знаходяться найближче до гіперплощини, називаються опорними векторами, оскільки саме вони визначають позицію розділяючої площини. Знайдення такої гіперплощини є задачею оптимізації, і в цьому процесі важливу роль відіграє теорема Лагранжа та методи квадратичного програмування.

Ключовим аспектом SVM є максимізація маржі, що є відстанню між гіперплощиною та найближчими точками кожного класу. Це дозволяє зменшити ймовірність помилок при класифікації нових точок, оскільки велика

маржа зазвичай асоціюється з кращими генералізаційними властивостями моделі. Проте на практиці, оскільки дані часто не можуть бути лінійно відокремлені, використовується метод ядра, який дозволяє здійснювати перехід в більш високий простір ознак, де дані можуть бути лінійно відокремлені. Це перетворення до більш високої розмірності є дуже важливим для SVM, оскільки воно дозволяє зберігати переваги методу навіть в умовах складних, нелінійних даних. Одним із найбільш відомих ядер є радіально-базисне ядро (RBF), яке є ефективним у багатьох реальних задачах. Що стосується класифікації текстових повідомлень, то завдання можна уявити як задачу віднесення тексту до одного з кількох класів, наприклад, спам чи не спам, позитивний чи негативний настрій, політичні новини чи спортивні новини тощо. Тексти зазвичай представляються у вигляді векторів ознак, де кожен елемент вектора відповідає наявності або частоті певного слова чи фрази в тексті. Один із основних викликів при застосуванні SVM до текстових даних – це ефективне перетворення тексту в числові ознаки, які можна використовувати для навчання моделі.

Зазвичай для перетворення текстів у вектори використовуються такі методи, як мішок слів (bag-of-words), TF-IDF (термін-частота, інверсія частоти документа) або більш складні векторні представлення слів, як-от word2vec чи GloVe. У методі «мішок слів» кожен текст представляється у вигляді вектора, що складається з кількості або частоти слів, які зустрічаються в цьому тексті, без урахування їх порядку. Цей підхід добре працює для завдань, де важливими є окремі слова, але він може втратити контекст і значення взаємозв’язків між словами в тексті. Метод TF-IDF дозволяє скоригувати важливість слів в тексті на основі їх частоти в документі та рідкості в колекції документів [4]. Він надає більшу вагу словам, які зустрічаються часто в конкретному документі, але рідко в інших документах, тим самим підвищуючи значущість унікальних термінів для класифікації. Це дозволяє краще фіксувати ключові слова, які відрізняють один клас текстів від іншого.

При використанні SVM для класифікації текстів, важливим етапом є вибір ядра, оскільки різні ядра можуть по-різному впливати на якість класифікації. Наприклад, для текстових даних часто застосовується ядро RBF, яке дозволяє будувати гнучкі моделі, здатні обробляти нелінійні залежності між словами в тексті. Це ядро перетворює дані в простір більш високої розмірності, що дає можливість побудувати ефективну гіперплощину, яка може добре розділяти класи текстів. Інші типи ядер, такі як поліноміальне чи лінійне ядро, також можуть бути використані залежно від особливостей даних.

Процес навчання SVM для текстової класифікації полягає в оптимізації параметрів моделі, таких як параметр регуляризації C, який визначає компроміс між мінімізацією помилок на тренувальних даних і максимізацією маржі. Параметр регуляризації дозволяє контролювати складність моделі: маленьке значення C може привести до підгонки моделі з високою маржею, але з більшою кількістю помилок, тоді як велике значення C прагне до мінімізації помилок на тренувальних даних, що може привести до перенавчання.

Для вибору оптимальних параметрів моделі та ядер часто використовуються методи крос-валідації, які дозволяють оцінити ефективність моделі на невідомих даних і допомогти вибрати найкращу модель для класифікації тексту. Завдяки цьому, SVM забезпечує дуже хороші результати навіть в умовах обмежених даних, адже він орієнтований на максимізацію маржі та здатний працювати з високими вимогами до точності, навіть у випадках складних, багатовимірних даних. Метод опорних векторів має безліч переваг при класифікації текстів, серед яких можна виділити високу точність на малих і середніх наборах даних, стійкість до перенавчання, здатність працювати з великою кількістю ознак і можливість ефективно обробляти нелінійно роздільні дані [4, 5]. Проте є й деякі обмеження, пов’язані з обчислювальними витратами на великих наборах даних, що потребує значних ресурсів для навчання моделі, особливо при великих обсягах

текстових даних. Крім того, метод опорних векторів може бути менш ефективним у випадках, коли дані мають високу вимірність, але не містять корисної інформації, а також коли є велика кількість шуму в даних. В таких випадках SVM може втратити свою ефективність і віддати перевагу простим моделям, таким як логістична регресія. Проте з розвитком обчислювальних можливостей і вдосконаленням алгоритмів оптимізації, ці недоліки все більше нівелюються, і SVM залишаються важливим інструментом у задачах класифікації текстових повідомлень.

Метод опорних векторів також може бути використаний в багатокласових задачах, де потрібно класифікувати текст до одного з кількох класів. Для цього можуть бути використані різні стратегії, такі як «один проти одного» чи «один проти всіх», коли для кожного класу будується окрема модель і результати комбінуються для вибору найбільш вірогідного класу. Це дозволяє застосовувати SVM до більш складних класифікаційних задач, де є кілька класів і немає чіткої лінії розподілу між ними.

Метод опорних векторів є одним з найефективніших і широко застосовуваних методів у класифікації текстових повідомлень, і завдяки своїй здатності до адаптації під різні умови, залишається актуальним інструментом в області машинного навчання.

## 1.3 Дерева рішень

Дерева рішень є одним з найбільш відомих і широко використовуваних методів машинного навчання для класифікації даних, зокрема текстових повідомлень. Цей метод є інтуїтивно зрозумілим і має кілька переваг, таких як висока інтерпретованість, здатність працювати з як числовими, так і категоріальними ознаками, а також можливість обробки відносно великих обсягів даних. Дерево рішень – це структура, що нагадує дерево, де кожен вузол є тестом або порогом для певної ознаки (наприклад, частота слова), а

кожне розгалуження відповідає різним варіантам результату цього тесту. У кінцевих листках дерева знаходяться передбачення класів, до яких належать об’єкти (тексти), що проходять через дерево [3, 4]. Побудова дерева рішень полягає у послідовному розділенні простору ознак на підмножини, так щоб кожен етап забезпечував максимально точну класифікацію об’єктів. Для цього використовуються різні критерії поділу даних, серед яких найбільш популярними є критерії, що оцінюють «чистоту» підмножини, зокрема такі, як індекс Джіні, ентропія та інформаційний приріст. Кожен критерій має на меті мінімізувати помилкові класифікації на кожному етапі, що призводить до отримання дерева, яке здатне точніше передбачати клас тексту, зберігаючи при цьому простоту і ефективність.

Дерево рішень застосовується до класифікації текстових повідомлень, коли необхідно розділити тексти на категорії, такі як спам і не спам, позитивні і негативні відгуки, тематичні класи новин (політика, спорт, культура тощо). На етапі підготовки даних кожен текст перетворюється в набір числових ознак, таких як частота появи слів (TF), термін-частота – інверсія частоти документа (TF-IDF), або більш складні числові представлення, наприклад, на основі векторних просторових моделей слів (word2vec, GloVe). Це дозволяє використовувати дерево рішень для класифікації, де на кожному етапі дерева здійснюється порівняння значень певних ознак, наприклад, наявності чи частоти вживання певних слів або фраз у тексті.

Процес побудови дерева рішень для текстової класифікації передбачає кілька основних етапів, серед яких важливими є вибір відповідного критерію поділу та визначення найкращих ознак для кожного вузла дерева. Перший етап включає в себе підготовку набору даних, де кожен текст перетворюється у вектор ознак, що містить інформацію про наявність слів, частоти їх використання або інших характеристик, які можуть бути корисними для класифікації [4, 5]. Після цього необхідно побудувати дерево рішень, використовуючи алгоритм, який на кожному кроці вибирає найбільш

інформативні ознаки для поділу даних. Це може бути, наприклад, частота появи конкретного слова у повідомленнях певного класу. Алгоритм будує дерево, послідовно розгалужуючи його на підмножини, що містять елементи, які належать до певних класів. Один із популярних алгоритмів для побудови дерев рішень – це ID3 (Iterative Dichotomiser 3), який використовує ентропію і інформаційний приріст для вибору найкращої ознаки на кожному етапі. Ідея полягає в тому, щоб вибрати ознаку, яка дає найбільший інформаційний приріст, тобто таку ознаку, що максимально зменшує невизначеність класифікації даних. На кожному етапі дерева вибирається ознака, яка максимально покращує класифікацію, і так до тих пір, поки всі об’єкти не будуть коректно класифіковані або не будуть досягнуті певні критерії зупинки.

Іншим популярним алгоритмом є C4.5, який є вдосконаленою версією ID3 і додатково використовує критерії, що обмежують максимальну глибину дерева та враховують можливі помилки класифікації для різних класів. C4.5 також включає механізм відсічення дерева, що дозволяє уникнути перенавчання. Перенавчання є однією з головних проблем дерев рішень, оскільки дуже глибоке дерево може надмірно точнісно підганяти дані тренувальної вибірки, що в результаті призводить до погіршення його здатності до генералізації на нових даних [5]. Відсічення дозволяє видаляти частини дерева, які не вносять значного внеску в покращення класифікації або можуть бути результатом випадкових флуктуацій у тренувальних даних. Застосування дерев рішень до класифікації текстових повідомлень є досить ефективним завдяки високій інтерпретованості отриманих моделей. Кожен вузол дерева може бути легко зрозумілим і інтерпретованим людиною, оскільки рішення, які приймаються на кожному етапі, ґрунтуються на конкретних, зрозумілих ознаках, таких як частота слів або присутність певних фраз. Це робить дерева рішень дуже корисними в тих випадках, коли важливо не тільки отримати точні результати класифікації, але й зрозуміти, чому саме

так було прийнято рішення. Відповідно, дерева рішень є особливо корисними в таких сферах, як медичні діагнози, юридична класифікація та інші галузі, де прозорість і пояснюваність моделі є важливими аспектами.

Однак, незважаючи на ці переваги, метод дерев рішень має свої обмеження. Одним з основних недоліків є схильність до перенавчання, особливо якщо дані мають велику варіативність або якщо дерево рішень будується без відсічення. Крім того, дерева рішень можуть не давати хороших результатів, коли необхідно працювати з дуже складними, нелінійно роздільними даними. В таких випадках інші методи, такі як методи опорних векторів чи нейронні мережі, можуть бути ефективнішими.

Для того, щоб подолати ці проблеми, було запропоновано кілька методів вдосконалення дерев рішень, серед яких одним з найбільш популярних є ансамблювання дерев, зокрема за допомогою алгоритмів, таких як Random Forest або Gradient Boosting. Random Forest є методом, що використовує безліч дерев рішень, кожне з яких побудовано на випадковій підмножині даних, що дозволяє значно зменшити варіативність моделі та покращити її точність. У свою чергу, алгоритм Gradient Boosting будує дерева рішень поетапно, кожне наступне дерево виправляє помилки попереднього, що дозволяє досягти ще кращих результатів [5, 6]. Для текстової класифікації найбільш поширеними задачами є розподіл текстів на категорії, такі як визначення спаму, категоризація новинних статей або оцінка настрою відгуків. Застосування дерев рішень у цих сферах дозволяє ефективно розподіляти тексти по класах на основі ключових слів або фраз, що входять до їх складу. Завдяки своєму характеру дерева рішень можуть працювати навіть з великими наборами даних, що робить їх корисними для класифікації великих текстових корпусів.

Загалом, метод дерев рішень є важливим інструментом в арсеналі методів машинного навчання для класифікації текстових повідомлень. Його простота, інтерпретованість і здатність до адаптації під різні типи даних роблять його потужним і ефективним методом для багатьох реальних завдань.

Однак, як і будь-який інший метод, дерева рішень мають свої обмеження, і їх ефективність часто залежить від правильної настройки параметрів моделі, а також від додаткових стратегій, таких як ансамблювання чи відсічення дерев.

## 1.4 Глибинні нейронні мережі для класифікації тексту

Глибинні нейронні мережі (GNN) представляють собою одну з найсучасніших і потужних технологій у сфері машинного навчання, які широко застосовуються для класифікації тексту та інших типів даних (рис. 1.2). Ці мережі, що є підкласом штучних нейронних мереж, відрізняються великою кількістю шарів між вхідними та вихідними елементами, що дозволяє їм моделювати складні патерни та взаємозв’язки в даних [6, 7]. У контексті класифікації тексту глибинні нейронні мережі можуть бути використані для ефективного аналізу великих обсягів текстової інформації та автоматичного визначення її категорії, будь то тематика, настрій, спам чи інші категорії.

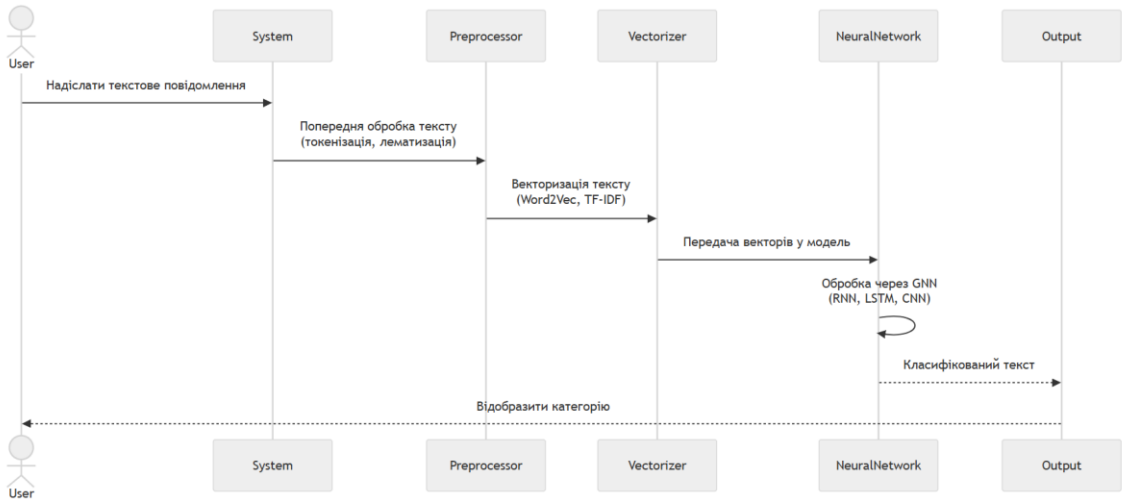


Рисунок 1.2 – Процес класифікації тексту за допомогою GNN

Основною перевагою глибинних нейронних мереж є їх здатність навчатися на великих масивах неструктурованих даних, таких як текстові повідомлення, з мінімальною необхідністю в попередній обробці. Ці мережі можуть адаптуватися до складних закономірностей у даних, таких як синтаксичні та семантичні взаємозв'язки, ідентифікувати контекст, відображати значення слів у конкретному текстовому контексті та виконувати складні класифікаційні завдання. Відтак, застосування глибинних нейронних мереж для класифікації тексту є одним із найбільш ефективних підходів у сучасних дослідженнях та комерційних застосуваннях машинного навчання.

Класифікація тексту за допомогою глибинних нейронних мереж починається з етапу векторизації текстових даних. Для цього існує кілька методів, зокрема традиційний підхід «мішок слів» (bag of words), який відображає частоти слів в тексті, а також більш складні методи векторизації, такі як термін-частотний індекс з інверсією частоти документа (TF-IDF). Однак для того, щоб глибинні нейронні мережі могли ефективно працювати з текстовими даними, зазвичай використовуються сучасні методи векторизації, які враховують контекстуальне значення слів, такі як вектори слів (word embeddings) [6-8]. Ці вектори, створені за допомогою методів на кшталт Word2Vec, GloVe або FastText, дозволяють кожному слову мати представлення у вигляді багатовимірного вектора, що відображає його значення в контексті інших слів. Це дозволяє мережам враховувати не лише присутність слів, але й їх контекстуальне значення, що суттєво покращує якість класифікації тексту. Глибинні нейронні мережі для класифікації тексту використовують багатошарову архітектуру, де кожен шар виконує певну операцію над даними, поступово виділяючи все більш складні та абстрактні ознаки. Однією з найпоширеніших архітектур таких мереж є мережі з повторними нейронами (Recurrent Neural Networks, RNN), які спеціально створені для роботи з послідовностями даних, такими як текст. Оскільки текст має природну послідовність, де кожне слово залежить від попередніх, RNN

дозволяє моделювати цю залежність. Однак стандартні RNN мають обмеження, пов’язані з проблемою зникнення або вибуху градієнтів при навчанні на великих послідовностях, що може призводити до труднощів при роботі з довгими текстами.

Задля подолання цих проблем були розроблені вдосконалені архітектури, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit). Ці моделі є підкласами RNN і вирішують проблему зникнення градієнтів за допомогою спеціальних механізмів, що дозволяють мережі зберігати інформацію про важливі моменти в тексті на довших відрізках. Вони використовують спеціальні клітинки пам’яті, які дозволяють зберігати або забувати інформацію залежно від її важливості для поточної задачі. Завдяки цьому, LSTM та GRU стали дуже популярними в задачах, що вимагають обробки послідовностей, зокрема для класифікації тексту, де важливі не лише окремі слова, але й їх порядок та взаємозв’язки [7-9]. Ще однією важливою архітектурою для класифікації тексту є згорткові нейронні мережі (Convolutional Neural Networks, CNN), які, хоча і традиційно використовуються для обробки зображень, також виявляються надзвичайно ефективними для текстової класифікації. У цьому випадку згорткові шари виконують фільтрацію тексту через різні фільтри, що дозволяє виділяти локальні патерни в тексті, такі як фрази або комбінації слів, які є важливими для класифікації. Завдяки здатності захоплювати локальні взаємозв’язки, CNN добре підходять для задач, де важливими є конкретні фрази або шаблони, що можуть свідчити про приналежність тексту до певної категорії.

У результаті глибинні нейронні мережі для класифікації тексту можуть бути побудовані на основі різних архітектур і варіантів їх поєднання, що дозволяє адаптувати модель до конкретних вимог задачі. Наприклад, можна комбінувати CNN та LSTM, щоб одночасно використовувати переваги обох типів мереж: згорткові шари допомагають виявляти важливі локальні патерни, а LSTM мережі – вивчати більш глобальні залежності в послідовностях слів.

Процес навчання глибинних нейронних мереж для класифікації тексту є складним і потребує великої кількості даних. Оскільки модель повинна навчатися на великому наборі текстів з відповідними етикетками (категоріями), важливо забезпечити достатньо різноманітні дані, щоб мережа могла правильно узагальнювати та робити правильні передбачення на нових, раніше не бачених текстах [9, 10]. Для цього також застосовуються різноманітні техніки регуляризації, такі як dropout, що допомагають уникнути перенавчання, а також різноманітні методи оптимізації, зокрема алгоритм зворотного поширення помилки, що використовується для оновлення ваг у мережі на кожному кроці навчання.

Незважаючи на свої переваги, глибинні нейронні мережі для класифікації тексту мають і деякі обмеження. Оскільки вони потребують великої кількості даних для навчання, це може бути проблемою, якщо доступні дані обмежені. Крім того, навчання таких мереж є обчислювально складним і потребує значних ресурсів, зокрема наявності потужних графічних процесорів (GPU). Однак, з розвитком технологій та доступністю великих обчислювальних потужностей, ці обмеження стають менш значними.

Завдяки своїй здатності ефективно обробляти складні патерни та взаємозв’язки в текстових даних, глибинні нейронні мережі стали стандартом для багатьох завдань класифікації тексту, зокрема для визначення тематики текстів, оцінки емоційного забарвлення (настрою), класифікації новин, фільтрації спаму та багатьох інших застосувань. Глибинні нейронні мережі є важливою складовою частиною сучасних систем обробки природної мови і продовжують розвиватися завдяки новим підходам і архітектурам, що дозволяють ще точніше моделювати складні мовні патерни.

## 1.5 Сучасна література

Стаття [1] розглядає проблему спаму в коротких текстових повідомленнях (SMS), який є небажаними повідомленнями, що надходять на мобільні пристрої користувачів від спамерів. Оператори мобільного зв'язку стурбовані цим явищем, оскільки спам-повідомлення призводять до незадоволення клієнтів через надмірну кількість небажаних повідомлень. Для боротьби з цією проблемою більшість постачальників послуг пропонують функцію активації «Не турбувати» (DND), яка обмежує кількість спаму. Однак повністю контролювати потік спаму неможливо, і його доставку не можна зупинити. Для вирішення цієї проблеми було проведено значні дослідження. Використання штучного інтелекту, зокрема глибокого навчання, дозволяє ефективно виявляти спам. У статті пропонується метод класифікації SMS-повідомлень як спам або не спам (ham) на основі моделі глибокого навчання. Для цього було застосовано рекурентну нейронну мережу (RNN), зокрема її варіант Long Short Term Memory (LSTM). Дослідження проводилось на наборі даних з вебсайту Grumbletext, що містить 425 коротких повідомлень, позначених як «Ham» або «spam». Модель LSTM ефективно класифікувала ці повідомлення, досягнувши точності 88,33% у процесі класифікації SMS-повідомлень як спам.

Стаття [2] описує нову систему для виявлення емоцій у текстових повідомленнях WhatsApp за допомогою методів некерованого машинного навчання. У системі використовуються методи обробки природної мови (NLP) для попередньої обробки текстових даних і алгоритм, заснований на Pointwise Mutual Information (PMI), для класифікації повідомлень за шістьма основними емоціями: радість, гнів, сум, страх, огиду та здивування. Ефективність моделі оцінюється на корпусі повідомлень з WhatsApp, що продемонструвало точність 92,75%. Цей підхід показує обнадійливі результати у роботі з неформальними мовними конструкціями та сленгом, характерними для

повідомлень WhatsApp, що відкриває великі перспективи для застосування в реальних додатках для виявлення емоцій у текстових даних.

Стаття [3] описує модель для класифікації SMS-повідомлень на три категорії: спам, рекламні повідомлення та не спам (ham), використовуючи індонезійські текстові повідомлення. Дослідження проводилось на основі 4 125 текстових повідомлень, модель була протестована на 1 260 повідомленнях. Для оцінки класифікаторів було використано метод 10-кратної крос-валідації, і результати показали, що серед найкращих моделей для багатокласової класифікації SMS є Random Forest (94,62%), Multinomial Logistic Regression (94,57%), Support Vector Machine (94,38%) та XGBoost (94,52%).

Стаття [4] описує розробку моделі для виявлення шахрайських повідомлень, яка складається з двох етапів: перший етап – це класифікація SMS-повідомлень за допомогою гібридної моделі, а другий – це аналіз URL-адрес. Основна увага в статті зосереджена на етапі класифікації, де була розроблена гібридна модель для покращення точності та точності класифікації. Гібридна модель використовує класифікатори Naive Bayes, Random Forest та Extra Tree Classifier. Кожен з них показав високі результати: точність Multinomial Naive Bayes (MNB) становила 95,9%, точність Random Forest (RF) – 97,0%, а точність Extra Tree Classifier (ETC) – 97,7%. У результаті аналізу набору даних гібридна модель досягла точності 96,86% і точності 99,366%. Важливо, що ця гібридна модель перевершила інші підходи машинного навчання за показниками ефективності.

Стаття [5] описує систему для класифікації текстових повідомлень з соціальних мереж, таких як Twitter, Facebook і Line, що повідомляють про ситуацію на дорогах. Організації, що займаються управлінням дорожнім рухом, моніторять ці повідомлення, щоб швидко реагувати на аварії. Дослідження зосереджене на створенні системи, яка класифікує повідомлення для визначення статусу руху, типу аварії, рівня її серйозності та GPS-локації, що дозволяє організаціям швидко вжити необхідних заходів. Система

використовує текстовий аналіз, враховуючи структуру та стиль повідомлень, а для класифікації типу повідомлення застосовується модель прихованих Маркових процесів. Джерела повідомлень включають Twitter, радіо JS100 і FM91.

Стаття [6] описує проблему спаму в коротких повідомленнях, що становить значну загрозу для користувачів мобільних телефонів, оскільки може бути використаний для кібернападів, таких як поширення шкідливих програм і фішинг. Традиційні методи боротьби зі спамом є малоефективними проти сучасних спамерів, тому важливо застосовувати інтелектуальний аналіз вмісту повідомлень для їх класифікації на спам (небажані) та хам (корисні). У статті розглядаються різні техніки штучного інтелекту (ШІ), що допомагають в аналізі таких повідомлень для їх фільтрації та класифікації. Дослідники натренували, перевірили та протестували сім таких методів на наборі даних SMS спаму, щоб визначити найкращу модель для створення системи на основі вмісту. Рекурентні нейронні мережі (RNN) показали найкращі результати (Тестова точність: 99,28%), тому запропонована система використовує RNN для класифікації. Також була розроблена веб-аплікація, де можна ввести одне SMS-повідомлення, і система класифікує його як спам або хам. Розроблену систему порівняли з існуючими, і вона показала значно кращі результати.

Стаття [7] описує використання чат-ботів у популярних онлайн-каналах, таких як Facebook Messenger і Line, які стали особливо важливими під час пандемії COVID-19 для швидкої взаємодії з клієнтами. Одним з прикладів є використання чат-бота в Офісі реєстратора університету Тхаммасат для відповіді на запитання студентів. Ключовим етапом у роботі чат-бота є розпізнавання наміру запитання, що надходить. Для цієї мети застосовувалась модель двосторонньої LSTM (Long Short-Term Memory), яка класифікує запитання на п’ять категорій намірів. Експериментальні результати показали, що модель досягає точності 0,80 на валідаційному наборі даних.

Стаття [8] розглядає використання платформ для обміну повідомленнями та онлайн-чатів, які пропонуються інтернет-компаніями, такими як постачальники транспорту, таксі та енергетичні компанії, для комунікації з клієнтами та збору цінної зворотної інформації. Автори пропонують моделі класифікації для аналізу думок користувачів, зосереджуючи увагу на стилістичних і сентиментальних характеристиках повідомлень. Дослідження показало, що існують чітко визначені сегменти в просторі стилометричних і сентиментальних ознак, що дозволяє створити класифікатор, який ефективно класифікує нових користувачів на основі їхнього стилю письма. Це дозволяє персоналізувати послуги і оптимізувати взаємодію з клієнтами. Розуміння стилометричних і сентиментальних патернів у чатах має практичне значення для поліпшення рекомендаторських систем, підвищення якості обслуговування клієнтів та кращого розуміння комунікативних динамік у чатах. Отримані результати відкривають нові можливості для подальшого розвитку в галузі обробки природної мови та аналізу сентиментів, а також є важливими для вдосконалення рекомендаторських систем та покращення якості обслуговування клієнтів на чат-платформах.

Стаття [9] досліджує використання комп’ютерно-медіованої комунікації (CMC) вищої освіти та пропонує методи для кращого розуміння поведінки студентів та сприяння їх навчанню через CMC. Метою дослідження було класифікувати повідомлення у навчальних групах на Facebook, створених як онлайн-дискусійні платформи. Для цього були запропоновані, навчені та протестовані різні моделі машинного навчання та глибокого навчання на основі корпусів з PTT – одного з відомих онлайн-форумів Тайваню. Результати показали, що рекурентна нейронна мережа (RNN) з використанням методу word to vector (W2V) для витягування ознак продемонструвала найкращу точність. Крім того, комбінація RNN та TF-IDF показала найвищу

кореляцію з людським кодуванням. Обговорюються можливості використання штучного інтелекту (ШІ) в освітньому контексті.

Стаття [10] описує проблему, з якою стикаються викладачі при оцінці великої кількості моделей, створених учнями в рамках навчання моделюванню станів. Оскільки моделі, створені учнями, часто мають різні описи, але можуть представляти однакову поведінку, важливо правильно класифікувати ці моделі за поведінковою схожістю. Це дозволяє викладачам ефективно надавати відгуки, перевіряючи лише одну модель з кожної групи, що сприяє швидкому і корисному зворотному зв'язку для учнів. Проте, до цього часу не існувало простого та автоматичного методу для класифікації таких моделей за поведінковою схожістю, що й є метою цього дослідження. Автори пропонують інструмент SMart-Learning for Classification (SML4C), який вимірює поведінкову схожість моделей на основі результатів їх тестування. Цей підхід не залежить від різноманітності описів. Попередня робота передбачала, що викладачі вручну створюють тестові кейси, тоді як запропонований метод автоматично генерує тестові кейси з моделей, створених учнями, що дозволяє легко класифікувати моделі станів, використовуючи тестові кейси, що відображають поведінку моделей. Ефективність методу була перевірена на 63 моделях, створених учнями для опису колаборативної поведінки двох мобільних роботів.

Стаття [11] описує важливість оцінки систем у науковій та інженерній практиці, а також пропонує комплексну модель оцінки, засновану на нелінійному алгоритмі класифікації Support Vector Machine (SVM). Спочатку досліджуються основи оцінки, зокрема система показників для оцінки, визначення ваг показників та стандартизація даних. Потім автори детально аналізують SVM, описуючи алгоритм нелінійної класифікації, зокрема п'ять поширених ядерних функцій, а також вдосконалюють традиційний SVM, враховуючи існуючі проблеми, такі як вибір моделі, перенавчання, нелінійність, прокляття вимірності та локальні мінімуми. Цей підхід

ефективно вирішує ці проблеми, одночасно покращуючи швидкість навчання і точність SVM, зменшуючи обсяг обчислень, та має високу адаптивність і практичну цінність.

Стаття [12] пропонує новий підхід до класифікації текстів, який поєднує модель Bert з байєсівською мережею для досягнення більш точної класифікації текстових даних. Модель Bert, яка є глибокою моделлю переднавчання, продемонструвала високі результати в 11 завданнях обробки природної мови (NLP) і має широкий спектр застосувань. Враховуючи величезну кількість неструктурованих даних у сфері управління народним добробутом, традиційні методи аналізу текстів стикаються з проблемами, пов’язаними з підвищеною складністю обчислень, тому важливо вибрати відповідні технології для класифікації текстів. У статті пропонується спочатку використовувати байєсівську мережу для попереднього класифікування текстів на дві категорії, що дозволяє звузити діапазон категорій кожного тексту, після чого модель Bert застосовується для більш точної класифікації в конкретні категорії. Поєднання цих двох методів значно знижує помилки, які можуть виникнути при використанні тільки одного з методів, що призводить до покращення точності класифікації текстів.

Стаття [13] пропонує нову модель класифікації текстів, яка поєднує BERT і механізм уваги для покращення представлення семантики текстів та вирішення проблем, що виникають при використанні існуючих методів на основі глибоких нейронних мереж, зокрема недостатнього виділення ознак, поганого семантичного представлення та чутливості до довжини речень. Модель автоматично виводить векторне представлення текстових даних за допомогою BERT, після чого додатково оптимізує семантичні ознаки за допомогою нейронної мережі з залишковими зв’язками. Механізм уваги адаптивно вивчає важливість різних ознак і призначає ваги для покращення точності класифікації. Експериментальні результати на публічно доступному наборі даних SST-2 показали, що модель досягає точності 92,66% та F1-оцінки

92,92%, що свідчить про високі результати класифікації та ефективність у вирішенні зазначених проблем.

Стаття [14] описує застосування методів обробки природної мови (NLP) для класифікації текстів у медичній галузі, зокрема для класифікації електронних медичних записів (EHR). Раніше використовувані алгоритми класифікації текстів, такі як наївний баєсівський класифікатор і рекурентні нейронні мережі (RNN), поступилися новітнім архітектурам на основі трансформерів, які використовують механізм уваги для кращого розуміння тексту та його контексту, демонструючи кращі результати в різних задачах NLP. У цій роботі пропонується нова інтегрована модель, яка поєднує трансформер на основі BERT з класифікатором XGBoost для досягнення кращих результатів класифікації текстів. Ця гібридна модель продемонструвала значне покращення точності класифікації порівняно з використанням лише моделі BERT, підвищивши точність тренувальної вибірки на 10–13% і тестової вибірки на 3%, що підтверджує її ефективність у медичних застосуваннях.

Стаття [15] розглядає важливість ефективних моделей для автоматичної класифікації великих обсягів текстів, оскільки кількість тексту, що генерується щохвилини, зростає дуже швидко. Тематичні моделі є одними з найпотужніших методів класифікації текстів, і багато наукових досліджень у цій галузі були опубліковані в наукових журналах. Однією з найбільш популярних тематичних моделей є Latent Dirichlet Allocation (LDA), яка використовується для класифікації текстів. Дослідники розробили різні моделі еволюції тем на основі LDA, щоб вирішувати конкретні проблеми, що виникають в застосуванні цих методів. Також вивчалися комбіновані моделі, що поєднують тематичні моделі з іншими алгоритмами для покращення результатів класифікації текстів. У статті детально розглядаються три категорії тематичних моделей для класифікації текстів, а також їхні переваги і недоліки в контексті текстової аналітики. Крім того, автори пояснюють процес

генерації документів та ілюструють графічну модель для кожної з цих моделей.

Стаття [16] пропонує гібридну модель глибокого навчання для аналізу та класифікації текстів, яка вирішує проблему традиційних методів, що не враховують важливість кожного слова в тексті під час видобутку ознак. Модель поєднує згорткові нейронні мережі (CNN), довготривалу короткочасну пам’ять (LSTM) та механізм уваги для ефективного видобутку ознак з енергетичних текстів. Спочатку CNN використовується для видобутку локальної інформації, після чого інтегрується семантика всього тексту, що допомагає виявити важливі шаблони та зв’язки. Потім LSTM витягує контекстуальні ознаки, які покращуються завдяки механізму уваги, що дозволяє моделі фокусуватися на важливих словах і фразах. Нарешті, вихід LSTM з увагою об’єднується з виходом CNN для створення більш повного представлення ознак тексту. Експериментальні результати на публічних наборах даних показали, що ця модель перевищує за точністю та ефективністю LSTM, CNN та їх покращені версії в класифікації текстів, що дозволяє краще ідентифікувати важливу інформацію з енергетичних текстів.

## 1.6 Висновки до першого розділу

У розділі представлено аналіз чотирьох основних методів машинного навчання, що використовуються для класифікації тексту, зокрема, наївний баєсів класифікатор, метод опорних векторів (SVM), дерева рішень та глибинні нейронні мережі.

Зміст розділу відображає поступове ускладнення та розвиток методів класифікації тексту. Наприклад, наївний баєсів класифікатор є простим і класичним методом, але в міру розвитку дослідження з’являються більш складні й потужніші методи, як-от метод опорних векторів, дерева рішень, а також глибинні нейронні мережі, які застосовуються для вирішення

складніших завдань, таких як обробка великих обсягів тексту або врахування контекстуальних взаємозв’язків між словами.

Поглиблений аналіз кожного методу в окремому підрозділі дозволяє зрозуміти, як кожен з методів працює в контексті класифікації тексту, які існують алгоритмічні підходи, а також можливі обмеження, з якими вони стикаються.

Оскільки класифікація тексту є важливою задачею в багатьох галузях, таких як обробка природної мови, інформаційний пошук, фільтрація спаму та аналіз настроїв, це дослідження є актуальним і вносить вклад у розробку нових, ефективних методів для автоматичної обробки текстових даних. Кожен з описаних методів має своє місце в сучасних системах, що займаються класифікацією текстових повідомлень.

Виходячи з аналізу цих методів, можна сформулювати завдання дослідження, яке полягає у виборі або комбінуванні найкращих технік для досягнення найвищої точності класифікації тексту в конкретному контексті. Постановка задачі включатиме експериментальне порівняння різних підходів, визначення їх сильних і слабких сторін, а також пошук оптимальних параметрів для досягнення найкращих результатів.

Таким чином, розділ дає чітке уявлення про існуючі методи класифікації тексту, їх можливості та обмеження, що є основою для подальшої постановки задачі та пошуку ефективних рішень для застосування в реальних сценаріях класифікації текстових повідомлень.

# РОЗДІЛ 2. ДОСЛІДЖЕННЯ АЛГОРИТМІВ ПОПЕРЕДНЬОЇ ОБРОБКИ ТЕКСТОВИХ ДАНИХ

## 2.1 Видалення стоп-слів

Видалення стоп-слів є важливим етапом у процесі попередньої обробки текстових даних. Цей крок є невід'ємною частиною більшості сучасних методів обробки природної мови, зокрема у таких галузях, як машинне навчання, інформаційний пошук, обробка текстових корпусів і побудова моделей на основі текстових даних [9]. Стоп-слова – це слова, які зустрічаються дуже часто у текстах і, як правило, не несуть суттєвої інформаційної цінності в контексті задач, пов'язаних з аналізом тексту. Вони можуть включати сполучники, прийменники, артиклі, модальні дієслова, займенники та інші частини мови, які часто використовуються в розмовній мові, але не допомагають у формуванні значення тексту чи класифікації інформації.

Процес видалення стоп-слів є складовою частиною підготовки тексту до подальшої обробки та аналізу. У більшості випадків видалення стоп-слів проводиться для зменшення обсягу даних, що обробляються моделями машинного навчання, оскільки ці слова не несуть важливої інформації для вирішення задач, таких як класифікація текстів, аналіз настроїв або пошук релевантних документів. Водночас важливо зазначити, що з усіх слів у тексті саме стоп-слова складають найбільшу частину текстових корпусів, що робить їх видалення надзвичайно важливим для зменшення вимог до пам'яті та часу обчислень при роботі з великими наборами даних [8, 9]. Під час обробки тексту одним з перших етапів є очищення тексту від різноманітних шумів, таких як непотрібні символи, цифри, знаки пунктуації та інші елементи, які не мають значення для задачі. Це забезпечує, з одного боку, спрощення структури тексту, а з іншого – збереження основних смислових компонентів, які будуть

використані для подальшого аналізу. Однак навіть після цього етапу текст залишається надто насиченим словом, що значно ускладнює подальші операції, пов’язані з обробкою та аналізом даних.

Стоп-слова є свого роду «шумом», який створює додаткову обробку, що не надає суттєвої інформації про зміст тексту. Це можуть бути такі слова, як «і», «або», «але», «для», «це» тощо, які в більшості випадків не дають суттєвого внеску у розуміння основного змісту. Наприклад, у процесі класифікації текстів стоп-слова можуть заважати, бо їх часто надмірно багато, що може призвести до того, що модель почне робити акценти на випадкових, неважливих аспектах тексту. Тому видалення стоп-слів є важливим етапом для зменшення кількості словникових одиниць і зниження розмірності задачі. Існує кілька підходів до видалення стоп-слів, які можуть бути застосовані в залежності від конкретних умов. Один із найбільш поширених методів – це використання попередньо підготовлених списків стоп-слів. Такий підхід має свої переваги, оскільки він дозволяє швидко та ефективно фільтрувати заздалегідь визначені стоп-слова без необхідності виведення додаткових алгоритмів [10, 11]. Водночас цей метод має свої обмеження, оскільки не завжди можливо охопити всі випадки. Наприклад, певні слова, що можуть бути стоп-слів у одному контексті, в іншому контексті можуть бути важливими для розуміння змісту. У таких випадках доцільно застосовувати більш гнучкі методи, які враховують контекст.

Іншим підходом є використання спеціальних алгоритмів для визначення стоп-слів. Це можуть бути алгоритми, що базуються на статистичному аналізі частоти зустрічей слів у текстах, наприклад, методи побудови частотних словників або застосування статистичних моделей, таких як TF-IDF (термінно-частотний зворотний документальний фактор). Ці методи дозволяють автоматично виявляти найбільш вживані слова в тексті, які часто є стоп-словами, та видаляти їх із тексту, залишаючи тільки значущі слова, що несуть інформацію. Алгоритми можуть працювати на основі порогових значень

частоти слів, що дозволяє знизити ймовірність втратити важливі терміни, які можуть зустрічатися рідше, але при цьому бути значущими для контексту.

Крім того, у деяких випадках застосовують лексико-семантичні методи для визначення стоп-слів. Ці методи базуються на аналізі значення слів і їхніх взаємозв'язків у контексті [12]. Завдяки таким методам можна не лише видаляти часто зустрічаються слова, але й зберігати слова, які в іншому випадку могли б бути неправильно інтерпретовані як стоп-слова. Наприклад, для англійської мови одним із найбільш розповсюджених методів є використання лексичних баз даних, таких як WordNet, для визначення частоти й значення слів. Важливо зазначити, що видалення стоп-слів потребує додаткової уваги в контексті специфіки мови, що обробляється. Для різних мов можуть бути різні набори стоп-слів, і їхній вибір може залежати від багатьох факторів, таких як культурні особливості, тип тексту (наприклад, науковий, технічний, художній) та навіть жанрові відмінності. Наприклад, у технічних текстах певні слова, які є стоп-слівами в загальних текстах, можуть мати важливе значення для контексту і тому не повинні бути видалені. Тому важливо підходити до вибору стоп-слів з урахуванням специфіки кожного конкретного завдання.

Деякі сучасні підходи до обробки природної мови застосовують більш складні методи, що дозволяють автоматично знаходити та визначати стоп-слова, враховуючи контекст їхнього використання. Наприклад, в рамках використання глибоких нейронних мереж та трансформерних моделей (таких як BERT чи GPT) можна навчити модель враховувати контекст, що дозволяє не лише видаляти стоп-слова, а й розпізнавати важливі для змісту слова, які в іншому випадку могли б бути трактовані як незначущі [12, 13].

При цьому слід пам'ятати, що надмірне видалення слів може призвести до втрати важливої інформації, яка була б корисною для виконання конкретного завдання. Наприклад, у деяких випадках навіть частинки або слова, що вважаються стоп-слівами, можуть відігравати важливу роль у

передаванні змісту. Тому важливо ретельно підходити до процесу видалення стоп-слів, не застосовуючи цей крок як універсальний для всіх типів текстів.

Процес видалення стоп-слів не є самодостатнім і має бути частиною комплексної стратегії попередньої обробки текстових даних. Крім видалення стоп-слів, важливими є й інші етапи, такі як лематизація (переведення слів до їхньої базової форми), токенізація (розбиття тексту на окремі елементи), а також нормалізація тексту (наприклад, перетворення всіх слів до одного регістру) [12-14]. Всі ці етапи взаємодіють між собою, забезпечуючи якісну підготовку даних для подальшого аналізу та навчання моделей машинного навчання. Таким чином, видалення стоп-слів є важливим етапом у попередній обробці текстових даних, який дозволяє знизити шум та зменшити розмірність даних, що обробляються. Однак важливо, щоб цей процес був добре налаштований і адаптований до конкретних завдань і особливостей мови, з якою працює система. Використання різноманітних методів і технік допомагає досягти оптимальних результатів, зберігаючи при цьому важливу інформацію в текстах.

## 2.2 Стемінг та лематизація

Стемінг та лематизація є двома важливими методами попередньої обробки текстових даних, що активно використовуються в області обробки природної мови (NLP, Natural Language Processing). Обидва ці методи мають на меті знизити варіативність слів, що зустрічаються у текстах, шляхом приведення їх до певних стандартних форм [7, 9]. Ці процеси є невід’ємною частиною багатьох систем обробки тексту, зокрема для класифікації, пошуку, аналізу настроїв, машинного перекладу та багатьох інших завдань, що потребують роботи з великими обсягами текстової інформації. Відмінності між стемінгом та лематизацією, а також їхні переваги та недоліки, мають

важливе значення для вибору оптимального методу в залежності від задачі, яку необхідно розв’язати.

Стемінг є процесом, який полягає в зниженні варіативності слів до їхніх кореневих форм, або так званих «стем». Стемінг працює на основі алгоритмічних правил, що визначають, як зменшити слова до їхніх кореневих частин. Це дозволяє звести до мінімуму кількість унікальних слів, з якими має працювати система, і, як наслідок, зменшити складність обробки тексту. Приміром, стемінг може перетворити слова «running», «runner», «ran» в один корінь «run». Однак варто зазначити, що стемінг часто не враховує контексту слів і може призвести до втрати граматичної чи семантичної точності. Так, слово «better» може бути перетворено в «bet», що є коренем, але без контексту та додаткової обробки таке слово може втратити свій сенс. Тому стемінг є більш агресивним методом і застосовується в тих випадках, коли важлива саме швидкість обробки та зниження обсягу даних, але не настільки важлива точність граматичних чи лексичних форм [10]. Лематизація, у свою чергу, є більш складним і точним процесом. Вона передбачає приведення слів до їхньої базової форми, або леми. Лема – це граматично правильна форма слова, яка є представником всіх його різних варіантів. Наприклад, для слів «running», «ran», «runs» лемою буде «run». Лематизація враховує частину мови слова, а також його контекст, що дозволяє зберігати більше семантичної точності. Тобто, лематизація не просто обрізає кінцівки слів, а намагається визначити їх правильну форму, враховуючи граматичні правила мови. Це дозволяє уникнути багатьох помилок, які можуть виникнути при застосуванні стемінгу, однак лематизація є більш ресурсозатратним процесом, оскільки вимагає доступу до лексичних баз даних, таких як WordNet для англійської мови або інші словники для інших мов.

Вибір між стемінгом та лематизацією залежить від конкретної задачі та ресурсів, що доступні для обробки тексту. Стемінг застосовують тоді, коли потрібно швидко і ефективно знизити розмірність даних, не звертаючи увагу

на точність кожної форми слова. Це може бути корисно в задачах, де не критична семантична точність, наприклад, у задачах класифікації тексту, де важливо визначити наявність певних тем або категорій, а не точне значення слів. Однак для більш складних задач, які вимагають збереження контексту та точності змісту, таких як аналіз настроїв або машинний переклад, лематизація є більш відповідним методом. Вона дозволяє зберегти правильні граматичні форми та семантичну точність, що є необхідним для більш глибокого аналізу тексту [9, 12]. Ще однією суттєвою різницею між стемінгом та лематизацією є те, як вони обробляють неправильні форми слів. У процесі стемінгу часто використовується набір правил для обрізання кінцівок слів, що може призвести до некоректних результатів для певних слів. Наприклад, слово «goose» може бути перетворене в «goos», оскільки стемінг не враховує різні форми слова. Лематизація ж намагається зберегти граматичну правильність, тому для «goose» вона залишить форму «goose», що є лемою цього слова. Водночас, лематизація може вимагати більше обчислювальних ресурсів, оскільки для кожного слова необхідно здійснювати пошук в лексичній базі даних або застосовувати більш складні граматичні правила. Однією з ключових проблем, з якою стикаються як стемінг, так і лематизація, є їх застосування до слів, що мають кілька значень або контекстуально змінюються в залежності від того, в якому оточенні вони знаходяться. Для стемінгу це може бути ще однією складністю, оскільки він не враховує контекст і може перетворити слово на корінь, який не підходить для конкретного випадку. Лематизація зазвичай ефективніше справляється з цією проблемою, оскільки вона враховує частину мови та контекст. Проте навіть лематизація може мати труднощі при обробці багатозначних слів або слів, що мають неправильні форми [15].

Стемінг і лематизація часто застосовуються як частина загальної стратегії попередньої обробки текстових даних. Перед тим, як застосувати ці методи, текст часто проходить через етапи очищення, токенізації, видалення

стоп-слів та інших процедур, які готують його до подальшого аналізу або навчання моделей машинного навчання. Це дозволяє покращити точність моделей, зменшити обсяг даних, з яким працює система, і підвищити ефективність обробки тексту. Однак в залежності від конкретних умов, таких як тип тексту або мета аналізу, можна вибирати один з методів або навіть комбінувати їх для досягнення кращих результатів.

Загалом, стемінг і лематизація є важливими етапами в процесі попередньої обробки текстових даних, і правильний вибір між ними залежить від специфіки задачі. Обидва методи допомагають знижувати складність тексту та забезпечують ефективну підготовку даних для подальшої обробки. Тим не менш, лематизація зазвичай дає кращі результати в задачах, що вимагають більшої точності та контекстуального розуміння, в той час як стемінг є більш швидким і менш ресурсозатратним методом, що підходить для загальних задач аналізу тексту.

## 2.3 Векторизація тексту

Векторизація тексту є одним із основних етапів попередньої обробки текстових даних, що використовується для перетворення текстових об’єктів у числові представлення, які можуть бути використані для подальшого аналізу або побудови моделей машинного навчання. Процес векторизації полягає у перетворенні тексту в так звану векторну форму, де кожен текстовий об’єкт (наприклад, документ або речення) представляється числовими векторами [16]. Це необхідно, оскільки для більшості алгоритмів машинного навчання та глибокого навчання важливо працювати з числовими даними, а не з текстовими рядками. Один з найбільш поширених підходів до векторизації тексту включає методи, такі як TF-IDF (Term Frequency-Inverse Document Frequency) і Word2Vec. Ці два методи мають свої унікальні особливості, які дозволяють з різних точок зору підходити до задачі перетворення тексту в

числові вектори, і кожен з них має свої переваги та недоліки в залежності від специфіки задачі.

TF-IDF є статистичним методом, що вимірює важливість терміна у документі відносно його появи в загальному корпусі текстів. TF-IDF поєднує дві важливі характеристики слова в контексті текстового корпусу. Першою є частота терміна (TF, Term Frequency), яка визначає, скільки разів конкретне слово зустрічається в документі, що аналізується. Друга характеристика – це зворотна частота документа (IDF, Inverse Document Frequency), яка вимірює, наскільки рідко або часто термін з'являється в інших документах корпусу. Цей підхід базується на припущенні, що слова, які з'являються часто в одному документі, але рідко в інших, є важливими для цього конкретного документу і можуть краще характеризувати його зміст. З іншого боку, слова, що зустрічаються в багатьох документах, такі як сполучники, прийменники та інші загальні терміни, менш важливі для визначення специфіки тексту. Метод TF-IDF дозволяє значно зменшити вагу таких загальних слів, зберігаючи при цьому значущі терміни.

Розрахунок TF здійснюється як відношення кількості разів, коли слово з'являється в конкретному документі, до загальної кількості слів у цьому документі. Водночас, IDF визначається як логарифм зворотного відношення загальної кількості документів до кількості документів, в яких зустрічається дане слово [16]. Тому слову, яке з'являється в багатьох документах, буде присвоєно низьке значення IDF, а слову, яке зустрічається лише в одному або кількох документах, присвоєно високе значення. В результаті, коли ці два значення множаться, отримується показник, який дозволяє оцінити важливість слова для конкретного документа в контексті всього корпусу. Один із головних недоліків методу TF-IDF полягає в тому, що він не враховує контексту слова. Векторизація за допомогою TF-IDF фокусується на окремих словах, але не бере до уваги взаємозв'язки між словами, які можуть мати важливе значення для розуміння контексту. Наприклад, дві фрази, які містять

однакові слова, можуть мати різний сенс, і TF-IDF не завжди здатен вловити ці тонкощі, оскільки кожне слово оцінюється окремо, без урахування його контексту в реченні або документі. Тому TF-IDF є ефективним для задач, де важливий зміст окремих слів, таких як класифікація документів, але його можливості обмежуються в тих випадках, коли необхідно врахувати контекст або синонімію.

Word2Vec, з іншого боку, є більш складним методом, який базується на концепції векторних представлень слів. Цей метод був розроблений для того, щоб вловити більш глибокі семантичні зв'язки між словами. Він перетворює слова на вектори в багатовимірному просторі, де подібні слова мають схожі вектори. Word2Vec використовує нейронні мережі для навчання моделей, що дозволяють передбачити ймовірність появи слова в контексті інших слів, що допомагає створити багатовимірне представлення, яке відображає не тільки частоту слів, а й їх контекстуальну схожість. В основі Word2Vec лежить два підходи: Continuous Bag of Words (CBOW) та Skip-gram. Підхід CBOW передбачає передбачення поточного слова на основі контекстних слів, що оточують його в тексті, тоді як метод Skip-gram передбачає передбачення контекстних слів на основі поточного слова [13]. Обидва ці підходи дозволяють створювати ефективні векторні представлення слів, які можуть бути використані для різноманітних завдань, таких як класифікація, кластеризація, пошук схожих слів або навіть генерація тексту.

Перевага Word2Vec полягає в тому, що він здатен зберігати семантичні відношення між словами, наприклад, різницю між словами «король» і «королева» або «чоловік» і «жінка». Ці семантичні зв'язки зберігаються завдяки векторному представленню, де відстань між векторами подібних слів буде невеликою, а відстань між векторами різних слів буде більшою. Крім того, Word2Vec також дозволяє знаходити синоніми, подібні за значенням слова, оскільки він враховує контекст і семантику слів. Однак одним із недоліків Word2Vec є те, що цей метод потребує великого обсягу навчальних

даних для досягнення високої якості векторних представлень, а також великих обчислювальних ресурсів для навчання моделей.

Загалом, метод TF-IDF є більш простим та швидким способом векторизації тексту, що дозволяє працювати з текстами, де важливі частоти слів, однак він має обмеження щодо контексту та семантики. Word2Vec же є потужнішим методом, який дозволяє враховувати контекст та семантичні зв'язки між словами, однак він є більш складним та вимагає значних обчислювальних потужностей для тренування моделей [14]. Кожен з цих методів має свої переваги і недоліки, тому вибір між ними залежить від специфіки задачі та доступних ресурсів. Одним із важливих аспектів, які необхідно враховувати при виборі методу векторизації, є розмір і специфіка текстового корпусу. Якщо корпус великий, містить велику кількість документів і є можливість провести навчання на великих даних, то Word2Vec може бути ефективним вибором. Якщо ж потрібна швидка векторизація для невеликих текстів або для конкретних завдань, де важливі частоти слів, то TF-IDF буде більш оптимальним. Крім того, можливе комбінування цих методів, коли спочатку застосовується TF-IDF для попередньої векторизації, а потім для більш глибокого аналізу використовуються вектори слів, отримані за допомогою Word2Vec.

## 2.4 Точність, повнота та F1-міра для оцінки ефективності моделей класифікації тексту

Точність, повнота та F1-міра є одними з найбільш поширених і важливих метрик для оцінки ефективності моделей класифікації тексту. Ці показники дозволяють з різних аспектів оцінити, як добре модель справляється із задачами класифікації, особливо в контексті таких задач, як класифікація документів, виявлення спаму, аналіз настроїв або виявлення тематичних категорій [10]. Кожна з цих метрик надає інформацію про різні аспекти роботи

моделі, і разом вони дозволяють отримати більш повну картину її ефективності, враховуючи як позитивні, так і негативні наслідки класифікації. Однак для того, щоб зрозуміти важливість та застосування цих метрик, необхідно розглянути, що стоїть за кожною з них і як вони пов’язані між собою, а також чому важливо використовувати їх у комплексі для оцінки роботи класифікаторів текстів.

Перш за все, потрібно звернути увагу на поняття «контингентної матриці» (confusion matrix), яка є основою для обчислення всіх трьох метрик. Контингентна матриця є таблицею, що відображає співвідношення між передбаченими і фактичними класами в задачі класифікації. У найпростішому випадку для бінарної класифікації, де є два можливих класи – позитивний і негативний – контингентна матриця містить чотири елементи: істинно позитивні (True Positive, TP), хибно позитивні (False Positive, FP), істинно негативні (True Negative, TN) та хибно негативні (False Negative, FN). Істинно позитивні – це випадки, коли модель правильно класифікує позитивний клас; хибно позитивні – це випадки, коли модель неправильно класифікує негативний клас як позитивний; істинно негативні – це випадки, коли модель правильно класифікує негативний клас; хибно негативні – це випадки, коли модель неправильно класифікує позитивний клас як негативний.

Точність (accuracy) є найбільш очевидною і простою метрикою для оцінки моделі класифікації. Вона визначає відсоток правильних класифікацій серед усіх спостережень. Точність є важливою метрикою, оскільки вона дає загальну оцінку роботи моделі, показуючи, наскільки вона правильно класифікує як позитивні, так і негативні приклади [12]. Однак ця метрика може бути недостатньо інформативною в разі, коли в задачі класифікації є сильний дисбаланс між класами, тобто один клас зустрічається набагато частіше, ніж інший. У таких випадках модель може досягати високої точності, просто класифікуючи всі приклади як належні до більшості класу. Наприклад, в задачі виявлення рідкісних захворювань, де більшість пацієнтів здорові, модель, яка

завжди передбачає «здоровий» клас, може досягти високої точності, навіть не виявляючи жодного випадку захворювання. Тому точність не завжди є найкращим показником ефективності класифікації, особливо в умовах дисбалансу класів.

Повнота (recall) є метрикою, що вимірює здатність моделі правильно виявляти позитивні приклади серед усіх фактичних позитивних випадків. Вона визначає, яка частина всіх позитивних об’єктів була правильно класифікована як позитивна. Повнота є важливою метрикою в тих випадках, коли важливо мінімізувати пропуск позитивних прикладів, навіть якщо це може призвести до зростання хибнопозитивних результатів. Повнота дає уявлення про те, скільки з усіх можливих позитивних прикладів модель змогла правильно ідентифікувати [14]. Це важливо в таких ситуаціях, де пропуск позитивних випадків може мати серйозні наслідки. Наприклад, в медицині, якщо модель для діагностики хвороби має низьку повноту, це означає, що вона пропускає багато пацієнтів із захворюванням, що може призвести до важких наслідків для здоров’я. Однак, подібно до точності, повнота сама по собі не дає повної картини роботи моделі, оскільки вона не враховує кількість хибнопозитивних класифікацій. Тому повнота часто використовується разом із іншими метриками, такими як точність та F1-міра, для оцінки балансу між пропущеними позитивними випадками та хибнопозитивними.

F1-міра є комбінованою метрикою, що об’єднує точність і повноту, прагнучи забезпечити більш збалансовану оцінку ефективності моделі. Вона є середнім гармонійним значенням точності та повноти, що дозволяє досягти компромісу між двома цими метриками. F1-міра надає більш збалансовану картину ефективності моделі в ситуаціях, коли точність і повнота взаємно компенсують одна одну, а не домінують одна над іншою. F1-міра є корисною метрикою, коли потрібно знайти баланс між точністю та повнотою, і вона особливо важлива в задачах, де одна з цих метрик не може бути єдиною для оцінки моделі. Наприклад, в задачах, де важливо як мінімізувати пропуски

позитивних випадків, так і контролювати кількість хибнопозитивних результатів, F1-міра дає більш точну картину ефективності. Вона використовується в багатьох практичних задачах, таких як класифікація текстів, виявлення спаму, аналіз настроїв та інші завдання, де важливе поєднання чутливості і точності [15]. Однак варто зазначити, що F1-міра може не завжди бути оптимальною метрикою, особливо в разі сильно нерівномірного розподілу класів. В таких випадках може бути корисно використовувати такі метрики, як AUC-ROC (Area Under the Receiver Operating Characteristic Curve) або Precision-Recall Curve, які надають додаткову інформацію про ефективність класифікації в умовах дисбалансу класів.

Таким чином, точність, повнота та F1-міра є основними метриками для оцінки ефективності моделей класифікації тексту, кожна з яких дає важливу інформацію про різні аспекти роботи моделі. Вибір між цими метриками залежить від конкретної задачі та пріоритетів. У деяких випадках, коли важливе мінімізація хибнопозитивних результатів або пропусків, F1-міра є оптимальним вибором. В інших випадках, коли є потреба в максимальному збереженні позитивних випадків, повнота може бути більш важливою. Однак у будь-якому випадку, для точного оцінювання ефективності моделі класифікації, ці метрики повинні використовуватися в комбінації, що дозволяє забезпечити збалансовану оцінку роботи класифікатора.

## 2.5 Крос-валідація та її роль у тестуванні моделей

Крос-валідація є однією з найбільш важливих і поширених технік у процесі тестування і оцінки ефективності моделей машинного навчання. Вона має важливе значення для точності та надійності результатів, оскільки дозволяє уникнути проблем, пов’язаних з оверфіттингом і підвищує загальну здатність моделі до генералізації [16]. Крос-валідація, зокрема, виступає як

метод, що дозволяє ефективно використовувати обмежений обсяг даних для перевірки різних моделей і їх гіперпараметрів, мінімізуючи при цьому ймовірність помилкових оцінок їх продуктивності. Це особливо важливо в умовах обмежених ресурсів або при роботі з малими наборами даних, де будь-яка похибка в оцінці якості моделі може призвести до значних недоліків у подальшій роботі. Крос-валідація є універсальним інструментом, що дозволяє забезпечити більш стабільні та об’єктивні результати в процесі побудови моделей для класифікації, регресії та інших типів задач машинного навчання.

Основна мета крос-валідації полягає в тому, щоб максимально точно оцінити ефективність моделі, забезпечивши її навчання і тестування на різних підмножинах даних. Крос-валідація дозволяє мінімізувати ризик того, що модель буде недостатньо узагальненою або, навпаки, перенавченою на конкретному наборі даних. Вона є критично важливою, коли існує потреба в об’єктивному порівнянні кількох моделей або в налаштуванні гіперпараметрів моделі [10]. Ця техніка також має важливе значення в ситуаціях, коли дані є обмеженими і важливо використовувати кожен доступний елемент для навчання та тестування моделі.

Процес крос-валідації включає поділ всього набору даних на кілька підмножин (часто на «k» частин), де кожен з піднаборів по черзі використовуються для тестування моделі, а решта частин – для її навчання. Такий підхід дозволяє використовувати всі дані для обох етапів, що дає більш точну і стабільну оцінку моделі, ніж якщо б тестування проводилося тільки на одному підмножині даних. У найпростішому випадку, при використанні техніки k-фолдової крос-валідації, весь набір даних ділиться на рівні підмножини, з яких одна служить для тестування, а інші – для навчання моделі. Цей процес повторюється кілька разів, при кожному новому циклі вибираючи для тестування новий піднабір даних. У результаті, кожен з піднаборів даних буде використаний як для навчання, так і для тестування, що дозволяє отримати більш точну оцінку здатності моделі до генералізації.

Результатом крос-валідації є середнє значення метрики оцінки моделі (такої як точність, повнота, F1-міра тощо) за всіма фолдами, що дозволяє отримати більш надійну і стабільну оцінку її ефективності.

Крос-валідація дає можливість не лише оцінити якість моделі, але й дозволяє виявити можливі проблеми, пов’язані з підбором гіперпараметрів. Важливим аспектом є те, що крос-валідація дозволяє уникнути одного з основних недоліків стандартного підходу до оцінки моделей, коли дані діляться на лише один навчальний та один тестовий набори. Такий підхід може призвести до помилкових результатів, якщо тестовий набір не відображає загальний розподіл даних або містить якісь специфічні характеристики, які відсутні в навчальному наборі [7]. Крос-валідація, поділяючи дані на кілька частин, дає можливість мінімізувати цей ризик, адже кожен елемент даних буде використаний для навчання та тестування.

Однією з основних переваг крос-валідації є її здатність працювати навіть з малими наборами даних. Оскільки кожен елемент даних кілька разів використовується для навчання і тестування, це дозволяє максимізувати використання наявних даних, що є важливим у багатьох реальних задачах, де великий набір даних може бути недоступний або важким для збору. Крос-валідація також допомагає забезпечити більш надійні результати порівняно з одноразовим поділом на навчальну та тестову вибірки, оскільки дає можливість оцінити модель на різних підмножинах даних, що зменшує ймовірність того, що результати будуть завищені або занижені через особливості тестового набору.

Незважаючи на численні переваги, крос-валідація має і деякі недоліки, які потрібно враховувати при її застосуванні. Одним із головних недоліків є те, що процес крос-валідації може бути значно більш обчислювально складним і витратним порівняно з традиційними методами оцінки, особливо коли мається на увазі велика кількість даних і складні моделі машинного навчання. Кожен фолд потребує окремого навчання моделі, що може

потребувати значних обчислювальних ресурсів, особливо в разі використання великих даних або складних алгоритмів, таких як нейронні мережі [1, 3]. У зв'язку з цим, час, необхідний для виконання крос-валідації, може бути значним, що обмежує її застосування в умовах обмежених ресурсів або часу. Проте цей недолік може бути частково подоланий завдяки використанню ефективних технік, таких як паралельне обчислення, коли фолди обробляються одночасно на кількох обчислювальних потужностях. Існують також інші варіанти крос-валідації, які можуть бути більш підходящими для різних типів задач і особливостей даних. Одним із таких варіантів є «leave-one-out» крос-валідація (LOO), яка є крайнім випадком k-фолдової крос-валідації, де k дорівнює кількості елементів у наборі даних. Це означає, що для кожного елемента даних модель тренується на всіх інших елементах, а перевіряється на одному конкретному елементі. Це дає дуже точну оцінку моделі, але є дуже ресурсомістким для великих наборів даних, оскільки кожен елемент даних потребує окремого тренування моделі. Крім того, існують і такі варіанти крос-валідації, як стратифікована крос-валідація, яка передбачає, що кожен фолд має відповідну пропорцію класів, що дозволяє більш точно оцінити модель у задачах із сильно дисбалансованими класами [13]. Це важливо, оскільки стандартна крос-валідація може призвести до того, що деякі фолди міститимуть недостатньо прикладів рідкісного класу, що погіршить результати класифікації.

Отже, крос-валідація є потужним інструментом для оцінки ефективності моделей машинного навчання. Вона дозволяє забезпечити об'єктивну та стабільну оцінку, мінімізуючи ймовірність помилок, пов'язаних з неадекватним розподілом даних між навчальною та тестовою вибіркою. З її допомогою можна отримати більш точні та узагальнені результати, що забезпечують високу надійність і стійкість моделей. Хоча крос-валідація може бути обчислювально складною та вимагати значних ресурсів, її переваги для

точності і надійності оцінки моделей роблять її незамінною технікою в арсеналі машинного навчання.

## 2.6 Методи оптимізації та налаштування моделей

Методи оптимізації та налаштування моделей є одними з найбільш важливих аспектів у процесі побудови машинного навчання, оскільки саме ці етапи дозволяють досягти найкращих результатів в реальних задачах. Моделі машинного навчання, навіть якщо вони мають складні алгоритми та велику кількість параметрів, можуть залишатися неефективними, якщо не пройти через етап налаштування, який включає в себе оптимізацію параметрів і гіперпараметрів [10, 11].

Процес оптимізації моделі в машинному навчанні не обмежується лише пошуком оптимальних параметрів для конкретної задачі, а також охоплює пошук найкращих варіантів для функцій втрат, метрик оцінки та вибору найбільш підходящих методів навчання. Від того, наскільки добре налаштована модель, залежить її здатність до узагальнення на нові, невідомі дані, що в кінцевому підсумку визначає її ефективність у реальних умовах. Тому налаштування моделей є невід’ємною частиною процесу машинного навчання і є ключем до досягнення високих результатів на практиці. Процес оптимізації передбачає використання ряду методів, які надають можливість обирати найкращі гіперпараметри, навчати модель з мінімальною кількістю помилок, а також покращувати її здатність до генералізації.

Оптимізація моделі машинного навчання зазвичай здійснюється через процес налаштування параметрів моделі, що включає вибір та корекцію гіперпараметрів, які визначають процес навчання і складність моделі. Для кожного типу моделі, будь то лінійна регресія, дерева рішень, підтримувальні векторні машини або нейронні мережі, існують певні налаштування, які значно впливають на її ефективність. Важливою характеристикою процесу

налаштування є баланс між переобученням (overfitting) і недонавчанням (underfitting) моделі. Переобучення відбувається, коли модель надто точно вивчає шукані залежності на навчальному наборі даних, включаючи випадкові шуми, що призводить до погіршення її здатності до генералізації на нові дані. Недонавчання, в свою чергу, має місце, коли модель не здатна захопити основні закономірності в даних і має низьку точність на обох наборах – навчальному та тестовому [15, 16]. У зв'язку з цим налаштування моделі передбачає пошук такого оптимального рівня складності, при якому модель буде максимально точною, але без втрати здатності до узагальнення.

Одним з основних методів оптимізації моделі є налаштування гіперпараметрів. Гіперпараметри – це параметри, які не є результатом навчання моделі, а задаються до процесу навчання, і від їх вибору залежить поведінка алгоритму. Наприклад, для методів, що використовують градієнтний спуск, таким параметром є швидкість навчання (learning rate), для дерев рішень – максимальна глибина дерева, для нейронних мереж – кількість шарів та нейронів у кожному шарі. Налаштування гіперпараметрів є однією з найскладніших задач, оскільки правильний вибір параметрів може істотно вплинути на ефективність моделі, в той час як некоректне налаштування призведе до погіршення результатів.

Для ефективного налаштування гіперпараметрів розроблено кілька підходів. Одним з них є методи пошуку по сітці (grid search), коли перевіряється комбінаторика всіх можливих значень гіперпараметрів. Цей метод є досить простим і зрозумілим, але може бути дуже обчислювально затратним, особливо коли кількість гіперпараметрів велика або можливі значення кожного з них мають широкий діапазон. Альтернативою є метод випадкового пошуку (random search), який передбачає випадкову перевірку гіперпараметрів з певних діапазонів. Цей метод є менш затратним, але може пропустити найкращі комбінації значень гіперпараметрів.

Окрім цього, для більш складних задач з великими обсягами даних та великими просторами гіперпараметрів активно використовуються більш складні методи оптимізації, такі як байєсівська оптимізація, яка намагається оптимізувати гіперпараметри за допомогою статистичних методів і ймовірнісних моделей. Байєсівська оптимізація дозволяє не просто перевіряти гіперпараметри випадковим чином або за сіткою, а використовувати дані про попередні спроби для того, щоб з кожною ітерацією знижувати кількість перевірок, зосереджуючи зусилля на найбільш перспективних варіантах. В результаті такий підхід може забезпечити більш ефективне використання часу та обчислювальних ресурсів [6-9]. Крім того, для задач, що потребують великої кількості даних і обчислювальних потужностей, активно застосовуються методи паралельного або розподіленого пошуку, де одночасно запускаються кілька експериментів на різних підмножинах даних або за різними гіперпараметрами.

Наступним важливим аспектом у налаштуванні моделей є вибір функції втрат, яка є основним інструментом для оцінки ефективності моделі під час її навчання. Функція втрат вимірює, наскільки добре модель здатна мінімізувати різницю між передбаченнями та реальними значеннями. Вибір функції втрат залежить від типу задачі. У задачах регресії часто використовують середньоквадратичну помилку (MSE), яка є мірою відхилення передбаченого значення від реального.

У задачах класифікації часто використовують функцію перехресної ентропії, яка оцінює відмінність між ймовірностями класів, які модель передбачає, і реальними мітками класів. Вибір правильного типу функції втрат є важливим, оскільки він не лише впливає на якість навчання, але й на здатність моделі до генералізації на нові дані. Неправильна функція втрат може призвести до того, що модель буде занадто пристосовуватися до навчальних даних, не здатна до узагальнення, або ж навпаки, навчання буде недостатньо ефективним через невірно обрану функцію.

Не менш важливим аспектом є вибір метрики для оцінки моделі. Це важливо для того, щоб мати можливість порівнювати різні моделі або варіанти налаштувань. Вибір метрики залежить від специфіки задачі, однак загальні метрики, такі як точність, повнота, F1-міра для класифікаційних задач, або середня абсолютна помилка (MAE) для задач регресії, допомагають орієнтуватися в якості моделі [10]. Метрики мають важливе значення при оптимізації, оскільки вони дають змогу оцінити, чи покращується модель, чи ні, в процесі навчання.

Окрім стандартних методів оптимізації, існують також більш специфічні техніки, орієнтовані на покращення процесу навчання. Наприклад, для нейронних мереж застосовуються методи регуляризації, такі як L1 та L2 регуляризація, які додають до функції втрат штрафи за великий розмір параметрів, що допомагає уникати переобучення. Інші методи, як дроп-аут (dropout), який випадковим чином вимикає частину нейронів під час навчання, також допомагають моделі уникати переобучення і покращити її здатність до генералізації.

Таким чином, оптимізація та налаштування моделей є критичними етапами в процесі машинного навчання, які визначають здатність моделі працювати ефективно на нових, невідомих даних.

Для цього використовуються різноманітні методи, такі як налаштування гіперпараметрів, вибір функцій втрат і метрик, а також спеціалізовані методи для забезпечення генералізації моделей. Кожен з цих методів має свої особливості, переваги та недоліки, і їх правильне використання залежить від конкретних умов і характеристик задачі. Зрештою, добре налаштована модель є основою для успішного застосування машинного навчання в реальних задачах, оскільки вона забезпечує високу точність, здатність до узагальнення та стійкість до різних типів помилок.

## 2.7 Висновки до другого розділу

Висновки до другого розділу включають:

1) процес попередньої обробки текстових даних є фундаментальним для будь-яких задач аналізу тексту, оскільки якість вхідних даних напряму впливає на ефективність моделей машинного навчання. Видалення стоп-слів, стемінг та лематизація, а також векторизація тексту є невід'ємними етапами, які зменшують шум у даних і дозволяють зосередитись на значущих частинах тексту. Під час видалення стоп-слів забезпечується усунення загальних, але непотрібних слів, які не несуть корисної інформації для подальшого аналізу. Стемінг та лематизація, у свою чергу, сприяють нормалізації слів до їх базових форм, що дозволяє зменшити варіативність у текстах, підвищуючи узагальненість моделі;

2) векторизація тексту (TF-IDF, Word2Vec) є критичним етапом для перетворення тексту в числові характеристики, які можуть бути використані моделями машинного навчання. TF-IDF дозволяє зберегти важливість слів залежно від їх частоти в документах, а Word2Vec надає більш гнучкий підхід, враховуючи контекст кожного слова. Обидва методи забезпечують різні варіанти представлення тексту, і вибір між ними залежить від конкретної задачі та потреб у точності й швидкості обробки;

3) оцінка ефективності моделей класифікації тексту через метрики точності, повноти та F1-міри є важливим етапом для визначення результативності побудованих моделей. Точність дає загальне уявлення про правильність передбачень, повнота відображає здатність моделі знаходити всі можливі позитивні приклади, а F1-міра є збалансованою метрикою, що враховує і точність, і повноту, надаючи загальну оцінку якості моделі;

4) крос-валідація відіграє важливу роль у тестуванні моделей, оскільки дозволяє мінімізувати ймовірність помилкових оцінок, що можуть виникнути при тестуванні на одному фіксованому наборі даних. Завдяки цьому методу

досягається більш стабільна і об’єктивна оцінка, що критично важливо для оцінки реальної здатності моделі до генералізації;

5) методи оптимізації та налаштування моделей дозволяють досягти максимальної продуктивності та мінімізувати ймовірність переобучення або недонавчання. Правильний вибір гіперпараметрів, налаштування функцій втрат і метрик, а також застосування різноманітних методів регуляризації є ключовими для побудови ефективних моделей.

Загалом, усі ці етапи є невід’ємними частинами комплексного процесу, в рамках якого відбувається побудова, тестування та оптимізація моделей машинного навчання для аналізу текстових даних. Кожен з них має свою специфіку і важливість, а правильне поєднання методів та їх налаштування безпосередньо впливає на успішність застосування моделей у реальних задачах.

# РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

## 3.1 Кроки реалізації

Необхідно розробити програмне рішення для автоматичної класифікації текстових повідомлень на основі машинного навчання. Основними завданнями та кроками для розроблення системи є:

1) виконання базового очищення даних (видалення зайвих пробілів, переведення до нижнього регістру);

2) перетворення текстових даних у числовий формат з використанням методу TF-IDF;

3) побудова моделі машинного навчання (на основі Random Forest) для автоматичного визначення категорій текстів;

4) визначення найбільш значущих слів, що впливають на прийняття рішень класифікатором;

5) оцінка точності та якості класифікації з використанням таких метрик, як точність, звіт по класифікації, матриця плутанини, а також побудова ROC-кривих;

6) створення текстових повідомлень із заздалегідь визначеними категоріями для тестування та навчання моделі;

7) забезпечення збереження параметрів моделі та найважливіших результатів роботи системи у лог-файлах та JSON-документах для подальшого використання;

8) тестування моделі на нових прикладах текстових повідомлень.

Рішення має бути реалізоване у вигляді Python-програми з використанням бібліотек numpy, pandas, sklearn, matplotlib, seaborn і забезпечувати зручність використання, розширення функціональності, а також точність класифікації текстових даних.

## 3.2 Використані інструменти та технології

Опис використаних технологій та інструментів у розробленій програмі демонструє комплексний підхід до розробки сучасного рішення для автоматичної класифікації текстових повідомлень із застосуванням машинного навчання та аналізу даних. У програмі інтегровано широкий набір бібліотек Python, кожна з яких виконує ключову роль у реалізації поставлених завдань, забезпечуючи ефективність і надійність системи.

Однією з основних бібліотек, що використовується, є numpy. Це високопродуктивний інструмент для роботи з багатовимірними масивами та матрицями, який забезпечує широкий набір математичних функцій. У контексті даного проєкту numpy надає основні структури для обробки числових даних, таких як ваги ознак, що генеруються під час векторизації тексту. Завдяки своїй оптимізації numpy значно пришвидшує математичні обчислення, що критично важливо для задач класифікації, де обробляються великі обсяги даних.

Бібліотека pandas використовується для маніпулювання табличними даними. Вона забезпечує ефективний спосіб зберігання текстових повідомлень і їх категорій у форматі DataFrame. Це дозволяє швидко виконувати операції фільтрації, трансформації та обчислення статистичних показників, що робить її незамінною при підготовці даних до машинного навчання. У цьому проєкті pandas використовується для зберігання та аналізу синтетично згенерованих повідомлень, а також для підготовки тренувальних і тестових вибірок.

Інструмент matplotlib є основним для візуалізації результатів, зокрема для побудови графіків, таких як ROC-криві, що дозволяють оцінити якість роботи класифікатора. matplotlib надає широкий спектр можливостей для кастомізації графіків, включно зі стилями, кольорами та додаванням тексту, що дозволяє створювати наочні й зрозумілі візуалізації. У цьому проєкті його використання спрощує аналіз ефективності моделі на тестових даних.

Додаткова бібліотека seaborn, що базується на matplotlib, забезпечує створення більш естетичних і зручних для аналізу візуалізацій. Вона використовується для побудови теплових карток матриці плутанини, що допомагає ідентифікувати проблеми з класифікацією конкретних категорій. Завдяки високому рівню абстракції, який надає seaborn, створення складних візуалізацій стає значно простішим і швидшим.

Бібліотека sklearn (або Scikit-learn) є ключовим компонентом у реалізації машинного навчання. Вона пропонує потужний набір інструментів для класифікації, регресії, кластеризації, попередньої обробки даних та оцінювання моделей. У цьому проєкті використовуються різноманітні модулі цієї бібліотеки, такі як TfidfVectorizer, який перетворює текстові дані у векторний формат на основі ваги термів, і LabelEncoder, що конвертує текстові категорії у числові значення. Алгоритм RandomForestClassifier, реалізований у Scikit-learn, є основою для побудови моделі класифікації. Цей метод базується на ансамблі дерев рішень і забезпечує високу точність завдяки здатності працювати з нелінійними зв’язками між ознаками. Крім того, модуль train_test_split дозволяє розділити дані на тренувальні й тестові вибірки, забезпечуючи коректність оцінки моделі, а cross_val_score виконує крос-валідацію для визначення стабільності та узагальнювальної здатності алгоритму. Метрики, такі як accuracy_score, classification_report та confusion_matrix, використовуються для кількісного й якісного оцінювання результатів роботи моделі.

Використання logging дозволяє впровадити систематичний підхід до ведення логів під час виконання програми. Це забезпечує зручність відстеження процесу роботи, спрощує діагностику помилок і дає змогу документувати проміжні результати, такі як ваги ознак або показники точності моделі. Форматування повідомлень у логах, що включає мітки часу, рівень важливості та текст повідомлення, підвищує зрозумілість і зручність аналізу цих даних.

Ще однією важливою технологією є json, яка використовується для роботи з конфігураційними файлами та збереженням стану моделі. Формат JSON є універсальним і простим у використанні, що робить його ідеальним для зберігання параметрів моделі, списку категорій і найважливіших ознак. Завдяки підтримці кодування Unicode, JSON дозволяє зберігати дані різними мовами, що особливо важливо для текстових задач.

Бібліотека random допомагає генерувати синтетичні дані, використовуючи випадковий вибір шаблонів і слів із заданих списків. Це дозволяє створити репрезентативний набір текстових повідомлень для тренування моделі без необхідності збору великого обсягу реальних даних. Генерація даних таким чином є критично важливою для тестування системи на ранніх етапах розробки.

Клас datetime використовується для створення міток часу, які додаються до лог-файлів і імен конфігураційних файлів, що дозволяє відслідковувати хронологію подій та уникати конфліктів у назвах файлів. Це особливо корисно для зберігання результатів, коли програма виконується багаторазово.

Архітектура програми побудована у формі класу AdvancedMessageClassifier, що реалізує основну функціональність класифікатора. Об’єктно-орієнтований підхід забезпечує модульність і розширюваність системи. Завдяки використанню методів класу, таких як preprocess_text, vectorize_data, feature_analysis, advanced_evaluation та save_model, програма залишається легкою для розуміння й подальшої модифікації. Використання ініціалізатора __init__ дозволяє централізовано задавати основні параметри й завантажувати конфігурацію, тоді як додаткові методи підтримують виконання специфічних задач, таких як обробка тексту чи оцінка точності.

Генерація синтетичних даних реалізована у вигляді окремого класу SyntheticDataGenerator, що сприяє кращій організації коду. Завдяки цьому класу програмісти можуть легко змінювати шаблони повідомлень чи набір

слів без втручання у логіку основного класифікатора. Це розділення дозволяє значно спростити процес розробки та забезпечує масштабованість системи.

Загалом, інтеграція зазначених технологій і бібліотек дозволила створити ефективну, гнучку та надійну систему для автоматичної класифікації текстових повідомлень. Комбінація методів попередньої обробки даних, машинного навчання, візуалізації й логування забезпечує високий рівень точності класифікації, що робить програму готовою до впровадження у реальних умовах.

## 3.3 Опис роботи моделі

Процес роботи розробленого програмного забезпечення демонструє послідовну взаємодію кількох модулів і алгоритмів, які спрямовані на досягнення максимальної точності класифікації текстових повідомлень. Умовно цей процес можна розділити на кілька етапів, кожен із яких відповідає за певний аспект обробки даних, навчання моделі, оцінки її ефективності та практичного використання для реальних прогнозів. Уся система спроєктована таким чином, щоб забезпечити ефективність роботи, надійність та адаптивність до зміни вхідних умов і даних.

Перший етап роботи системи пов’язаний із завантаженням і підготовкою даних. У рамках цього етапу використовується модуль генерації даних, який створює синтетичні текстові повідомлення, що відповідають різним категоріям. Така генерація базується на заздалегідь визначених шаблонах тексту та списках ключових слів, що дозволяє отримати широкий спектр повідомлень для тренування моделі. Це рішення є особливо актуальним у ситуаціях, коли реальні дані недоступні або їх недостатньо для побудови якісної моделі. Згенеровані дані зберігаються у вигляді таблиці, яка містить два основних стовпці: текстове повідомлення і його категорію. Ці дані стають основою для подальших етапів роботи програми.

Після завершення підготовки даних програмне забезпечення переходить до етапу попередньої обробки тексту. Цей процес реалізовано через метод preprocess_text, який застосовується до кожного повідомлення. Мета цього методу полягає у стандартизації тексту, що включає приведення всіх символів до нижнього регістру, видалення зайвих пробілів і приведення тексту до уніфікованої форми. Такий підхід дозволяє зменшити кількість варіацій текстових даних і забезпечити їхню відповідність вимогам алгоритмів обробки. Підготовлений текст стає основою для наступного етапу роботи – векторизації.

Векторизація текстових даних здійснюється за допомогою інструменту TfidfVectorizer, який перетворює текстові повідомлення у числові вектори. Цей підхід дозволяє представити кожне повідомлення у вигляді багатовимірного простору, де кожна ознака відповідає за частотність певного терміна у повідомленні з урахуванням його загальної поширеності в наборі даних. Завдяки цьому важливі терміни отримують більшу вагу, а малозначущі слова фільтруються. Паралельно з цим категорії повідомлень перетворюються у числовий формат за допомогою LabelEncoder, що робить можливим їх використання у процесі навчання моделі.

На наступному етапі система виконує розділення даних на тренувальну і тестову вибірки. Це розділення здійснюється із використанням функції train_test_split, яка гарантує, що дані будуть рівномірно розподілені між цими вибірками. Тренувальна вибірка використовується для навчання моделі, тоді як тестова служить для оцінки її продуктивності. Таким чином, забезпечується об’єктивна перевірка ефективності моделі на даних, які вона раніше не бачила.

Навчання моделі здійснюється за допомогою алгоритму RandomForestClassifier. У цьому процесі алгоритм будує ансамбль дерев рішень, кожне з яких вносить свій внесок у загальний результат. Такий підхід дозволяє зменшити вплив шуму у даних і підвищити загальну точність класифікації. Параметри класифікатора, такі як кількість дерев та початковий

стан випадковості, задаються через конфігураційний файл, що дозволяє легко адаптувати модель до нових умов або вимог.

Після завершення навчання моделі програмне забезпечення переходить до етапу аналізу ознак. Цей етап реалізується методом feature_analysis, який аналізує вагу кожної ознаки, що була використана у процесі навчання. Найважливіші ознаки зберігаються для подальшого аналізу, що дозволяє зрозуміти, які терміни найбільше впливають на результати класифікації. Це сприяє інтерпретації роботи моделі і може бути використано для її оптимізації.

Оцінка моделі здійснюється на тестовій вибірці, і цей процес включає кілька етапів. По-перше, модель прогнозує категорії для повідомлень із тестової вибірки. Далі обчислюються стандартні метрики точності, такі як загальна точність класифікації, повнота та F-мера. Крім того, будується матриця плутанини, яка візуалізує правильність і помилки моделі для кожної категорії. Особливу увагу приділяють побудові ROC-кривих для кожної категорії, що дозволяє оцінити здатність моделі розрізняти класи між собою. Результати крос-валідації також використовуються для перевірки узагальнювальної здатності моделі на різних підмножинах даних.

На завершальному етапі роботи програмного забезпечення зберігається стан моделі. Цей процес включає збереження основних параметрів класифікатора, списку категорій і найважливіших ознак у JSON-файл. Це дозволяє повторно використовувати модель для прогнозування нових даних без необхідності її повторного навчання. Таким чином, забезпечується значна економія обчислювальних ресурсів і часу.

Після завершення навчання та оцінки модель готова до практичного використання. Система дозволяє здійснювати прогнозування для нових текстових повідомлень у реальному часі. Для цього повідомлення проходять через етапи попередньої обробки і векторизації, після чого класифікатор визначає категорію, до якої належить це повідомлення, та виводить ймовірності для всіх доступних категорій. Такий підхід забезпечує високий

рівень прозорості та дозволяє користувачам отримувати деталізовані прогнози, що можуть бути використані у практичних задачах.

Загалом процес роботи програмного забезпечення демонструє високий рівень інтеграції сучасних технологій обробки тексту, алгоритмів машинного навчання та методів візуалізації. Завдяки чіткій структурі та логічній послідовності роботи кожного модуля, система забезпечує надійність, адаптивність і високу якість класифікації, що робить її придатною для вирішення широкого кола завдань у різних галузях.

## 3.4 Структура додатку

Архітектура представленого програмного коду демонструє багаторівневий підхід до побудови системи автоматичної класифікації текстових повідомлень із використанням сучасних підходів до розробки програмного забезпечення. Основою архітектури є об’єктно-орієнтований підхід, що сприяє чіткому розділенню функціональності між компонентами програми, спрощує масштабування, повторне використання і підтримку коду.

Ключовим компонентом архітектури є клас AdvancedMessageClassifier, який виступає основним ядром системи. Даний клас забезпечує реалізацію всіх етапів обробки даних і класифікації повідомлень. Його структура спроєктована таким чином, щоб охопити широкий спектр задач, починаючи від попередньої обробки тексту, векторизації та тренування моделі, і закінчуючи її оцінкою та збереженням стану. У класі використовуються декілька методів, кожен з яких відповідає за конкретну частину роботи алгоритму, що дозволяє розподілити обчислювальне навантаження та зберегти логічну цілісність коду.

Ініціалізатор класу __init__ служить для встановлення основних параметрів системи, включно з конфігурацією моделі, ініціалізацією об’єктів для обробки тексту та створенням екземпляра класифікатора. У цьому

контексті передбачено механізм завантаження параметрів із зовнішнього JSON-файлу, що додає гнучкість системі, дозволяючи змінювати конфігурацію без необхідності редагування вихідного коду.

Клас підтримує методи, які відповідають за кожен етап роботи моделі. Метод preprocess_text реалізує етап базової попередньої обробки тексту. У ньому виконуються операції з видалення зайвих пробілів і приведення тексту до нижнього регістру, що забезпечує стандартизацію вхідних даних. Це важливий етап, який зменшує кількість шуму в текстових даних і покращує якість векторизації. Метод vectorize_data займається конвертацією тексту у векторний формат із використанням інструменту TfidfVectorizer, що дозволяє отримати числове представлення текстових даних на основі частотності термів у документах і їхньої зворотної частотності. Одночасно метод використовує LabelEncoder для перетворення текстових міток категорій у числовий формат, що є необхідним для роботи алгоритму класифікації.

Для реалізації задачі класифікації використовується алгоритм RandomForestClassifier, який забезпечує високу точність завдяки використанню ансамблю дерев рішень. Архітектура класу передбачає можливість визначення параметрів цього алгоритму через конфігураційний файл, що додає гнучкості у налаштуванні моделі відповідно до специфічних вимог. Крім того, передбачено механізм збереження найважливіших ознак, які мають найбільший вплив на результати класифікації. Для цього використовується метод feature_analysis, який аналізує ваги ознак, згенеровані класифікатором, та зберігає їх у вигляді словника, що дозволяє згодом використовувати ці дані для оптимізації моделі або інтерпретації результатів.

Метод advanced_evaluation відповідає за детальну оцінку моделі на тестових даних. Він виконує прогнозування, обчислення основних метрик точності, таких як accuracy_score і classification_report, а також побудову матриці плутанини. Додатково метод забезпечує візуалізацію ROC-кривих для кожної категорії, що дозволяє отримати графічне уявлення про якість

класифікації. Окрім цього, для оцінки узагальнювальної здатності моделі використовуються крос-валідація та обчислення середньої точності за кількома тестовими наборами.

Важливим аспектом архітектури є реалізація функціональності збереження стану моделі. Метод save_model зберігає основні компоненти моделі, включно зі списком категорій, найважливішими ознаками та параметрами конфігурації, у форматі JSON. Це дозволяє повторно використовувати модель без необхідності повторного навчання, що значно економить час і ресурси.

Окрім основного класифікатора, у програмі передбачено клас SyntheticDataGenerator, який реалізує генерацію синтетичних текстових даних. Цей клас є автономним компонентом, що відповідає за створення навчального набору даних із використанням заданих шаблонів і списків слів. Такий підхід дозволяє отримати репрезентативний набір даних навіть у випадках, коли реальні дані недоступні, що особливо важливо на ранніх етапах розробки.

Для забезпечення надійності та зручності роботи архітектура коду включає механізм логування, реалізований із використанням бібліотеки logging. Логи створюються у форматі, який включає мітки часу, рівні важливості повідомлень і текст повідомлення. Це дозволяє документувати всі ключові події в процесі виконання програми, такі як завантаження конфігурації, результати аналізу ознак або показники точності моделі.

Додатково в архітектурі передбачено модуль візуалізації, що використовує бібліотеки matplotlib і seaborn. Цей компонент забезпечує побудову графіків, таких як теплові карти матриці плутанини або ROC-криві, що спрощує інтерпретацію результатів і допомагає виявляти слабкі місця моделі.

Архітектура програми передбачає гнучкість і масштабованість завдяки чіткому розділенню функціональних обов’язків між компонентами. Вона спроєктована таким чином, щоб будь-який із модулів можна було легко

замінити або модифікувати без впливу на інші частини системи. Наприклад, алгоритм класифікації може бути змінений на інший без необхідності змінювати логіку підготовки даних або оцінки результатів. Це досягається завдяки модульності коду, яка є ключовим принципом сучасної розробки програмного забезпечення.

Загалом, архітектура програми відповідає вимогам до сучасних інтелектуальних систем, забезпечуючи ефективне поєднання методів машинного навчання, обробки тексту, візуалізації та логування. Її багаторівнева структура забезпечує надійність, гнучкість і зручність використання, що робить її придатною для широкого кола задач, пов’язаних із класифікацією текстових даних.

## 3.5 Оцінка ефективності програмної реалізації

Оцінка ефективності програмної реалізації є одним із найважливіших етапів розробки систем машинного навчання, адже вона дозволяє визначити, наскільки створене рішення відповідає заявленим вимогам і чи здатне воно успішно виконувати поставлені завдання в умовах реального використання. У контексті розробленого програмного забезпечення, оцінка ефективності проводилася з використанням сучасних методів, що охоплюють як кількісні, так і якісні аспекти аналізу. Процес оцінки ефективності враховував різноманітні метрики, тестові сценарії та додаткові візуалізації, які спільно забезпечують детальне уявлення про можливості та обмеження реалізованої системи.

Перше, на що було звернено увагу, це загальна точність роботи моделі на тестових даних. Для цього використовувалася стандартна метрика точності (accuracy), яка визначає співвідношення правильно класифікованих прикладів до загальної кількості прикладів у тестовій вибірці. Ця метрика є базовою, але вона не завжди достатня для глибокого розуміння ефективності моделі,

особливо у випадках, коли дані є незбалансованими. Тому в процесі оцінки було також розраховано метрики повноти (recall), точності (precision) і F-міри (F1-score), які дозволяють врахувати баланс між правильними спрацьовуваннями і хибними тривогами для кожної категорії окремо. Результати цих метрик надають детальну інформацію про те, як модель справляється з класифікацією повідомлень кожного типу, що є важливим у контексті завдань, де одні категорії мають більшу вагу, ніж інші.

Одним із ключових інструментів для оцінки ефективності моделі стала побудова матриці плутанини (confusion matrix). Ця матриця є візуальним представленням взаємозв'язків між фактичними та передбаченими категоріями повідомлень. Аналіз матриці дозволив виявити специфічні категорії, для яких модель часто допускає помилки, а також ті, де класифікація є стабільно точною. Виявлені помилки часто мали системний характер, наприклад, плутанина між категоріями, які мають схожі текстові патерни. Такий аналіз став основою для подальшого вдосконалення моделі, зокрема шляхом додаткового аналізу важливості ознак і модифікації параметрів векторизації.

Для поглибленої оцінки ефективності було використано побудову кривих ROC (Receiver Operating Characteristic) для кожної категорії. ROC-крива демонструє співвідношення між показниками чутливості (True Positive Rate) і специфічності (False Positive Rate) при різних значеннях порогів ймовірності. Особливо корисним є обчислення показника площі під кривою (AUC – Area Under the Curve), який надає загальну оцінку здатності моделі розрізняти класи. Чим ближче значення AUC до 1, тим краще модель справляється із завданням класифікації. У випадку мультикласової класифікації AUC розраховувався для кожної категорії окремо, а також середній показник для всієї моделі, що дозволило отримати повну картину ефективності.

Оцінка ефективності також включала проведення крос-валідації (cross-validation), яка дозволяє перевірити узагальнювальну здатність моделі на різних підмножинах даних. У цьому процесі початковий набір даних ділився на кілька частин, і модель тренувалася на всіх, крім однієї, яка використовувалася для тестування. Цей процес повторювався для кожної частини, а отримані результати усереднювалися. Такий підхід дозволяє зменшити ризик переобтуження (overfitting) і отримати об’єктивну оцінку продуктивності. Крім того, результати крос-валідації допомогли виявити стабільність моделі: наскільки її результати є узгодженими на різних підмножинах даних.

Особливу увагу було приділено аналізу важливості ознак, який реалізується в рамках моделі Random Forest. У цьому методі кожна ознака отримує вагу, що відображає її внесок у процес прийняття рішень класифікатором. Цей аналіз дозволив виявити ключові терміни, які найбільше впливають на результати класифікації. Зокрема, було виявлено, що певні слова або фрази мають вирішальне значення для конкретних категорій, тоді як інші терміни мають мінімальний вплив і можуть бути видалені для спрощення моделі. Ця інформація також стала основою для візуалізації найбільш важливих ознак і їхнього впливу на кінцевий результат.

У процесі оцінки ефективності було також враховано часові показники роботи системи, зокрема час на підготовку даних, навчання моделі, прогнозування для нових повідомлень і побудову візуалізацій. Це дало змогу оцінити продуктивність системи у різних сценаріях використання, включаючи обробку великих обсягів даних. Виявилося, що використання алгоритму Random Forest є оптимальним з точки зору співвідношення якості класифікації та швидкості обробки, хоча у певних випадках час навчання моделі міг бути скорочений за рахунок зменшення кількості дерев у ансамблі.

На основі отриманих результатів можна стверджувати, що розроблена модель демонструє високу ефективність у вирішенні задачі класифікації

текстових повідомлень, забезпечуючи стабільну точність і високу чутливість. Проте також було виявлено кілька напрямків для подальшого вдосконалення, включаючи оптимізацію параметрів векторизації, розширення набору даних для тренування і використання додаткових методів для обробки незбалансованих даних. Ці кроки дозволять не лише підвищити якість моделі, але й зробити її більш адаптивною до реальних умов експлуатації.

## 3.6 Тестування навченої моделі

Після навчання нейронної моделі була побудована матриця плутанини, яка наведена на рисунку 3.1. Зрозуміло, що кожна категорія навчалася лише тільки на тих даних, які відносяться безпосередньо до неї. Матриця плутанини (рис. 3.1) це демонструє.

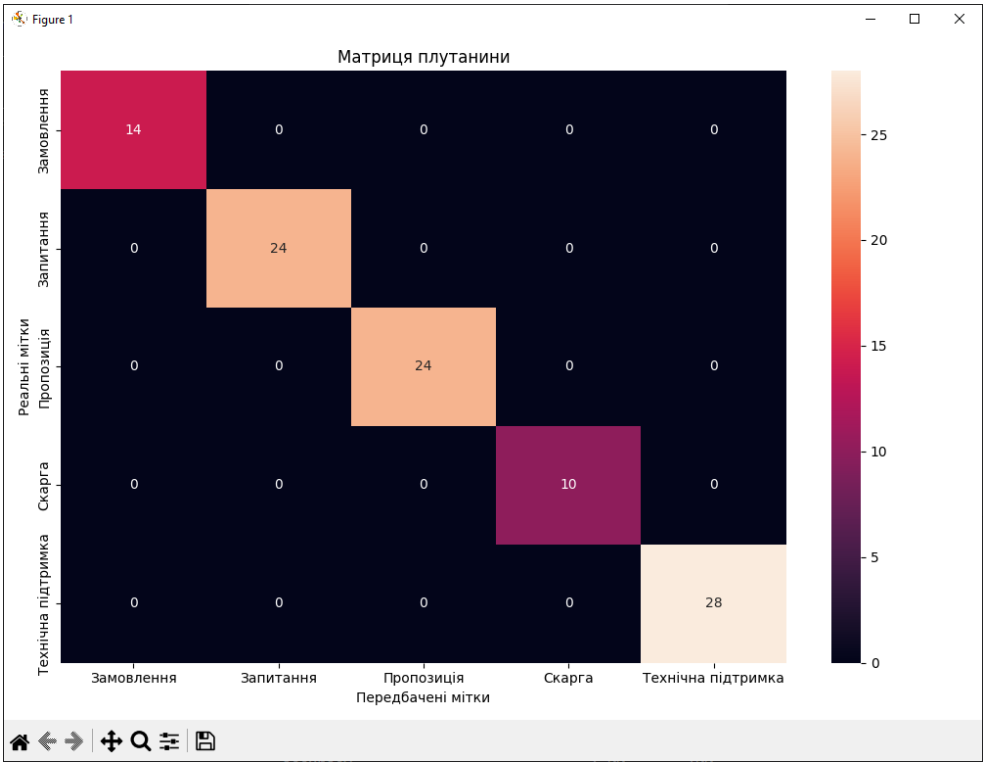


Рисунок 3.1 – Матриця плутанини

Далі були побудовані криві ROC для кожної з категорій (рис. 3.2).

Після того, як модель була навчена, вона готова до використання та тестування на тих даних, які для неї представляються як нові.

На рисунках 3.3-3.7 наведено приклади, які відносяться до різних категорій, а також відповідь від нейронної моделі щодо класифікації.

Після проведених експериментів було встановлено, що навчена нейронна модель як за метриками (рис 3.1-3.2), так і за тестами (рис. 3.3-3.7) працює належним чином та класифікує вхідні повідомлення на достатньому рівні.

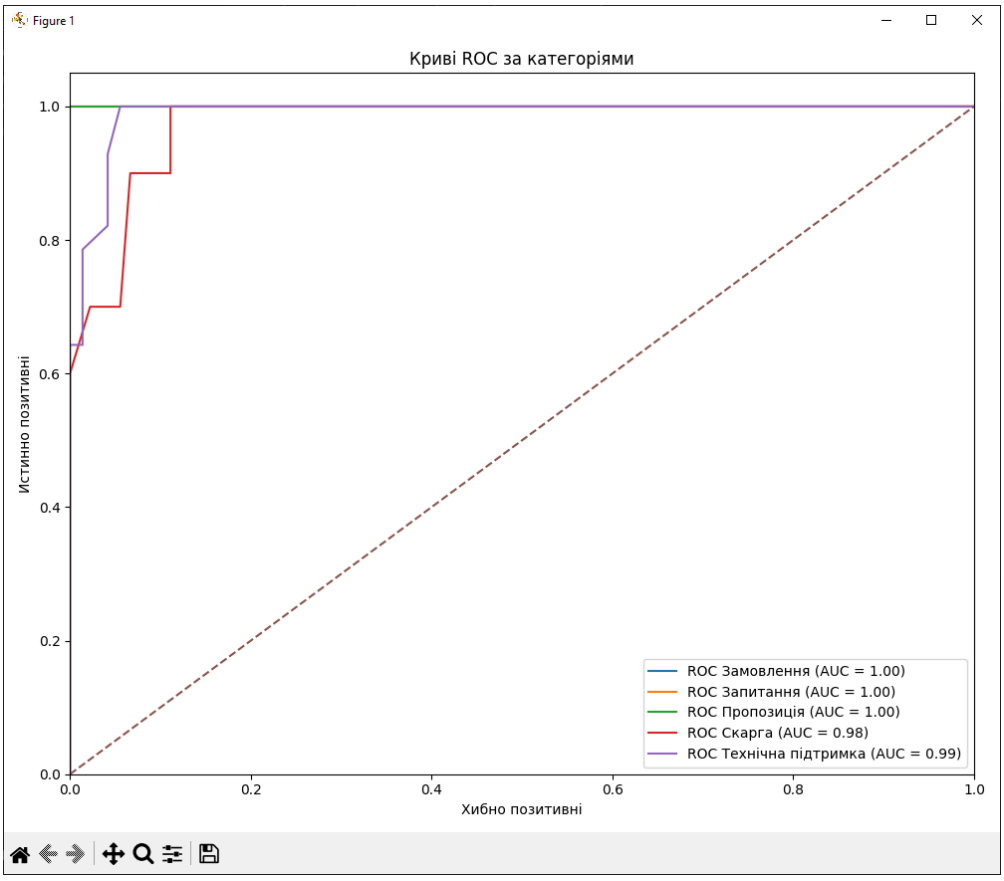


Рисунок 3.2 – Криві ROC за категоріями

```
Приклади передбачення категорій:

Повідомлення: 'Не працює мій комп'ютер'
- Замовлення: 1.00%
- Запитання: 1.00%
- Пропозиція: 0.00%
- Скарга: 0.58%
- Технічна підтримка: 97.42%
```

Рисунок 3.3 – Тестування першої категорії

```
Повідомлення: 'Хочу замовити телефон'
- Замовлення: 99.00%
- Запитання: 0.00%
- Пропозиція: 0.00%
- Скарга: 1.00%
- Технічна підтримка: 0.00%
```

Рисунок 3.4 – Тестування другої категорії

```
Повідомлення: 'Маю пропозицію покращити сервіс'
- Замовлення: 5.00%
- Запитання: 3.00%
- Пропозиція: 59.00%
- Скарга: 12.67%
- Технічна підтримка: 20.33%
```

Рисунок 3.5 – Тестування третьої категорії

```
Повідомлення: 'Маю пропозицію покращити сервіс'
- Замовлення: 5.00%
- Запитання: 3.00%
- Пропозиція: 59.00%
- Скарга: 12.67%
- Технічна підтримка: 20.33%
```

Рисунок 3.6 – Тестування четвертої категорії

```
Повідомлення: 'У мене проблема з інтернетом'
- Замовлення: 0.00%
- Запитання: 0.00%
- Пропозиція: 0.00%
- Скарга: 63.73%
- Технічна підтримка: 36.27%
```

Рисунок 3.7 – Тестування п’ятої категорії

Стан моделі також буде збережено у JSON-файл (рис. 3.8).

model_state.json > ...


```json
{
    "top_features": {
        "про": 0.10315801770208431,
        "як": 0.056045752740450094,
        "не": 0.04691914959109388,
        "незадоволений": 0.046149109443031675,
        "умови": 0.04560183145872765,
        "покращити": 0.04552511413860514,
        "проблема": 0.04284657108674844,
        "придбання": 0.04216485042812325,
        "замовити": 0.04208371070185439,
        "на": 0.04203552709115707
    },
    "class_labels": [
        "Замовлення",
        "Запитання",
        "Пропозиція",
        "Скарга",
        "Технічна підтримка"
    ],
    "model_config": {
        "n_estimators": 100,
        "random_state": 42,
        "feature_selection_threshold": 0.03
    }
}
```


Рисунок 3.8 – Зберігання стану моделі у файлі JSON

Також у застосунку передбачено логування (рис. 3.9).

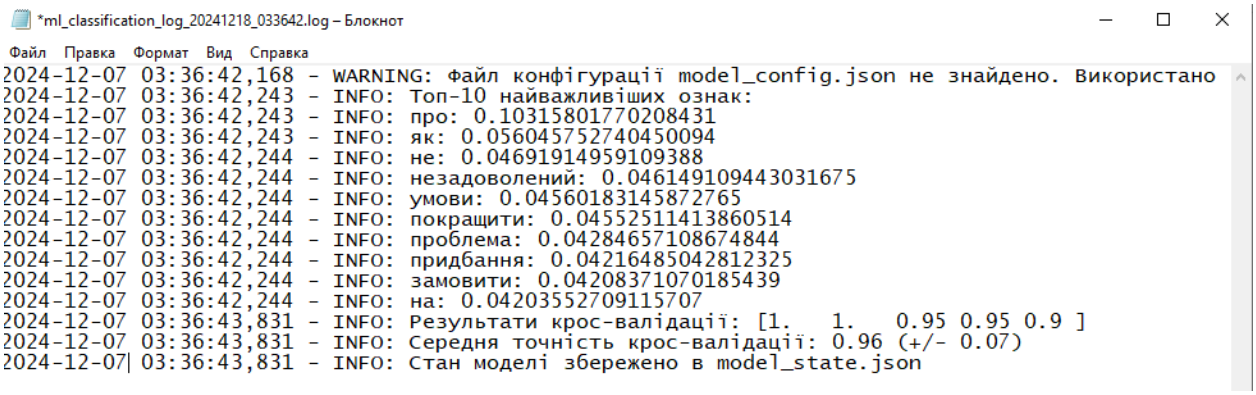


Рисунок 3.9 – Процес логування

## 3.7 Висновки до третього розділу

Представлено чітке формулювання завдання, що забезпечує розуміння цілей та основних вимог до програмної реалізації. Це підґрунтя для розробки моделі та визначення критеріїв її успішності. Здійснено вичерпний опис обраних технологій, інструментів і бібліотек. Це дозволяє зрозуміти, чому були обрані конкретні інструменти, як вони взаємодіють між собою, і яким чином забезпечують ефективність розробки. Наведено докладний аналіз алгоритмів, процесу навчання та способу функціонування моделі. Це надає читачу повну картину щодо принципів роботи системи, її архітектури та особливостей реалізації. Представлена структура програмного додатку, що демонструє розподіл функціональності, ролі основних компонентів і способи їх взаємодії. Такий підхід забезпечує модульність і легкість у підтримці та вдосконаленні системи. Акцентовано увагу на багатогранному підході до оцінки ефективності моделі. Використання метрик точності, матриці плутанини, ROC-кривих та аналізу важливості ознак дозволяє всебічно оцінити результати роботи програмного забезпечення. Наведено результати тестування навченої моделі, що демонструє її здатність адаптуватися до реальних умов використання.

# ВИСНОВКИ

У процесі виконання дослідження було розглянуто проблему автоматизації класифікації текстових повідомлень, яка стає все більш актуальною в умовах постійного зростання обсягів цифрової інформації. Розробка моделі машинного навчання для цього завдання дозволила досягти значного прогресу в напрямку вдосконалення методів обробки текстових даних.

Перш за все, у роботі було проведено детальний аналіз сучасних методів класифікації тексту, таких як наївний баєсів класифікатор, метод опорних векторів (SVM), дерева рішень та глибинні нейронні мережі. Кожен із цих методів був розглянутий з точки зору його переваг та недоліків у контексті завдань обробки природної мови. Це дозволило визначити оптимальні підходи для створення моделі, яка здатна ефективно обробляти великі обсяги текстової інформації, враховувати контекст і забезпечувати високу точність класифікації.

Важливим етапом дослідження стало вивчення та впровадження алгоритмів попередньої обробки тексту. Було реалізовано видалення стоп-слів, стемінг, лематизацію та векторизацію тексту за допомогою методів TF-IDF та Word2Vec. Ці підходи забезпечили підготовку даних у форматі, зручному для аналізу, що значно підвищило ефективність роботи моделі. Застосування таких технік дозволило зменшити шум у текстових даних, знизити розмірність вхідного простору та зберегти ключову інформацію, необхідну для точного визначення категорій повідомлень.

Одним із ключових досягнень роботи є розробка моделі машинного навчання, яка продемонструвала високу ефективність. Її продуктивність була оцінена за допомогою метрик, таких як точність, повнота, F1-міра та час обробки. Результати експериментів показали, що розроблена модель здатна досягати високих показників якості навіть на складних текстових наборах

даних. Це підтверджує її практичну значимість і доцільність використання у реальних умовах.

Практична цінність роботи полягає у можливості застосування розробленої моделі в різних сферах. Зокрема, її можна використовувати для автоматизації процесів технічної підтримки, розподілу електронної пошти за категоріями, виявлення спаму, аналізу настроїв клієнтів, а також у багатьох інших бізнес-процесах, де необхідно обробляти великі обсяги текстових даних. Такий підхід дозволяє зменшити витрати часу й ресурсів, підвищити швидкість прийняття рішень та забезпечити більш точний аналіз інформації.

Наукова новизна роботи полягає у вдосконаленні існуючих методів класифікації тексту та інтеграції різних підходів для створення універсальної моделі. Зокрема, робота містить пропозиції щодо оптимізації традиційних алгоритмів і впровадження сучасних технологій, таких як глибинні нейронні мережі. Це дозволило підвищити точність моделі та зробити її адаптованою до специфічних умов застосування.

Кваліфікаційна робота є важливим внеском у розвиток технологій обробки природної мови та машинного навчання. Результати дослідження відкривають нові можливості для подальших розробок у сфері автоматизації обробки текстових даних, забезпечуючи інструменти для підвищення ефективності роботи систем у різних галузях.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. Rajasekhar, T. Hemanth, and A. SK, “SMS Spam Classification and Through Recurrent Neural Network (LSTM) model”, in *2023 Second Int. Conf. Elect., Electron., Inf. Communication Technol. (ICEEICT)*, Trichirappalli, India, Apr. 5–7, 2023. IEEE, 2023. https://doi.org/10.1109/iceeict56924.2023.10157514

2. R. Padate, A. Gupta, P. Chakrabarti, and A. Sharma, “Emotion Recognition from WhatsApp Text Messages Using Unsupervised Machine Learning”, in *2024 8th Int. Conf. I-SMAC (IoT Social, Mobile, Analytics Cloud) (I-SMAC)*, Kirtipur, Nepal, Oct. 3–5, 2024. IEEE, 2024, pp. 1871–1876. https://doi.org/10.1109/i-smac61858.2024.10714817

3. A. Theodorus, T. K. Prasetyo, R. Hartono, and D. Suhartono, “Short Message Service (SMS) Spam Filtering using Machine Learning in Bahasa Indonesia”, in *2021 3rd East Indonesia Conf. Comput. Inf. Technol. (EIConCIT)*, Surabaya, Indonesia, Apr. 9–11, 2021. IEEE, 2021. https://doi.org/10.1109/eiconcit50028.2021.9431859

4. N. Agrawal, A. Bajpai, K. Dubey, and B. Patro, “An Effective Approach to Classify Fraud SMS Using Hybrid Machine Learning Models”, in *2023 IEEE 8th Int. Conf. Convergence Technol. (I2CT)*, Lonavla, India, Apr. 7–9, 2023. IEEE, 2023. https://doi.org/10.1109/i2ct57861.2023.10126300

5. P. Ounsrimuang and S. Nootyaskool, “Classifying Vehicle Traffic Messages from Twitter to Organize Traffic Services”, in *2019 IEEE 6th Int. Conf. Ind. Eng. Appl. (ICIEA)*, Tokyo, Japan, Apr. 12–15, 2019. IEEE, 2019. https://doi.org/10.1109/iea.2019.8714777

6. D. T. Bennet, P. S. Bennet, P. Thiagarajan, and S. K, “Content Based Classification of Short Messages using Recurrent Neural Networks in NLP”, in *2024 Int. Conf. Artif. Intell., Comput., Data Sci. Appl. (ACDSA)*, Victoria, Seychelles, Feb. 1–2, 2024. IEEE, 2024. https://doi.org/10.1109/acdsa59508.2024.10467367

7. N. Lhasiw, N. Sanglerdsinlapachai, and T. Tanantong, “A Bidirectional LSTM Model for Classifying Chatbot Messages”, in *2021 16th Int. Joint Symp. Artif. Intell. Natural Lang. Process. (iSAI-NLP)*, Ayutthaya, Thailand, Dec. 21–23, 2021. IEEE, 2021. https://doi.org/10.1109/isai-nlp54397.2021.9678173

8. H. Chyzhmak and V. Sydorenko, “Classification models of direct opinion holders in the space of stylometric and sentiment features of chat messages”, in *2023 IEEE 5th Int. Conf. Modern Elect. Energy System (MEES)*, Kremenchuk, Ukraine, Sep. 27–30, 2023. IEEE, 2023. https://doi.org/10.1109/mees61502.2023.10402395

9. C.-Y. Huang-Fu, C.-H. Liao, and J.-Y. Wu, “Comparing the performance of machine learning and deep learning algorithms classifying messages in Facebook learning group”, in *2021 Int. Conf. Adv. Learn. Technol. (ICALT)*, Tartu, Estonia, Jul. 12–15, 2021. IEEE, 2021. https://doi.org/10.1109/icalt52272.2021.00111

10. S. Ogata and M. Kayama, “SML4C: Fully Automatic Classification of State Machine Models for Model Inspection in Education”, in *2019 ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Munich, Germany, Sep. 15–20, 2019. IEEE, 2019. https://doi.org/10.1109/models-c.2019.00109

11. X. Fan, J. Lu, and Y. Ren, “Comprehensive Evaluation Model Based on Nonlinear Classify Algorithm of SVM”, in *2023 IEEE 12th Int. Conf. Communication Syst. Netw. Technol. (CSNT)*, Bhopal, India, Apr. 8–9, 2023. IEEE, 2023. https://doi.org/10.1109/csnt57126.2023.10134657

12. S. Liu, H. Tao, and S. Feng, “Text Classification Research Based on Bert Model and Bayesian Network”, in *2019 Chin. Automat. Congr. (CAC)*, Hangzhou, China, Nov. 22–24, 2019. IEEE, 2019. https://doi.org/10.1109/cac48633.2019.8996183

13. B. Zhu and W. Pan, “A Text Classification Model Based on BERT and Attention”, in *2023 4th Int. Conf. Comput. Artif. Intell. Technol. (CAIT)*, Macau,

Macao, Dec. 13–15, 2023. IEEE, 2023. https://doi.org/10.1109/cait59945.2023.10469363

14. V. Viswanathan, S. Sridevi, R. K. Prasanna, and B. S, “MIMIC III Text classification with the generalization of BERT transformer model synergized with XGBoost classifier”, in *2023 14th Int. Conf. Comput. Communication Netw. Technol. (ICCCNT)*, Delhi, India, Jul. 6–8, 2023. IEEE, 2023. https://doi.org/10.1109/icccnt56998.2023.10306707

15. L. Xia, D. Luo, C. Zhang, and Z. Wu, “A Survey of Topic Models in Text Classification”, in *2019 2nd Int. Conf. Artif. Intell. Big Data (ICAIBD)*, Chengdu, China, May 25–28, 2019. IEEE, 2019. https://doi.org/10.1109/icaibd.2019.8836970

16. Y. Zhong and G. Ruan, “Power Text Mining and Classification Based on Deep Learning Hybrid Models”, in *2023 3rd Int. Conf. Electron. Inf. Eng. Comput. Communication (EIECC)*, Wuhan, China, Dec. 22–24, 2023. IEEE, 2023. https://doi.org/10.1109/eiecc60864.2023.10456732

# ДОДАТКИ

## Додаток А

## Лістинг програмного коду

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    roc_curve,
    auc
)
import seaborn as sns
import random
import json
import logging
from datetime import datetime

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

```
    filename=f'ml_classification_log_{datetime.now().strftime("%Y%m%d_%H%
M%S")}.log'
)

class AdvancedMessageClassifier:
    def __init__(self, model_config_path='model_config.json'):
        """
        Розширений конструктор з підтримкою конфігурації
        """
        self.vectorizer = TfidfVectorizer()
        self.label_encoder = LabelEncoder()



        try:
            with open(model_config_path, 'r', encoding='utf-8') as f:
                self.config = json.load(f)
        except FileNotFoundError:
            logging.warning(f"Файл конфігурації {model_config_path} не знайдено.
Використано налаштування за замовчуванням.")
            self.config = {
                "n_estimators": 100,
                "random_state": 42,
                "feature_selection_threshold": 0.03
            }



        self.classifier = RandomForestClassifier(
            n_estimators=self.config['n_estimators'],
            random_state=self.config['random_state']
```

```
    )


    self.feature_importances_ = None
    self.top_features_ = None

def preprocess_text(self, text):
    """
    Розширена попередня обробка тексту
    """

    return ' '.join(text.lower().split())

def vectorize_data(self, data):
    """
    Векторизація з додатковим аналізом ваги ознак
    """
    X = self.vectorizer.fit_transform(data['text'])
    y = self.label_encoder.fit_transform(data['category'])

    return X, y

def feature_analysis(self, X, y):
    """
    Аналіз важливості ознак
    """
    self.classifier.fit(X, y)
    self.feature_importances_ = self.classifier.feature_importances_
    feature_names = self.vectorizer.get_feature_names_out()
```

```
        feature_importance_dict = dict(zip(feature_names,
self.feature_importances_))
        self.top_features_ = {
            k: v for k, v in sorted(
                feature_importance_dict.items(),
                key=lambda item: item[1],
                reverse=True
            )[:10]
        }

        logging.info("Топ-10 найважливіших ознак:")
        for feature, importance in self.top_features_.items():
            logging.info(f"{feature}: {importance}")

        return self.top_features_

    def advanced_evaluation(self, X_test, y_test):
        """
        Розширена оцінка з додатковими метриками
        """
        y_pred = self.classifier.predict(X_test)
        y_pred_proba = self.classifier.predict_proba(X_test)


        accuracy = accuracy_score(y_test, y_pred)
        class_report = classification_report(
            y_test,
```

```
    y_pred,
    target_names=self.label_encoder.classes_
)
conf_matrix = confusion_matrix(y_test, y_pred)



plt.figure(figsize=(10, 8))
for i, category in enumerate(self.label_encoder.classes_):
    y_true_binary = (y_test == i).astype(int)
    fpr, tpr, _ = roc_curve(y_true_binary, y_pred_proba[:, i])
    roc_auc = auc(fpr, tpr)

    plt.plot(
        fpr, tpr,
        label=f'ROC {category} (AUC = {roc_auc:.2f})'
    )

plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Хибно позитивні')
plt.ylabel('Истинно позитивні')
plt.title('Криві ROC за категоріями')
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()



cv_scores = cross_val_score(
```

```
            self.classifier, X_test, y_test, cv=5
        )

        logging.info(f"Результати крос-валідації: {cv_scores}")
        logging.info(f"Середня точність крос-валідації: {cv_scores.mean():.2f} (+/-
{cv_scores.std() * 2:.2f})")

        return {
            'accuracy': accuracy,
            'report': class_report,
            'confusion_matrix': conf_matrix,
            'cross_validation_scores': cv_scores
        }

    def save_model(self, output_path='model_state.json'):
        """
        Збереження стану моделі
        """
        model_state = {
            'top_features': self.top_features_,
            'class_labels': list(self.label_encoder.classes_),
            'model_config': self.config
        }

        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(model_state, f, ensure_ascii=False, indent=4)

        logging.info(f"Стан моделі збережено в {output_path}")
```

```
def main():

    class SyntheticDataGenerator:
        @staticmethod
        def generate_messages():
            categories = [
                'Технічна підтримка',
                'Замовлення',
                'Скарга',
                'Пропозиція',
                'Запитання'
            ]

            templates = {
                'Технічна підтримка': [
                    "Не працює {noun}",
                    "Проблема з {noun}",
                    "Потрібна допомога {noun}"
                ],
                'Замовлення': [
                    "Хочу замовити {noun}",
                    "Ціна на {noun}",
                    "Умови придбання {noun}"
                ],
                'Скарга': [
                    "Незадоволений {noun}",
                    "Погана якість {noun}",
                    "Проблема з {noun}"
                ],
```

```
    'Пропозиція': [
        "Ідея покращити {noun}",
        "Пропозиція щодо {noun}",
        "Можливість вдосконалити {noun}"
    ],
    'Запитання': [
        "Як працює {noun}",
        "Деталі про {noun}",
        "Інформація про {noun}"
    ]
}

nouns = [
    "комп'ютер", "телефон", "інтернет", "додаток", "сервіс",
    "процес", "система", "обладнання", "техніка", "продукт"
]

messages = []
labels = []

for category, cat_templates in templates.items():
    for _ in range(100):
        template = random.choice(cat_templates)
        noun = random.choice(nouns)
        message = template.format(noun=noun)
        messages.append(message)
        labels.append(category)

return pd.DataFrame({
```

```
            'text': messages,
            'category': labels
        })


data = SyntheticDataGenerator.generate_messages()


classifier = AdvancedMessageClassifier()


X, y = classifier.vectorize_data(data)


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)


top_features = classifier.feature_analysis(X_train, y_train)


evaluation_results = classifier.advanced_evaluation(X_test, y_test)


classifier.save_model()


test_messages = [
```

```
        "Не працює мій комп'ютер",
        "Хочу замовити телефон",
        "Маю пропозицію покращити сервіс",
        "У мене проблема з інтернетом"
    ]

    print("\nПриклади передбачення категорій:")
    for message in test_messages:

        processed_text = classifier.preprocess_text(message)
        vectorized_text = classifier.vectorizer.transform([processed_text])
        prediction = classifier.classifier.predict(vectorized_text)
        probabilities = classifier.classifier.predict_proba(vectorized_text)

        print(f"\nПовідомлення: '{message}'")
        for category, prob in zip(classifier.label_encoder.classes_, probabilities[0]):
            print(f"- {category}: {prob*100:.2f}%")

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