ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методики автоматизованого семантичного аналізу та класифікації текстових масивів на основі інтелектуальних алгоритмів Data Science»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Олексій КАЗЬМІРЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Олексій КАЗЬМІРЧУК

Керівник: Володимир САДОВЕНКО
канд. фіз-мат. наук., доц.

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Казьмірчуку Олексію Геннадійовичу

1. Тема кваліфікаційної роботи: «Розробка методики автоматизованого семантичного аналізу та класифікації текстових масивів на основі інтелектуальних алгоритмів Data Science»

керівник кваліфікаційної роботи Володимир САДОВЕНКО, канд. фіз-мат. наук

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з NLP та Data Science, текстові масиви для навчання, методи семантичного аналізу (TF-IDF, Embeddings, BERT), архітектури нейронних мереж (Transformer, LSTM).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі NLP та існуючих методів семантичного аналізу.
2. Математичний опис моделі автоматизованого семантичного аналізу та класифікації.

3. Розробка методики автоматизованого аналізу, що враховує лінгвістичну специфіку різних мов.
4. Реалізація програмного комплексу та експериментальне дослідження.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Структурна схема запропонованої методики.
   2. Загальна архітектура розробленої системи семантичного аналізу.
   3. Порівняльна таблиця ефективності алгоритмів.
   4. Екранні форми результатів тестування системи.
   5. Матриця плутанини.
   6. Графіки динаміки навчання моделі.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10.-03.11.2025	
2	Дослідження методів семантичного аналізу та класифікації	04.11.-08.11.2025	
3	Аналіз існуючих інтелектуальних алгоритмів класифікації текстів Data Science та математичної моделі процесу класифікації	09.11.-12.11.2025	
4	Розробка методики та архітектури "Model-per-Language"	12.11.-28.11.2025	
5	Програмна реалізація системи (Python, TensorFlow)	28.11.-05.12.2025	
6	Проведення експериментальних досліджень та аналіз результатів	06.12.-09.12.2025	
7	Оформлення роботи: вступ, висновки, реферат	10.12.-15.12.2025	
8	Розробка демонстраційних матеріалів	16.12.-19.12.2025	

Здобувач вищої освіти _____ Олексій КАЗЬМІРЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 76 стор., 2 табл., 19 рис., 64 джерела.

Мета роботи – удосконалення процесу та підвищення точності обробки текстових масивів за рахунок використання розробленої комплексної методики автоматизованого семантичного аналізу та класифікації на основі інтелектуальних алгоритмів Data Science.

Об'єкт дослідження – процес автоматизованої обробки та аналізу неструктурованих текстових масивів.

Предмет дослідження – методика, моделі та алгоритми семантичного аналізу і класифікації текстів на основі методів Data Science.

В роботі досліджено методи автоматизованого семантичного аналізу та класифікації текстових масивів на основі інтелектуальних алгоритмів Data Science. Проведено огляд існуючих підходів до обробки природної мови, зокрема класичних методів (Naive Bayes, SVM) та сучасних нейромережевих архітектур (RNN, LSTM, Transformers). Розроблено комплексну методику з використанням архітектури «Model-per-Language» та програмно реалізовано ключові функціональні можливості системи, зокрема: адаптивний препроцесинг для різних мовних груп, векторизацію даних методом TF-IDF, класифікацію текстів за допомогою повнозв'язних нейронних мереж та створення REST API. Проведено експериментальне дослідження ефективності методики на власному мультимовному корпусі та еталонних наборах даних, досягнуто F1-Score 97%. В роботі використано технології Python, TensorFlow, Keras, веб-фреймворк Flask, платформу контейнеризації Docker, а також бібліотеки NumPy, Pandas та NLTK.

КЛЮЧОВІ СЛОВА: DATA SCIENCE, NLP, СЕМАНТИЧНИЙ АНАЛІЗ, КЛАСИФІКАЦІЯ ТЕКСТІВ, НЕЙРОННІ МЕРЕЖІ, TF-IDF, PYTHON, DOCKER.

ABSTRACT

Text part of the master's qualification work: 76 pages, 2 table, 19 pictures, 64 sources.

The purpose of the work is to improve the process and increase the accuracy of text array processing by using the developed complex methodology of automated semantic analysis and classification based on intelligent Data Science algorithms.

Object of research – processes of automated processing and analysis of unstructured text arrays.

Subject of research – methodology, models, and algorithms for semantic analysis and text classification based on Data Science methods.

The thesis investigates methods of automated semantic analysis and classification of text arrays based on intelligent Data Science algorithms. An overview of existing approaches to Natural Language Processing is provided, including classical methods (Naive Bayes, SVM) and modern neural network architectures (RNN, LSTM, Transformers). A comprehensive methodology was developed using the "Model-per-Language" architecture, and the key functional capabilities of the system were implemented, specifically: adaptive preprocessing for different language groups, data vectorization using the TF-IDF method, text classification using fully connected neural networks, and the creation of a REST API. An experimental study of the methodology's effectiveness was conducted on a custom multilingual corpus and benchmark datasets, achieving an F1-Score of 97%. The work utilizes Python, TensorFlow, Keras, the Flask web framework, the Docker containerization platform, and the NumPy, Pandas, and NLTK libraries.

KEYWORDS: DATA SCIENCE, NLP, SEMANTIC ANALYSIS, TEXT CLASSIFICATION, NEURAL NETWORKS, TF-IDF, PYTHON, DOCKER.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

API	-	Application Programming Interface
BERT	-	Bidirectional Encoder Representations from Transformers
BoW	-	Bag-of-Words
CBOW	-	Continuous Bag-of-Words
CNN	-	Convolutional Neural Networks
GloVe	-	Global Vectors
GRU	-	Gated Recurrent Unit
LLM	-	Large Language Models
LSTM	-	Long Short-Term Memory
ML	-	Machine Learning
MLOps	-	Machine Learning Operations
NER	-	Named Entity Recognition
NLP	-	Natural Language Processing
OOV	-	Out Of Vocabulary
REST	-	Representational State Transfer
RNN	-	Recurrent Neural Networks
SOTA	-	State-of-the-Art
SPA	-	Single Page Application
SRL	-	Semantic Role Labeling
SVM	-	Support Vector Machines
TF-IDF	-	Term Frequency-Inverse Document Frequency
WSD	-	Word Sense Disambiguation

ВСТУП

Актуальність теми. В сучасну епоху стрімкого зростання обсягів даних, що генеруються щодня, текстові дані є одним з найбільш поширених і цінних типів інформації. З розвитком сучасних інформаційних технологій та глобалізацією цифрового простору спостерігається зростання обсягів неструктурованих текстових даних: електронних листів, повідомлень у соціальних мережах, новинних стрічок та наукових публікацій. Ефективна обробка цих масивів вручну є неможливою, тому постає необхідність вдосконалення систем автоматизованого аналізу для вирішення викликів сучасного світу. Однією з ключових галузей є розробка методик автоматизованого семантичного аналізу та класифікації текстових масивів на основі інтелектуальних алгоритмів Data Science.

Ці методи мають великий потенціал у різноманітних сферах, включаючи бізнес-аналітику, моніторинг медіа-простору, підтримку клієнтів та наукові дослідження. Хоча класичні методи машинного навчання (такі як Naive Bayes та SVM) внесли свій внесок, забезпечуючи швидкість та інтерпретованість, вони мають обмеження у розумінні семантичного контексту. Сучасний напрямок розвитку зосереджується на застосуванні глибокого навчання (Deep Learning) та контекстуалізованих моделей (BERT, Transformers) для покращення точності та глибини розуміння змісту. Використання нейронних мереж та векторних представлень слів у цьому контексті відкриває нові можливості та дозволяє створювати більш точні системи, здатні враховувати полісемію та синонімію.

Дана дипломна робота присвячена аналізу та дослідженню сучасних методів обробки природної мови (NLP), зокрема використанню гібридних підходів Data Science для цієї мети. Під час роботи виконано детальний аналіз різних підходів, їх переваг та недоліків, а також розроблено комплексну методику з використанням архітектури «одна модель для кожної мови» (Model-per-Language) з метою покращення процесу класифікації текстів у багатомовному середовищі.

Робота спрямована на вирішення важливих проблем в області обробки природної мови та дослідження перспективних напрямків інтеграції лінгвістичних знань з алгоритмами машинного навчання.

Таким чином, завдання розробки методики автоматизованого семантичного аналізу та класифікації текстових масивів на основі інтелектуальних алгоритмів Data Science є сучасним та актуальним.

Об'єкт дослідження – процес автоматизованої обробки та аналізу неструктурованих текстових масивів.

Предмет дослідження – методика, моделі та алгоритми семантичного аналізу і класифікації текстів на основі методів Data Science.

Мета дослідження – удосконалення процесу та підвищення точності обробки текстових масивів за рахунок використання розробленої комплексної методики автоматизованого семантичного аналізу та класифікації на основі інтелектуальних алгоритмів Data Science.

Для досягнення мети дослідження визначені основні **задачі**:

- проаналізувати предметну галузь NLP, методи семантичного аналізу та класифікації текстових масивів;
- навести математичний опис моделі автоматизованого семантичного аналізу та класифікації на основі інтелектуальних алгоритмів Data Science;
- розробити методику автоматизованого аналізу та класифікації текстових масивів, яка враховує лінгвістичну специфіку різних мов;
- реалізувати програмний комплекс з використанням сучасного стеку (Python, TensorFlow, Docker), який підтримує повний цикл MLOps: від навчання до розгортання;
- провести експериментальне дослідження на основі тестування методу з різними вхідними умовами для підтвердження ефективності розробленої методики.

Вирішення цих задач сприяє створенню ефективних інструментів для перетворення неструктурованих текстових даних у структуровані знання, що є критично важливим для сучасних інтелектуальних систем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ АВТОМАТИЗОВАНОГО АНАЛІЗУ ТЕКСТІВ

1.1 Обробка природної мови (NLP) як науковий напрям

Обробка природної мови, яка широко відома за англомовною абревіатурою NLP (Natural Language Processing), є фундаментальним та динамічно зростаючим міждисциплінарним науковим напрямом. Вона розташована на перетині кількох ключових галузей знань: комп’ютерних наук (Computer Science), штучного інтелекту (Artificial Intelligence) та обчислювальної лінгвістики (Computational Linguistics). Глобальна мета NLP полягає у розробці теорій, методологій, алгоритмів та програмних систем, які надають комп’ютерам здатність не просто зберігати чи відображати, але й глибоко "розуміти", інтерпретувати, обробляти та генерувати людську мову в її природній формі – як текстовій, так і усній.

Як зазначають Деніел Джурафскі та Джеймс Мартін [39], ключова відмінність NLP від звичайної обробки даних полягає у використанні знань про мову. Якщо утиліта підрахунку слів оперує лише символами, то система NLP повинна моделювати лінгвістичні структури на морфологічному, синтаксичному та семантичному рівнях. У вітчизняній науковій школі, зокрема у працях М.М. Глибовця та О.В. Олецького [6], наголошується, що системи NLP є підкласом інтелектуальних систем, які намагаються емулювати когнітивні процеси людини, пов'язані з вербалізацією думок та інтерпретацією отриманих повідомлень. Ю.В. Нікольський [13] додає, що ефективність таких систем напряму залежить від повноти та несуперечливості баз знань, які лежать в їх основі.

Обробка природної мови – технологія, яка активно розвивається та допомагає бізнесу отримати максимальну користь від використання штучного інтелекту.

Людська мова є надзвичайно складною, багатогранною та нюансованою системою. Вона не лише передає фактологічну інформацію, але й виражає емоції,

інтенції, сарказм, культурний контекст та приховані смисли. Це робить автоматичну обробку мови однією з найскладніших проблем у галузі штучного інтелекту.

Обробка природної мови дозволяє використовувати технології, які в іншому випадку можуть бути важкими у використанні. Це також дозволяє комп'ютерам розуміти текст і мову таким чином, що вони не могли до NLP. [22]

Сьогодні важко уявити галузь, у якій не застосовується технологія NLP.

NLP застосовується при щоденній взаємодії з мобільними пристроями, наприклад через Siri, використанні розумних помічників, таких як Cortana в операційній системі Windows, в автоматичному визначенні спам-пошти у Gmail, при використанні онлайн-перекладачів тощо.

Мовлення людини відрізняється від мовлення машини! Мова машин проста, логічна і структурована, своєю чергою людська мова є дуже різноманітна з точки зору побудови словесних конструкцій, використання багатозначних слів та контекстного значення розмови в цілому. [21]

Люди спілкуються словами, а машини – числами. Будь-яке звернення людини до машини голосом або текстом потребує його перекладу на машинну мову чисел. Далі машина повинна зрозуміти, ідентифікувати та розпізнати сутність наміру людини, з яким вона до неї звертається. І після цього – прийняти певне рішення, виконати якусь функцію, чи відповісти. Цей процес ідентичний спілкуванню людей між собою, просто ми не придаємо значення, завдяки чому та яким чином ми вирізняємо той чи інший зміст у нашому спілкуванні. Машина ж потребує детальних пояснень на зрозумілій їй логічній мові. Власне для цього всього і призначена технологія NLP. [21]

Історичний розвиток та еволюція підходів

Розвиток NLP як науки можна умовно поділити на кілька ключових етапів, кожен з яких характеризувався домінуючою парадигмою.

Символічний етап (1950-ті – 1980-ті рр.)

Цей ранній період був тісно пов'язаний із загальними ідеями символічного штучного інтелекту. Дослідники, натхненні роботами Ноама Чомскі у галузі генеративної граматики, намагалися створити системи, засновані на величезних наборах жорстко кодифікованих правил. Це були експертні системи, які намагалися розібрати речення за допомогою детально прописаних граматичних структур та лексичних баз даних (словників). Яскравим прикладом амбіцій того часу були ранні спроби машинного перекладу (наприклад, Джорджтаунський експеримент). Однак цей підхід виявився надзвичайно крихким: він погано масштабувався, не міг впоратися з мовними винятками, метафорами та багатозначністю (полісемією), а створення правил вручну було надзвичайно трудомістким процесом.

Статистичний етап (1990-ті – 2000-ні рр.)

З появою значних обсягів машиночитаних текстів (корпусів) та зростанням обчислювальних потужностей відбулася так звана "статистична революція". Фокус змістився від створення ідеальних правил до навчання на реальних даних. Алгоритми почали використовувати імовірнісні моделі для вирішення конкретних завдань. Такі методи, як N-грами, приховані Марковські моделі (HMM) для тегування частин мови, та класичні алгоритми машинного навчання (як-от Наївний Баєс, дерева рішень та, особливо, метод опорних векторів (SVM)) стали домінуючими. Цей етап заклав основи сучасного Data Science підходу до аналізу текстів, продемонструвавши, що моделі, навчені на великих масивах даних, часто працюють ефективніше за складні, але обмежені ручні правила.

Нейромережевий етап (з 2010-х рр. – дотепер)

Цей етап, що триває і зараз, характеризується домінуванням глибокого навчання (Deep Learning). Поштовхом стали дві ключові інновації:

- векторні представлення слів (Word Embeddings): такі моделі, як Word2Vec та GloVe, дозволили представляти слова у вигляді щільних числових векторів у багатовимірному просторі, де семантично близькі слова мають близькі вектори. Це

вперше дало змогу алгоритмам "розуміти" семантичні зв'язки (наприклад, "король" - "чоловік" + "жінка" ≈ "королева");

- глибокі архітектури: спочатку рекурентні нейронні мережі (RNN) та їхні вдосконалені варіанти LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit) показали видатні результати у задачах, що вимагають врахування послідовності (як-от машинний переклад). Згодом, поява архітектури Трансформерів (Transformers) та механізму уваги (attention), що лежить в їхній основі (моделі як BERT, GPT, T5), спричинила справжній прорив. Ці моделі здатні враховувати контекст цілих документів, ефективно навчатися на гігантських текстових масивах (так звані "великі мовні моделі" – LLM) та встановлювати нові стандарти якості (State-of-the-Art) практично у всіх задачах NLP.

Основні рівні та задачі обробки природної мови

Для розв'язання яких задач використовується NLP? На перший погляд, може здаватися, що ця технологія призначена лише для розпізнавання голосу або тексту, але насправді це лише поверхневі завдання, які вона здатна вирішувати. Сама ж технологія призначена для набагато глибших та складніших завдань. [21]

Для того, щоб комп'ютер міг "зрозуміти" текст, він має послідовно або паралельно вирішити низку завдань на різних лінгвістичних рівнях:

- морфологічний аналіз – робота на рівні слова. Включає токенізацію (розбиття тексту на окремі слова або "токени"), стемінг (виділення основи слова, часто шляхом грубого відсікання закінчень, наприклад, "аналіз", "аналізу" -> "аналіз") та лематизацію (приведення слова до його базової словникової форми, леми, наприклад, "йшли", "йшов", "йдемо" -> "йти").

- синтаксичний аналіз (парсинг) – аналіз структури речення. Включає тегування частин мови (POS-tagging), тобто визначення для кожного слова його граматичної ролі (іменник, дієслово, прикметник тощо), та власне синтаксичний розбір – побудову дерева залежностей або дерева складових речення, щоб зрозуміти, які слова з якими пов'язані та яку роль виконують (що є підметом, що є присудком, які є додатки та обставини).

- семантичний аналіз – аналіз значення та смислу на рівні слів, речень та документів. Це ключова галузь для даної магістерської роботи.

До задач семантичного аналізу належать:

- розпізнавання іменованих сутностей (NER) – ідентифікація та класифікація об'єктів реального світу (імена людей, назви організацій, географічні локації, дати);
- визначення семантичних ролей (SRL) – встановлення, хто що зробив, кому, коли і де (наприклад, [Агент] "компанія" [Предикат] "випустила" [Пацієнт] "новий продукт");
- усунення неоднозначності слів (WSD) – визначення, в якому саме значенні вжито багатозначне слово (наприклад, "замок" як будівля чи як пристрій).

- прагматичний аналіз (аналіз дискурсу) – найвищий і найскладніший рівень, що стосується інтерпретації мови у широкому контексті. Включає вирішення анафори (встановлення, на що посилаються займенники "він", "вона", "це"), аналіз зв'язності тексту (дискурсу) та розуміння прихованих намірів, іронії чи сарказму.

Це відображено на рисунку 1.1.

Рис. 1.1 Ієрархія рівнів аналізу природної мови

Прикладні задачі та зв'язок з Data Science

Сучасна NLP тісно інтегрована з Data Science, оскільки інтелектуальні алгоритми Data Science [32, 53] (зокрема, з машинного [33] та глибокого навчання [34, 40, 43]) є основним інструментарієм для вирішення прикладних NLP-задач. Ці задачі є центральними для бізнесу, науки та повсякденного життя в епоху цифрової трансформації.

Автоматичне віднесення документа до однієї чи кількох із заздалегідь визначених категорій. Приклади: сортування електронних листів (спам/не спам), категоризація новин, аналіз тональності (сентимент-аналіз – позитивний, негативний, нейтральний відгук). Це задача класифікації текстів. І це одна з центральних тем даного дослідження.

До задач, які можливо вирішити за допомогою алгоритмів Data Science можна віднести: машинний переклад (MT) – автоматичний переклад тексту з однієї мови на іншу; інформаційний пошук (IR) – пошук релевантних документів у великому корпусі за запитом користувача (основа пошукових систем); запитально-відповідні системи (Q&A) – системи, що здатні давати пряму відповідь на запитання користувача (наприклад, чат-боти, віртуальні асистенти); автоматичне реферування (Summarization) – створення короткого, змістовного викладу (анотації) для великого тексту або набору документів.

Таким чином, обробка природної мови (NLP) є фундаментальною науковою дисципліною, що забезпечує методологічний та алгоритмічний апарат для перетворення неструктурованих текстових даних, найпоширенішого типу даних у сучасному світі, у структуровані знання. Розробка ефективних методик семантичного аналізу та класифікації, що є предметом цієї магістерської роботи, безпосередньо лежить у руслі вирішення цих актуальних науково-практичних завдань.

1.2 Семантичний аналіз текстів: задачі та методи

Семантичний аналіз є однією з найбільш фундаментальних, складних та водночас критично важливих галузей в межах обробки природної мови (NLP). Якщо синтаксичний аналіз, про який йшлося раніше, відповідає на питання "Як побудоване речення?" (його граматична структура, зв'язки між словами), то семантичний аналіз намагається дати відповідь на питання "Що це речення означає?". Він фокусується на вилученні, інтерпретації та представленні смислу, закладеного в текстових даних, виходячи за рамки простої граматичної коректності. Це процес наділення комп'ютерних систем здатністю "розуміти" людську мову на рівні значень, інтенцій та взаємозв'язків. (див. рис. 1.2)

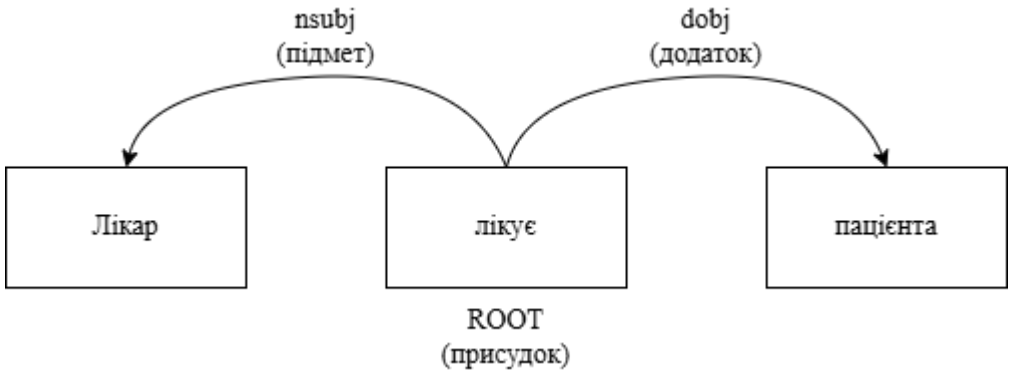


Рис. 1.2 Приклад дерева синтаксичних залежностей.

Задачею семантичного аналізу в класичному розумінні є визначення семантичної структури речень та тексту в цілому. Входом для цього етапу є розібрані синтаксичні структури речень тексту. [1, с. 379]

Семантика – це той самий «смисловий» багаж, який несе та чи інша лексична одиниця у даному конкретному контексті.

Фундаментальні виклики семантичного аналізу

Семантичний аналіз даних передбачає вивчення тексту для виявлення закономірностей, тем і настроїв, які можуть бути неочевидними з першого погляду. Він дозволяє витягти глибші значення і зв'язки в тексті, забезпечуючи більш тонке розуміння змісту. [12, с. 141]

Складність автоматичного розуміння смислу обумовлена самою природою людської мови, яка є набагато менш формалізованою та логічною, ніж мови програмування. До основних викликів, які покликаний вирішувати семантичний аналіз, належать: лексична неоднозначність, синонімія, контекстуальна залежність, прагматика та імпліцитні знання, анафора та кореферентність.

Лексична неоднозначність (полісемія) – це явище, коли одне й те саме слово має декілька різних, не пов'язаних між собою значень. Наприклад, слово "ключ" може означати пристрій для відкриття замка, джерело води, музичний знак або підказку для вирішення проблеми, "замок" може бути фортецею або замикаючим пристроєм. Без контексту комп'ютер не в змозі визначити, яке саме значення мається на увазі.

Синонімія – існування багатьох слів (синонімів), що позначають одне й те саме або дуже близьке поняття (наприклад, "гарний", "красивий", "вродливий", "чудовий"). Для машини це різні послідовності символів, але семантично вони мають бути опрацьовані як близькі.

Контекстуальна залежність – смисл слова, фрази або навіть цілого речення кардинально залежить від оточуючих його слів (локальний контекст) та загальної теми документа (глобальний контекст) – фраза "холодний прийом" має зовсім інше значення, ніж "холодний чай".

Прагматика та імпліцитні знання – мова часто передає більше, ніж говорить буквально. Сюди належать іронія, сарказм, метафори, культурні алюзії та непрямі висловлювання (імплікатури). Наприклад, речення "Чудова погода для прогулянки" сказане під час зливи, насправді означає прямо протилежне. Розуміння таких нюансів вимагає не лише знання мови, але й знань про реальний світ.

Анафора та кореферентність – це проблема зв'язування займенників та інших референтних виразів з сутностями, про які йшлося раніше. У тексті "Ілон Маск заснував SpaceX. Він є її генеральним директором", машина має зрозуміти, що "він" – це "Ілон Маск", а "її" – це "SpaceX".

Для подолання цих викликів та досягнення мети розуміння смислу, семантичний аналіз поділяється на низку конкретних, більш вузьких науково-технічних завдань.

Розпізнавання іменованих сутностей (Named Entity Recognition, NER)

Це одна з найбільш базових і водночас важливих задач семантичного аналізу. У своєму всеосяжному огляді методів NER, Д. Надеу та С. Секіне [49] формалізують цю задачу як процес ідентифікації та класифікації жорстких десигнаторів (rigid designators) у тексті. Дослідники виділяють еволюційний перехід від систем, заснованих на правилах ручного написання (hand-crafted rules), які домінували на ранніх етапах (MUC-6), до методів машинного навчання, що дозволяють автоматично виявляти ознаки контексту. Сучасний стан галузі, представлений у роботах Г. Лампла (G. Lample) [42], демонструє перевагу гібридних нейромережевих архітектур (наприклад, BiLSTM-CRF), які здатні враховувати як символьні представлення слів, так і їхнє оточення без використання зовнішніх словників.

Стандартні категорії включають PER (персони), ORG (організації), LOC (географічні локації), DATE (дати), TIME (час), MONEY (грошові суми), PRODUCT (назви продуктів) тощо. Наприклад, у реченні "Компанія [Apple]ORG була заснована [Стівом Джобсом]PER у [Купертіно]LOC", NER-система має не просто знайти ці слова, але й присвоїти їм відповідні теги. Це є першим кроком до побудови фактологічного розуміння тексту.

Чому NER має значення, бо підвищується семантична ясність і контекстуальне розуміння, підвищується точність вилучення інформації, підтримуються різні додатки НЛП, такі як аналіз настроїв, SEO-оптимізація та класифікація контенту.

Розпізнавання іменованих об'єктів значно покращує семантичне розуміння, вилучення даних та ефективність НЛП. Ефективне впровадження NER підвищує точність і релевантність додатків, починаючи від SEO і закінчуючи аналізом настроїв.

Визначення семантичних ролей (Semantic Role Labeling, SRL)

Ця задача фокусується на розумінні структури подій, описаних у реченні. Вона полягає у визначенні предиката (зазвичай, дієслова) та його семантичних аргументів (учасників ситуації). SRL відповідає на питання "Хто що зробив? Кому? Де? Коли? Як?". Наприклад, для речення "Студент швидко написав магістерську роботу вчора", система SRL може видати:

- Предикат: "написав"
- Агент (Хто?): "Студент"
- Пацієнт (Що?): "магістерську роботу"
- Темпоральний маркер (Коли?): "вчора"
- Модифікатор (Як?): "швидко"

Це дозволяє отримати структуроване представлення інформації, що є значно глибшим за синтаксичне дерево.

Вирішення лексичної неоднозначності (Word Sense Disambiguation, WSD)

Це класична задача семантики, яка безпосередньо спрямована на боротьбу з полісемією. Мета WSD – для кожного багатозначного слова в конкретному контексті обрати правильне значення (або "сенс") із заздалегідь визначеного набору, наприклад, зі словника або тезауруса (як WordNet).

Видобування відношень (Relation Extraction, RE)

Якщо NER знаходить сутності, то RE йде на крок далі і намагається знайти та класифікувати семантичні зв'язки між цими сутностями. Наприклад, з тексту "Сергій Кавун народився в Одесі, Україна" система RE може видобути тріаду: (Сергій Кавун, [народився_в], Одесі) та (Одеса, [знаходиться_в], Україна). Це є ключовим для автоматичного наповнення баз знань.

Аналіз тональності (Sentiment Analysis)

Це надзвичайно популярна прикладна задача семантичного аналізу, яка полягає в автоматичному визначенні емоційного забарвлення тексту. Вона класифікує текст як позитивний, негативний або нейтральний. Більш просунуті моделі можуть визначати конкретні емоції (радість, гнів, сум) або аналізувати

тональність стосовно конкретних аспектів об'єкта (наприклад, у відгуку про телефон: "Камера [позитивно], але батарея [негативно]").

Розпізнавання текстового наслідування (Recognizing Textual Entailment, RTE)

Це задача визначення логічного зв'язку між двома текстовими фрагментами: "текстом" (T) та "гіпотезою" (H). Система має визначити, чи випливає гіпотеза з тексту (Entailment), чи суперечить йому (Contradiction), чи не пов'язана з ним (Neutral). Це складна задача, що вимагає глибокого розуміння смислу, синонімії та логіки.

Еволюція методів семантичного аналізу

Підходи до вирішення цих задач кардинально еволюціонували, що безпосередньо пов'язано з розвитком Data Science та штучного інтелекту.

Символічні методи та методи знання – ранні підходи (1970-1990-ті) покладалися на великі, створені вручну бази знань, онтології та тезауруси (як WordNet). Алгоритми намагалися зіставляти слова з поняттями в цих базах та використовувати прописані вручну правила для виведення смислу. Ці методи були точними, але дуже крихкими, трудомісткими у створенні та нездатними обробляти мовні нюанси, яких не було в базі знань.

Класичні статистичні методи та машинне навчання – з появою великих анотованих корпусів (1990-2000-ні), фокус змістився на статистику. Такі алгоритми, як Naive Bayes, Support Vector Machines (SVM) та Maximum Entropy (MaxEnt), почали використовуватись для задач класифікації (наприклад, аналізу тональності). Для цього текст перетворювався у вектор за допомогою таких ознак, як Bag-of-Words (Мішок слів) або TF-IDF. Ці методи були значно гнучкішими, але сильно залежали від ручного проектування ознак (feature engineering).

Сучасні методи глибокого навчання (Deep Learning) – починаючи з 2010-х років, глибоке навчання спричинило революцію в семантичному аналізі.

Векторні представлення слів (Word Embeddings)

Моделі Word2Vec (Міколова та ін.) та GloVe стали першим проривом. Вони дозволили представляти слова у вигляді щільних векторів у багатовимірному

просторі, де семантична близькість слів відповідає геометричній близькості їхніх векторів. Це дало змогу моделям автоматично "вивчити" синонімію та семантичні аналогії.

Архітектури глибоких мереж

Рекурентні (RNN, LSTM, GRU) та згорткові (CNN) нейронні мережі почали застосовуватись поверх цих векторів, дозволяючи враховувати послідовність та локальний контекст слів у реченні, що значно покращило якість NER, SRL та аналізу тональності.

Контекстуалізовані моделі (Трансформери)

Революційним кроком у розвитку NLP стала поява архітектури Transformer, представленої групою дослідників Google Brain. На відміну від рекурентних мереж, Трансформери відмовляються від послідовної обробки слів на користь механізму Self-Attention (самоуваги).

Поява архітектури Трансформер та моделей на її основі (зокрема BERT, RoBERTa, GPT) стала визначальним моментом. На відміну від Word2Vec, де слово "ключ" завжди має однаковий вектор, BERT генерує контекстуалізоване представлення. Вектор для "ключ" у "музичний ключ" буде зовсім іншим, ніж у "дверний ключ". Це дозволило ефективно вирішувати задачі, пов'язані з лексичною неоднозначністю, та досягти нового рівня якості (State-of-the-Art) практично у всіх перелічених вище задачах семантичного аналізу.

Цей механізм дозволяє моделі оцінювати важливість кожного слова в контексті всіх інших слів у реченні одночасно. Це значно прискорює навчання та дозволяє краще інтерпретувати семантичний зміст складних текстів, що є критичним для задач автоматизованої класифікації.

Таким чином, семантичний аналіз тексту є комплексною дисципліною, що об'єднує лінгвістичні знання з потужними інтелектуальними алгоритмами Data Science. Саме розвиток цих алгоритмів, особливо в галузі глибокого навчання, відкрив шлях до розробки ефективних методик автоматизованого аналізу та класифікації текстових масивів, що і є центральною темою даного магістерського дослідження.

1.3 Класифікація текстових масивів на основі Data Science

Фабриціо Себастьяні визначає автоматизовану категоризацію текстів (АТС) як задачу побудови алгоритмічного класифікатора ф: $D \times C \longrightarrow \{T, F\}$, де D – це домен документів, а C – набір визначених категорій. Він підкреслює, що домінуючим підходом у цій галузі стало індуктивне навчання (Machine Learning), яке замінило інженерію знань. Головною перевагою ML-підходу, за Себастьяні, є його здатність до узагальнення та переносимість на нові предметні домени без необхідності переписування правил експертами [58].

Класифікація текстових масивів (також відома як "категоризація текстів" або "текстова класифікація") є однією з найбільш фундаментальних та широко застосовуваних задач в області обробки природної мови (NLP) та, водночас, класичною проблемою, що вирішується методами Data Science. Суть цієї задачі полягає в автоматичному призначенні одного або декількох заздалегідь визначених категоріальних міток (класів) для текстового документа на основі його змісту. Це є яскравим прикладом керованого машинного навчання (Supervised Machine Learning), оскільки для тренування моделі потрібен великий набір документів, де для кожного з них вже відома правильна категорія (так звана "розмічена вибірка").

У контексті даної магістерської роботи, класифікація є практичною реалізацією та операціоналізацією семантичного аналізу. Якщо семантичний аналіз намагається зрозуміти "що" означає текст на глибокому рівні, то класифікація використовує це розуміння (явно чи неявно) для прийняття конкретного рішення: до якої теми, жанру, тональності чи категорії належить даний текстовий масив.

Актуальність цієї задачі зумовлена експоненціальним зростанням обсягів неструктурованих текстових даних (електронні листи, соціальні мережі, новини, наукові статті, відгуки клієнтів). Ручна обробка таких масивів є неможливою, тому автоматизована класифікація стає ключовим інструментом для:

- Фільтрації спаму: класична бінарна класифікація (спам / не спам).
- Аналізу тональності (Sentiment Analysis): класифікація відгуків, коментарів чи постів на позитивні, негативні або нейтральні.

- Тематичного рубрикування: автоматичне сортування новинних статей за рубриками (спорт, політика, технології), наукових публікацій за галузями знань.
- Маршрутизації запитів: автоматичне спрямування звернень клієнтів у службі підтримки до відповідного відділу (технічна підтримка, продажі, бухгалтерія).
- Виявлення мови (Language Detection): класифікація тексту за мовою, якою він написаний.
- Виявлення фейкових новин або токсичного контенту: критично важлива задача для модерації онлайн-платформ.

Data Science надає чітку, структуровану методологію (pipeline) для вирішення задачі класифікації. Цей процес перетворює хаотичні, неструктуровані текстові дані у формалізовану постановку задачі, яку можуть вирішувати інтелектуальні алгоритми.

Етап 1: Збір та підготовка даних

На цьому етапі формується корпус документів – навчальна вибірка. Кожен документ у цій вибірці повинен мати чітку мітку класу. Якість та репрезентативність цих даних є визначальним фактором для успішності майбутньої моделі.

Етап 2: Попередня обробка тексту (Text Preprocessing)

Це один з найважливіших етапів, оскільки текстові дані у "сирому" вигляді містять багато шуму, який не несе семантичного навантаження, але може завадити алгоритмам.

- Нормалізація: приведення тексту до єдиного регістру (зазвичай, нижнього).
- Токенізація: процес розбиття суцільного тексту на окремі смислові одиниці – токени (найчастіше це слова, але можуть бути і словосполучення чи символи).
- Очищення даних: видалення пунктуації, HTML-тегів, URL-адрес, цифр та іншої нерелевантної інформації.

- Видалення стоп-слів (Stop-word removal): видалення слів, що часто зустрічаються, але не мають значного семантичного ваги (наприклад, "і", "в", "на", "але", "був").
- Морфологічна нормалізація: приведення слів до їхньої базової форми для зменшення розмірності простору ознак.

Стемінг (Stemming) – більш грубий процес відсікання закінчень (наприклад, "аналіз", "аналізу" -> "аналіз") та лематизація (Lemmatization) – більш складний процес приведення слова до його словникової форми (леми) з урахуванням частини мови (наприклад, "йшли", "йдемо" -> "йти").

Етап 3: Векторизація (текстове представлення)

Це серце застосування Data Science до текстів. На цьому етапі відбувається перетворення очищених текстових даних у числовий формат (вектори), оскільки математичні моделі та алгоритми машинного навчання можуть працювати виключно з числами.

Bag-of-Words (BoW, "Мішок слів") – класична та найбільш поширена модель. Текст представляється як невпорядкований набір слів, ігноруючи граматику та порядок слів. Для кожного документа створюється вектор, де кожна розмірність відповідає певному слову з загального словника (усіх унікальних слів корпусу), а значенням є частота появи цього слова в документі.

TF-IDF (Term Frequency-Inverse Document Frequency) – більш просунуте зважування ознак, що базується на BoW. Воно враховує не лише частоту слова в конкретному документі (TF), але й його "унікальність" у всьому корпусі (IDF). А саме, TF (Term Frequency): як часто слово зустрічається в документі, IDF (Inverse Document Frequency): показник, що зменшує вагу слів, які зустрічаються у багатьох документах (наприклад, загальнотематичні слова), і збільшує вагу рідкісних, специфічних слів, які краще характеризують тему документа. В результаті, кожному слову в документі присвоюється вага TF-IDF, яка відображає його семантичну важливість для даного документа в контексті всього корпусу.

Етап 4: Вибір, навчання та оцінка моделі

Після того, як текстові масиви перетворені на числову матрицю (документи-ознаки), в дію вступають інтелектуальні алгоритми Data Science.

Вибір моделі: для задачі класифікації текстів традиційно високу ефективність показують такі алгоритми як, Naive Bayes (Наївний Баєс), Support Vector Machines, логістична регресія, усі вони будуть детально проаналізовані в розділі 2.

Naive Bayes (Наївний Баєс)

Імовірнісний класифікатор, який, незважаючи на "наївне" припущення про незалежність ознак, чудово працює з текстовими даними (особливо Мультиноміальний Наївний Баєс).

Support Vector Machines (SVM, Метод Опорних Векторів)

Ефективний для високорозмірних просторів (якими є текстові дані після векторизації) і добре розділяє лінійно нероздільні класи.

Логістична регресія

Ще один потужний лінійний класифікатор, що добре себе зарекомендував.

Навчання моделі (Training): алгоритм "вивчає" закономірності на розміченій навчальній вибірці (пари "вектор TF-IDF" – "мітка класу").

Оцінка моделі (Evaluation): якість навченої моделі перевіряється на тестовій вибірці (даних, які модель не бачила). Для цього використовуються стандартні метрики Data Science: Accuracy (точність) – загальний відсоток правильних прогнозів.

- Precision (влучність) – частка об'єктів, які класифікатор назвав позитивними, і які справді є позитивними.

- Recall (повнота) – частка позитивних об'єктів, які класифікатор зміг правильно ідентифікувати.

- F1-Score – гармонійне середнє між Precision та Recall, особливо корисне при незбалансованих класах.

Варто зазначити, що описана вище класична схема Data Science (Preprocessing -> TF-IDF -> SVM/Naive Bayes) хоч і є ефективною, але має обмеження: вона ігнорує порядок слів та їхній глибинний семантичний зв'язок.

Сучасний Data Science все більше інтегрує методи глибокого навчання (Deep Learning), які дозволяють вирішувати ці проблеми. Замість TF-IDF використовуються векторні представлення слів (Word Embeddings), такі як Word2Vec або GloVe. Ці моделі представляють слова як щільні вектори у семантичному просторі, де близькі за значенням слова мають близькі вектори.

Ці семантичні вектори потім подаються на вхід нейронним мережам (наприклад, RNN, LSTM або Трансформерам (BERT)), які здатні враховувати контекст та порядок слів, досягаючи вищої якості класифікації. Ця синергія класичних принципів Data Science та потужних інтелектуальних алгоритмів глибокого навчання є ключовою для розробки передових методик автоматизованого семантичного аналізу.

Таким чином, класифікація текстових масивів є центральною задачею, що об'єднує лінгвістичні цілі NLP з методологічним апаратом Data Science для створення прикладних інтелектуальних систем.

1.4 Огляд актуальних наукових робіт в галузі інтелектуального аналізу текстових даних

Проблема автоматизованого семантичного аналізу та класифікації текстових масивів знаходиться в епіцентрі наукових досліджень у галузі обробки природної мови (NLP) та інтелектуальних систем вже кілька десятиліть. Актуальність цього напряму невпинно зростає, що зумовлено, з одного боку, збільшенням обсягів неструктурованих текстових даних в Інтернеті, корпоративних базах знань, наукових архівах та соціальних медіа, а з іншого – появою нових, значно потужніших інтелектуальних алгоритмів Data Science, зокрема методів глибокого навчання. Даний огляд має на меті систематизувати ключові наукові праці та

методологічні підходи, що сформували сучасний ландшафт цієї предметної галузі, виявити їхні переваги, недоліки та невирішені проблеми.

Класичні підходи машинного навчання та їхня роль

Ранні та фундаментальні роботи в галузі класифікації текстів значною мірою спиралися на класичні алгоритми машинного навчання в поєднанні з ретельним проектуванням ознак (feature engineering).

Дослідження таких авторів, як Ф. Себастьяні (F. Sebastiani), зокрема його фундаментальний огляд "Machine learning in automated text categorization", стали канонічними. Вони систематизували застосування таких методів, як Наївний Баєсівський класифікатор (Naive Bayes), який, незважаючи на своє "наївне" припущення про незалежність ознак, демонстрував високу ефективність та швидкість на задачах фільтрації спаму та тематичної рубрикації, що було показано в роботах Д.Льюїса (D. Lewis).

Іншим фундаментом є роботи Т. Йоахімса (T. Joachims), який одним з перших продемонстрував виняткову ефективність Методу опорних векторів (Support Vector Machines, SVM) для текстових даних. У своїх працях, наприклад "Text categorization with support vector machines", він показав, що SVM чудово справляються з високорозмірними та розрідженими просторами ознак, якими є матриці TF-IDF (Term Frequency-Inverse Document Frequency). Саме модель представлення тексту TF-IDF, яка зважує важливість слова на основі його частоти в документі та рідкісності в усьому корпусі, домінувала в наукових працях 1990-х та 2000-х років.

Ці класичні підходи (Naive Bayes, SVM, Логістична регресія) на базі TF-IDF або "мішка слів" (Bag-of-Words) досі є важливим "бейзлайном" (baseline) – базовим рівнем якості, з яким порівнюються всі нові, складніші методики. Їхня сила полягає у швидкості, відносній простоті реалізації та інтерпретабельності. Однак, їх фундаментальний недолік – повне ігнорування семантичного контексту, синонімії та порядку слів, що обмежує глибину семантичного аналізу.

Революція глибокого навчання: від статичних до контекстуальних представлень

Справжній прорив у семантичному аналізі пов'язаний з переходом до розподіленої семантики (distributional semantics) та нейромережевих архітектур.

Статичні векторні представлення (Word Embeddings)

Наукові праці Т. Міколова (T. Mikolov) та ін. у 2013 році, що представили алгоритми Word2Vec (Skip-gram та CBOW), стали переломним моментом. Вони запропонували методику навчання щільних векторних представлень слів (word embeddings) на гігантських нерозмічених текстових масивах. Ці вектори фіксували тонкі семантичні відношення, дозволяючи виконувати векторну арифметику (наприклад, "король" - "чоловік" + "жінка" ≈ "королева"). Майже одночасно, праця Дж. Пеннінгтона (J. Pennington), Р. Сочера (R. Socher) та К. Меннінга (C. Manning) у 2014 році представила модель GloVe (Global Vectors), яка комбінувала переваги методів матричної факторизації (як LSA) та локального контекстного вікна (як Word2Vec).

Ці статичні (тобто незмінні незалежно від контексту) представлення стали новою основою для текстових класифікаторів. Наукова робота Ю. Кіма (Y. Kim) "Convolutional Neural Networks for Sentence Classification" (2014) продемонструвала, як відносно прості згорткові нейронні мережі (CNN), застосовані поверх попередньо навчених векторів Word2Vec, можуть досягати найвищої якості (State-of-the-Art, SOTA) у задачах класифікації речень, ефективно виділяючи ключові N-грами. Водночас, для завдань, де важлива послідовність, актуальними стали рекурентні нейронні мережі (RNN), зокрема їхні просунуті варіанти LSTM (Long Short-Term Memory) (дослідження С. Хохрайтера (S.Hochreiter) та Ю. Шмідхубера (J.Schmidhuber)) та GRU (Gated Recurrent Unit). Ці архітектури, як показано в багатьох дослідженнях, здатні утримувати інформацію про довгострокові залежності в тексті, що є критичним для глибокого семантичного аналізу.

Ера Трансформерів та контекстуальних моделей

Фундаментальною проблемою статичних представлень була полісемія (багатозначність): слово "ключ" мало однаковий вектор, незалежно від того, чи йшлося про музичний ключ, чи про дверний.

Цю проблему вирішила революційна наукова стаття А. Васвані (A.Vaswani) та ін. "Attention Is All You Need" (2017), яка представила архітектуру Трансформер (Transformer). Вона повністю відмовилася від рекурентних шарів на користь механізму само-уваги (self-attention), що дозволило моделі зважувати важливість кожного слова у реченні відносно всіх інших слів одночасно.

На основі цієї архітектури з'явилася плеяда контекстуальних мовних моделей, які домінують у сучасних дослідженнях.

BERT (Bidirectional Encoder Representations from Transformers) – робота Дж.Девліна (J. Devlin) та ін. є, мабуть, найвпливовішою. BERT запропонував парадигму попереднього навчання та тонкого налаштування (pre-training and fine-tuning). Модель навчається на гігантських обсягах тексту (як-от вся Вікіпедія) на двох задачах: Masked Language Model (MLM), де вона прогнозує приховані слова, та Next Sentence Prediction (NSP). Завдяки двонаправленому (bidirectional) енкодеру, BERT "розуміє" контекст слова, дивлячись одночасно на слова зліва та справа. Для класифікації текстів, дослідники просто "тонко налаштовують" (fine-tune) вже навченого BERT на своєму конкретному наборі розмічених даних, досягаючи безпрецедентної якості.

Подальший розвиток (RoBERTa, ALBERT, DistilBERT)

Численні наступні роботи були спрямовані на вдосконалення BERT. RoBERTa (Y. Liu et al., 2019) продемонструвала, що оригінальний BERT був "недонавчений", і шляхом оптимізації гіперпараметрів навчання (більше даних, динамічне маскування, видалення NSP) можна досягти значно кращих результатів. ALBERT (Z. Lan et al., 2019) зосередився на зменшенні кількості параметрів для створення легших моделей. DistilBERT (V. Sanh et al., 2019) успішно застосував дистиляцію знань (knowledge distillation) для створення менших, швидших версій BERT, що є критичним для практичного впровадження.

Багатомовні та специфічні моделі

Дослідження також зосередились на багатомовності. Моделі, як-от XLM-RoBERTa (A. Conneau et al., 2019), навчені на текстах зі 100 мов, показали високу здатність до "міжмовного перенесення" знань, дозволяючи будувати класифікатори для мов з обмеженими ресурсами (low-resource languages).

Виявлені проблеми та прогалини в дослідженнях

Незважаючи на успіхи трансформерних архітектур, огляд актуальних наукових робіт дозволяє виділити низку невирішених проблем.

Обчислювальна вартість та ефективність. Сучасні SOTA-моделі (як BERT-Large, RoBERTa) містять сотні мільйонів або мільярди параметрів. Їхнє навчання та навіть використання (inference) вимагає значних обчислювальних ресурсів (потужних GPU/TPU), що є недоступним для багатьох дослідників та компаній. Існує гострий запит на розробку ефективних методик, які б досягали порівнянної якості при менших затратах.

Проблема доменної адаптації. Моделі, попередньо навчені на загальних текстах (новини, Вікіпедія), часто показують суттєве падіння якості при застосуванні до вузькоспеціалізованих доменів (наприклад, юридичні документи, медичні записи, технічна документація), де термінологія та семантичні зв'язки відрізняються. Розробка методик для ефективної адаптації (domain adaptation) моделей є актуальною задачею.

Нестача інтерпретабельності ("чорна скринька"). Глибокі нейронні мережі є "чорними скриньками". Вони видають результат класифікації, але не пояснюють чому було прийняте саме таке рішення. Це є неприйнятним для критичних галузей (медицина, фінанси). Актуальними є дослідження в галузі XAI (Explainable AI) для NLP, спрямовані на розробку методик, що підвищують прозорість моделей.

Гібридні підходи. Більшість робіт фокусується або на класичних методах (SVM+TF-IDF), або на глибокому навчанні (BERT). Існує дослідницька прогалина у розробці гібридних методик, які б інтелектуально поєднували переваги обох підходів: наприклад, використання швидких класичних методів для "простих"

випадків та потужних нейромереж для "складних", або використання семантичних ознак з BERT у поєднанні з класичними класифікаторами.

Сучасний стан галузі характеризується домінуванням великих, попередньо навчених трансформерних моделей, які демонструють високу точність семантичного аналізу. Однак, їхня висока вартість, складність адаптації та низька інтерпретабельність формують чіткий запит. Таким чином, розробка комплексної методики автоматизованого семантичного аналізу та класифікації, яка б спиралася на інтелектуальні алгоритми Data Science, але при цьому була б ефективною, адаптивною та, за можливості, інтерпретованою, є надзвичайно актуальною та науково значущою задачею.

2 АНАЛІЗ МЕТОДІВ СЕМАНТИЧНОГО АНАЛІЗУ ТА КЛАСИФІКАЦІЇ ТЕКСТІВ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНИХ АЛГОРИТМІВ

2.1 Аналіз існуючих методів семантичного аналізу та класифікації

Методи семантичного аналізу та класифікації – це процес аналізу тексту для визначення його смислу та інтерпретації значення слів і фраз у контексті. Семантичний аналіз може включати лексичний, синтаксичний та контекстуальний аналіз, а класифікація текстів – це одне із застосувань семантичного аналізу, що дозволяє розподіляти тексти за категоріями.

Класичні методи машинного навчання для класифікації текстів (Naive Bayes, SVM тощо)

Історично першим та найбільш фундаментальним етапом у розвитку систем автоматизованої обробки природної мови (NLP) став підхід, що базується на використанні класичних алгоритмів машинного навчання (Classical Machine Learning). Цей клас методів, що домінував у наукових дослідженнях та промислових розробках з кінця 1990-х до початку 2010-х років, спирається на статистичну теорію навчання та використання поверхневих лексичних ознак тексту.

Головною особливістю класичного підходу є чітке розмежування етапу конструювання ознак (Feature Engineering) та етапу безпосередньо класифікації. Текстові дані, будучи за своєю природою неструктурованими послідовностями символів, не можуть бути безпосередньо оброблені математичними алгоритмами. Тому класичні методи нерозривно пов'язані з моделями векторного представлення, такими як «Мішок слів» (Bag-of-Words) та TF-IDF, які перетворюють документ у числовий вектор у багатовимірному просторі.

Нижче наведено детальний аналіз найбільш ефективних та поширених класичних алгоритмів, що застосовуються для задач категоризації текстових масивів.

Наївний Баєсівський класифікатор (Naive Bayes Classifier)

Наївний Баєсівський класифікатор є одним із найстаріших, але водночас напрочуд ефективних імовірнісних методів класифікації текстів. Його теоретичною основою є теорема Баєса, яка описує взаємозв'язок ймовірностей подій.

Основна ідея методу полягає у визначенні апостеріорної ймовірності належності документа d до класу c ($P(c|d)$) на основі апріорної ймовірності класу ($P(c)$) та ймовірності появи документа в цьому класі ($P(d|c)$). Згідно з теоремою Баєса:

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}, \tag{2.1}$$

де $\boldsymbol{P(c|d)}$ – апостеріорна ймовірність належності документа $\boldsymbol{d}$ до класу $\boldsymbol{c}$; $\boldsymbol{P(d|c)}$ – ймовірність появи документа в цьому класі; $\boldsymbol{P(c)}$ – апріорна ймовірність класу; $\boldsymbol{P(d)}$ – ймовірність документа (константа).

Оскільки знаменник $P(d)$ є константою для даного документа і не залежить від класу, задача класифікації зводиться до знаходження класу, що максимізує чисельник (Maximum A Posteriori estimation – MAP):

$$\hat{c} = arg \max_{c \in C} P(c) \prod_{i=1}^{n} P(w_i|c). \tag{2.2}$$

Термін «наївний» походить від сильного (і часто хибного з лінгвістичної точки зору) припущення про умовну незалежність ознак. Алгоритм вважає, що поява будь-якого слова w_i в документі ніяк не залежить від появи інших слів, за умови, що клас документа відомий. Незважаючи на те, що в реальній мові слова тісно пов'язані (контекст, стійкі словосполучення), на практиці це спрощення дозволяє суттєво зменшити обчислювальну складність та кількість параметрів

моделі, демонструючи високу точність у задачах тематичної класифікації та фільтрації спаму (див. рис. 2.1),

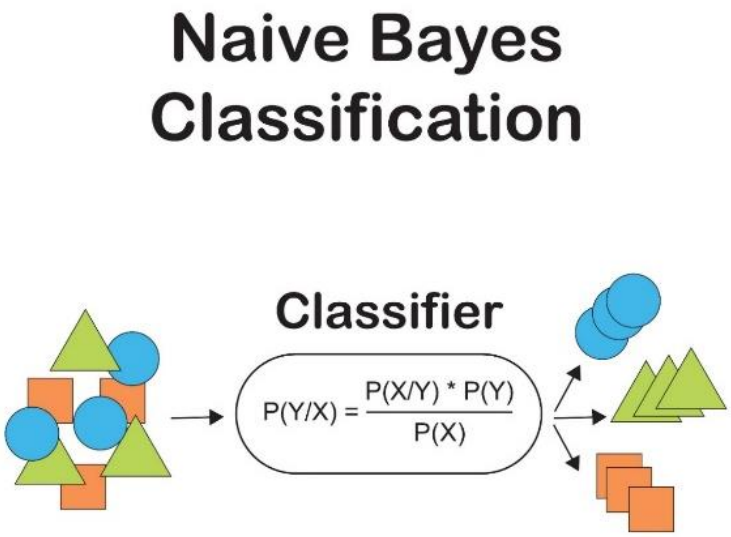


Рис. 2.1 Наївний Баєсівський класифікатор

Девід Льюїс у своєму аналізі зазначає парадокс цього методу: незважаючи на те, що припущення про незалежність слів є лінгвістично хибним, Наївний Баєс часто демонструє результати, співмірні зі складнішими методами, оскільки для задач класифікації важлива не точна оцінка ймовірності, а лише правильне ранжування класів [44].

У задачах NLP використовуються специфічні модифікації методу:

- Multinomial Naive Bayes (Мультиноміальний НБ) враховує кількість входжень (частоту) кожного слова в документі. Це найбільш популярний варіант для класифікації довгих текстів.

- Bernoulli Naive Bayes (Бернуллі НБ) оперує бінарними ознаками (слово присутнє/відсутнє), ігноруючи частоту. Ефективний для коротких текстів.

До ключових переваг методу належать висока швидкість навчання та передбачення, низькі вимоги до обсягу оперативної пам'яті та стійкість до шумів у даних. Він також добре працює на малих наборах даних. Головним недоліком є нездатність враховувати контекст та порядок слів через припущення незалежності.

Метод опорних векторів (Support Vector Machines – SVM)

Метод опорних векторів, розроблений В. Вапніком, вважається одним із найпотужніших алгоритмів серед класичних методів навчання з учителем. На відміну від імовірнісного підходу Баєса, SVM є геометричним алгоритмом.

Основна ідея SVM полягає у пошуку оптимальної розділяючої гіперплощини у N – вимірному просторі ознак (де N – розмір словника), яка б найкращим чином розділяла вектори документів різних класів. Це означає максимізацію зазору (margin) – відстані між гіперплощиною та найближчими до неї точками кожного класу (ці точки і називаються опорними векторами) (див. рис. 2.2).

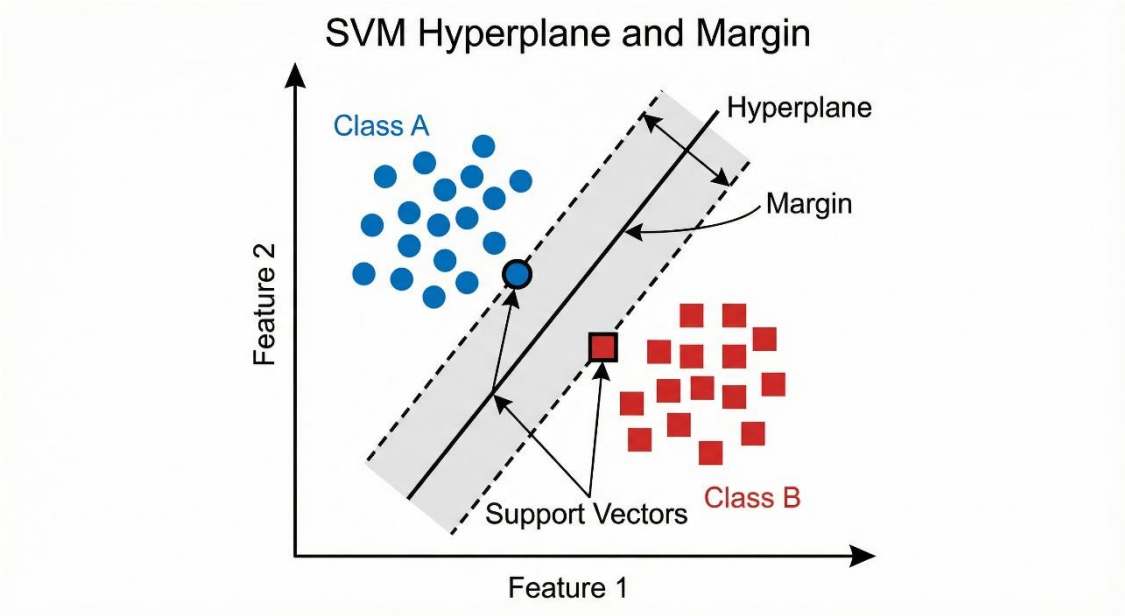


Рис. 2.2 Принцип розділення класів у методі опорних векторів (SVM).

Математично задача зводиться до мінімізації норми вагового вектора $||w||$ при виконанні умов розділення класів:

$$min_{w,b}\frac{1}{2}||w||^2, \tag{2.3}$$

$$\text{за умови } y_i(w \cdot x_i + b) \geq 1, \forall i. \tag{2.4}$$

При застосування в NLP текстові дані при використанні TF-IDF векторизації характеризуються двома специфічними властивостями: висока розмірність (High Dimensionality) – кількість ознак (слів) може сягати десятків або сотень тисяч; розрідженість (Sparsity) – більшість елементів векторів дорівнюють нулю.

SVM ідеально підходить для таких умов. Дослідження (зокрема, роботи Т.Йоахімса) показали, що текстові дані часто є лінійно роздільними у просторах високої розмірності. Тому найчастіше використовується SVM з лінійним ядром (Linear Kernel), який є обчислювально ефективним і дає результати, що часто перевершують складніші нелінійні моделі.

SVM забезпечує високу точність класифікації та має чудову узагальнюючу здатність (низький ризик перенавчання) завдяки принципу максимізації зазору. Головним недоліком є висока обчислювальна складність на етапі навчання ($O(N^2)$ або $O(N^3)$), що робить його повільним на дуже великих наборах даних (сотні тисяч документів). Також модель SVM не надає прямих ймовірнісних оцінок (тільки відстань до гіперплощини), що ускладнює інтерпретацію впевненості класифікатора.

Логістична регресія (Logistic Regression)

Попри назву, логістична регресія є алгоритмом лінійної класифікації, а не регресії. Це дискримінативна модель, яка, на відміну від генеративного Наївного Баєса, моделює безпосередньо ймовірність класу $P(c|d)$.

Алгоритм обчислює зважену суму вхідних ознак (слів) і пропускає результат через нелінійну сигмоїдну функцію (або Softmax для багатокласової класифікації), яка перетворює будь-яке дійсне число у значення ймовірності в діапазоні [0, 1]:

$$P(y = 1|x) = \sigma(w \cdot x + b) = \frac{1}{1} + e^{-(w \cdot x + b)}, \quad (2.5)$$

де $\boldsymbol{\sigma}$ – сигмоїдна функція; $\boldsymbol{w}$ – ваги моделі; $\boldsymbol{b}$ – зсув (bias).

Логістична регресія є «золотим стандартом» та базовим рівнем (baseline) для більшості задач NLP. Її головна цінність полягає в інтерпретованості. Вагові коефіцієнти w, отримані після навчання, прямо вказують на важливість кожного

слова для прийняття рішення. Наприклад, велика позитивна вага для слова «чудовий» у задачі аналізу тональності означає сильний вплив на клас «позитивний відгук».

Ансамблеві методи (Random Forest, Gradient Boosting)

Окрему групу класичних методів складають алгоритми, що базуються на деревах рішень. Одиночне дерево рішень рідко використовується для тексту через схильність до перенавчання на розріджених даних. Проте ансамблі дерев – випадковий ліс (Random Forest) та градієнтний бустинг (XGBoost, LightGBM) – демонструють високу ефективність.

Ці методи будують велику кількість слабких класифікаторів (дерев) і об'єднують їхні прогнози. Хоча вони менш популярні для роботи з «мішком слів» через надмірну розмірність, вони є незамінними, коли текстові ознаки комбінуються з іншими метаданими (наприклад, довжина тексту, час публікації, автор).

Узагальнюючи аналіз класичних методів, можна виділити їх спільну рису: вони розглядають текст як набір ізольованих ознак (слів або N-грам). Як переваги, можна визначити, високу швидкість роботи, інтерпретованість результатів (можливість пояснити рішення на основі ваги слів), ефективність на малих та середніх обсягах даних. Але, всі описані методи ігнорують семантичну близькість слів (синонімію) та порядок слів (контекст). Слова «добрий» та «хороший» для цих алгоритмів є такими ж різними, як «добрий» та «зелений».

Саме це обмеження зумовило необхідність переходу до більш складних методів на основі нейронних мереж та векторних представлень слів (Embeddings), які будуть розглянуті в наступних підрозділах.

Методи семантичного аналізу на основі глибокого навчання (RNN, LSTM, Трансформери)

Поява та стрімкий розвиток методів глибокого навчання (Deep Learning) у другому десятилітті XXI століття ознаменували собою фундаментальний зсув парадигми в галузі обробки природної мови (NLP). Якщо класичні методи спиралися на інженерію ознак вручну та статистичний аналіз частот слів («мішок

слів»), то методи глибокого навчання дозволили перейти до автоматичного виявлення ієрархічних ознак та моделювання складних нелінійних залежностей у тексті.

Ключовою перевагою цього підходу є відмова від трактування тексту як невпорядкованої множини слів. Натомість текст розглядається як послідовність (sequence), де порядок елементів, їхнє взаємне розташування та дистанція між ними несуть критично важливу семантичну інформацію. Для обробки таких послідовних даних було розроблено спеціалізовані архітектури нейронних мереж.

Рекурентні нейронні мережі (Recurrent Neural Networks – RNN)

Рекурентні нейронні мережі стали першою архітектурою, спеціально адаптованою для роботи з послідовностями довільної довжини.

На відміну від звичайних мереж прямого поширення (Feed-Forward Neural Networks), де інформація рухається тільки вперед, RNN мають внутрішній цикл, що дозволяє інформації зберігатися. Ключовим елементом RNN є прихований стан (hidden state) h_t, який діє як «пам'ять» мережі.

Процес обробки тексту відбувається крок за кроком (слово за словом):

1. На кожному часовому кроці t мережа отримує вхідний вектор поточного слова x_t та прихований стан з попереднього кроку h_{t-1}.

2. Мережа оновлює свій прихований стан за формулою:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h), \quad (2.6)$$

де $\boldsymbol{h_t}$ – прихований стан на кроці $\boldsymbol{t}$; $\boldsymbol{W_{hh}, W_{xh}}$ – вагові матриці; $\boldsymbol{b_h}$ – вектор зсуву; $\boldsymbol{tanh}$ – функція активації.

3. Цей новий стан h_t містить інформацію як про поточне слово, так і (теоретично) про весь попередній контекст.

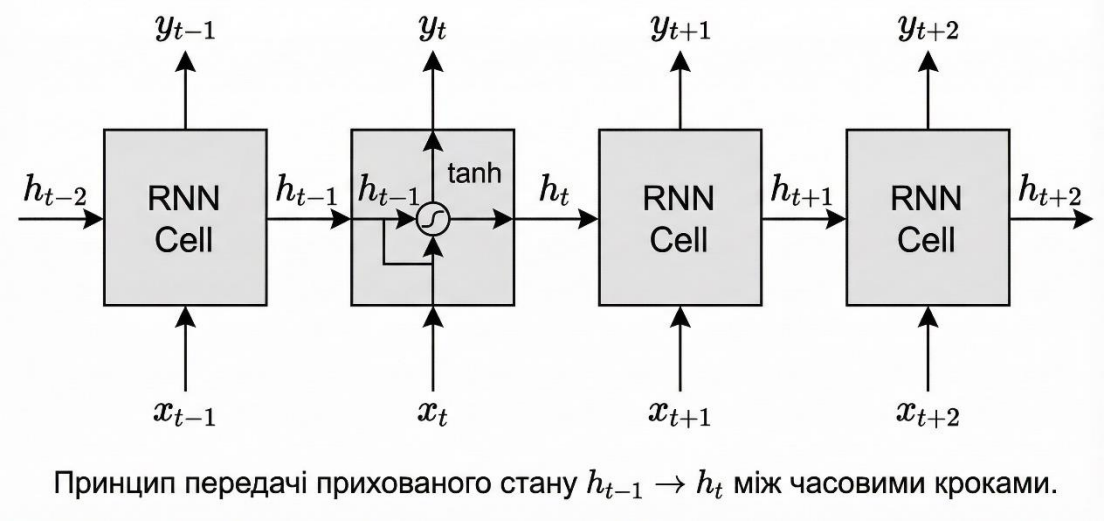


Рис. 2.3 Принцип роботи рекурентної нейронної мережі

RNN вперше дозволили враховувати контекст: мережа могла зрозуміти, що слово «замок» означає пристрій, якщо перед цим було слово «двері». Проте класичні RNN (Vanilla RNN) виявилися неефективними для довгих текстів через проблему зникаючого градієнта (Vanishing Gradient Problem).

Під час зворотного поширення помилки (Backpropagation through time) градієнти, що проходять через багато часових кроків, експоненціально зменшуються (якщо ваги < 1) або вибухають. У результаті мережа «забуває» інформацію, яка була на початку довгого речення або абзацу. Це унеможливлювало якісний семантичний аналіз складних текстів.

Як детально описано в статті Хохрайтера та Шмідхубера «Long Short-Term Memory», архітектура LSTM вводить поняття «комірки пам'яті» з механізмом воріт (gates), що створює «карусель постійної помилки» (Constant Error Carousel). Це дозволяє градієнту протікати через тисячі часових кроків без затухання, вирішуючи проблему довгострокових залежностей [36].

Мережі довгої короткострокової пам'яті (Long Short-Term Memory – LSTM)

Для вирішення проблеми зникаючого градієнта у 1997 році Зеппом Хохрайтером та Юргеном Шмідхубером була запропонована архітектура LSTM.

Вона стала «золотим стандартом» в NLP до появи Трансформерів (приблизно до 2017 року).

LSTM – це особливий вид RNN, здатний вивчати довготривалі залежності. Замість простого шару з $tanh$, комірка LSTM (LSTM cell) має складну внутрішню структуру, що складається з чотирьох взаємодіючих шарів та механізму воріт (gates). (див. рис. 2.4)

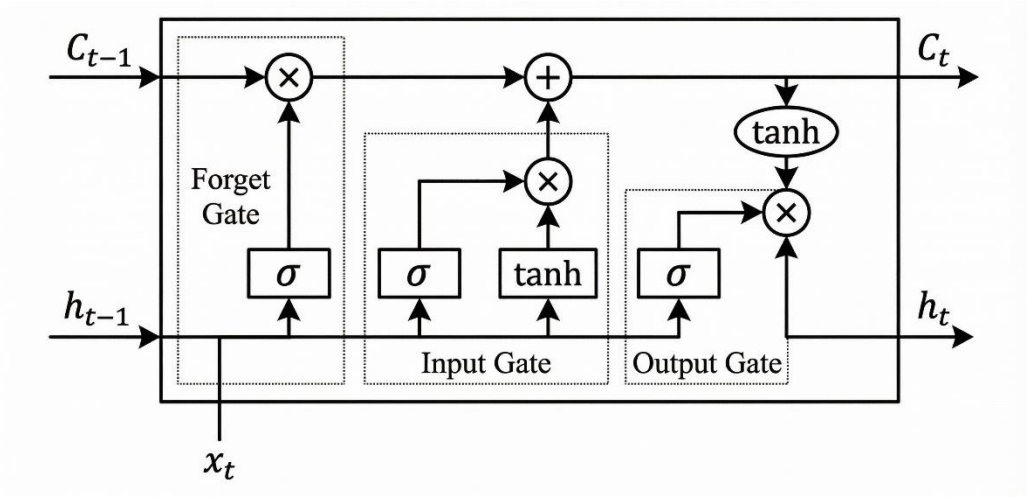


Рис. 2.4 Внутрішня структура комірки LSTM.

Ключовою інновацією є C_t (Cell State) – своєрідна «інформаційна магістраль», що проходить крізь весь ланцюжок з мінімальними лінійними взаємодіями, дозволяючи інформації протікати без змін на великі відстані.

LSTM регулює потік інформації за допомогою трьох типів воріт:

1. Ворота забування (Forget Gate): вирішують, яку інформацію з попереднього стану комірки C_{t-1} слід видалити (забути). Це сигмоїдний шар, що видає числа від 0 (забути все) до 1 (зберегти все).

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big), \tag{2.7}$$

де $\boldsymbol{f_t}$ – вектор воріт забування; σ – сигмоїдна функція.

2. Вхідні ворота (Input Gate): вирішують, яку нову інформацію з поточного входу x_t слід записати в стан комірки.

3. Вихідні ворота (Output Gate): визначають, що буде на виході комірки (прихований стан h_t) на основі оновленого стану комірки.

Завдяки цій архітектурі LSTM здатні «пам'ятати», що на початку абзацу йшлося про чоловіка, і через 50 слів правильно інтерпретувати займенник «він» (розрішення анафори). Для покращення якості часто використовують двонаправлені LSTM (Bi-LSTM), які читають текст одночасно зліва направо і справа наліво, формуючи повний контекст для кожного слова.

Архітектура Трансформер (Transformer) та механізм уваги

Справжня революція в методах семантичного аналізу відбулася у 2017 році з публікацією статті «Attention Is All You Need» дослідниками Google Brain. Вони запропонували архітектуру Трансформер, яка повністю відмовилася від рекурентності (RNN/LSTM) на користь механізму уваги (Attention).

У статті «Attention Is All You Need» (2017) Васвані та співавтори стверджують: «Ми пропонуємо нову просту архітектуру мережі, Transformer, засновану виключно на механізмах уваги, повністю відмовляючись від рекурентності та згорток» [62]. Головною перевагою є можливість повної паралелізації навчання та здатність моделювати глобальні залежності в реченні за постійну кількість операцій $O(1)$.

Головним недоліком LSTM була їхня послідовна природа. Щоб обробити 100-те слово, мережа мусила спершу обробити 99 попередніх. Це унеможливлювало паралелізацію обчислень на GPU, робило навчання повільним і обмежувало роботу з дуже довгими контекстами.

Трансформер обробляє всі слова в реченні одночасно (паралельно). Механізм Self-Attention дозволяє кожному слову в реченні «дивитися» на всі інші слова, щоб зрозуміти, наскільки вони важливі для його інтерпретації (див. рис. 2.5).

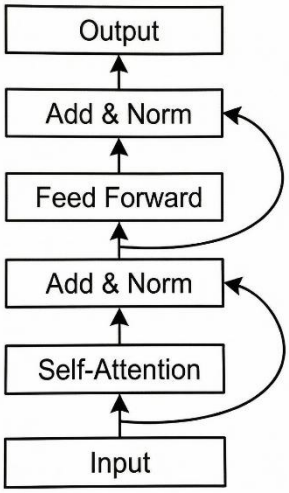


Рис. 2.5 Базовий блок архітектури Transformer.

Математично це реалізується через три вектори для кожного слова: запит (Query - Q), ключ (Key - K) та значення (Value - V). Вага уваги (Attention Score) обчислюється як скалярний добуток запиту одного слова на ключі всіх інших:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (2.8)$$

де $\boldsymbol{Q}$ – матриця запитів (Query); $\boldsymbol{K}$ – матриця ключів (Key); $\boldsymbol{V}$ – матриця значень (Value); $\boldsymbol{d_k}$ – розмірність векторів ключів.

Це дозволяє моделі динамічно фокусуватися на релевантних частинах речення. Наприклад, у фразі «Я поклав гроші в банк», слово «банк» матиме сильний зв'язок зі словом «гроші» та «поклав», що дозволить однозначно визначити його семантику як фінансової установи.

Трансформери використовують кілька паралельних механізмів уваги («голів»). Це дозволяє моделі одночасно відстежувати різні типи взаємозв'язків: одна «голова» може фокусуватися на граматичних зв'язках (хто зробив?), інша – на семантичних (яка тональність?), третя – на кореферентних (хто такий «він»?).

Еволюція методів глибокого навчання для семантичного аналізу пройшла шлях від спроб запам'ятати контекст через цикли (RNN) до складних механізмів керування пам'яттю (LSTM) і, нарешті, до повної відмови від послідовної обробки

на користь механізму глобальної уваги (Transformers). Саме архітектура Трансформера стала базою для створення сучасних великих мовних моделей (BERT, GPT), які демонструють безпрецедентну якість класифікації та розуміння текстів, що робить їх пріоритетним вибором для розробки методики.

Підходи до векторного представлення текстів (Word2Vec, GloVe, BERT embeddings)

Фундаментальною проблемою застосування будь-яких математичних алгоритмів, зокрема методів машинного та глибокого навчання, до задач обробки природної мови (NLP) є неможливість безпосередньої роботи обчислювальних систем із символьними даними. Комп'ютери та математичні моделі не можуть оперувати безпосередньо з символами та словами, вони вимагають чисельного представлення даних. Комп'ютерні системи оперують числами, векторами та матрицями. Протягом десятиліть ця задача вирішувалася за допомогою простих, але ефективних статистичних методів, таких як "Мішок слів" (Bag-of-Words, BoW) або TF-IDF (Term Frequency-Inverse Document Frequency). Тому критично важливим етапом побудови будь-якої системи семантичного аналізу є процес векторизації (Vectorization) або вбудовування (Embedding) – перетворення дискретних текстових одиниць (слів, речень) у неперервні числові вектори у багатовимірному просторі.

Історично перші підходи, такі як One-Hot Encoding (унітарне кодування) та TF-IDF, створювали розріджені вектори (sparse vectors). Їхня розмірність дорівнювала розміру словника (що може сягати 100 000+ слів), а 99% значень були нулями. Головним недоліком таких представлень була їхня семантична порожнеча: вектори були ортогональними, тобто математична відстань між словами «автомобіль» та «машина» була такою ж, як між словами «автомобіль» та «банан».

Сучасна парадигма NLP базується на використанні щільних векторних представлень (dense embeddings), які базуються на дистрибутивній гіпотезі (Distributional Hypothesis), сформульованій лінгвістом Дж. Р. Фертом: «Слово характеризується компанією, яку воно складає» (You shall know a word by the

company it keeps). Це означає, що слова, які часто зустрічаються в схожих контекстах, мають мати схожі (близькі) векторні представлення.

Нижче наведено детальний аналіз трьох еволюційних етапів розвитку технологій векторного представлення: Word2Vec, GloVe та BERT.

Word2Vec: революція прогнозуючих моделей

У 2013 році дослідники Google під керівництвом Томаша Міколова представили модель Word2Vec, яка здійснила революцію в NLP. Відмовившись від підрахунку статистики (як у LSA), Word2Vec запропонував використовувати прості нейронні мережі для навчання векторів.

Word2Vec – це не одна модель, а сімейство з двох архітектур, що навчаються вирішувати обернені задачі.

CBOW (Continuous Bag-of-Words): модель намагається передбачити цільове слово на основі його контексту (слів ліворуч і праворуч). Наприклад, маючи вхід «кіт ... на килимку», мережа вчиться прогнозувати слово «сидить». Вхідні вектори контекстних слів усереднюються (звідси назва «Bag-of-Words»), що робить модель швидкою та ефективною для частотних слів, але дещо згладжує порядок слів.

Skip-gram: ця архітектура діє навпаки, отримуючи на вхід цільове слово, вона намагається передбачити контекст (сусідні слова) у певному вікні. Наприклад, для слова «сидить» вона максимізує ймовірність появи слів «кіт», «на», «килимку». Skip-gram є обчислювально важчим, проте він набагато краще працює з рідкісними словами та меншими наборами даних, оскільки генерує більше тренувальних пар з одного речення.

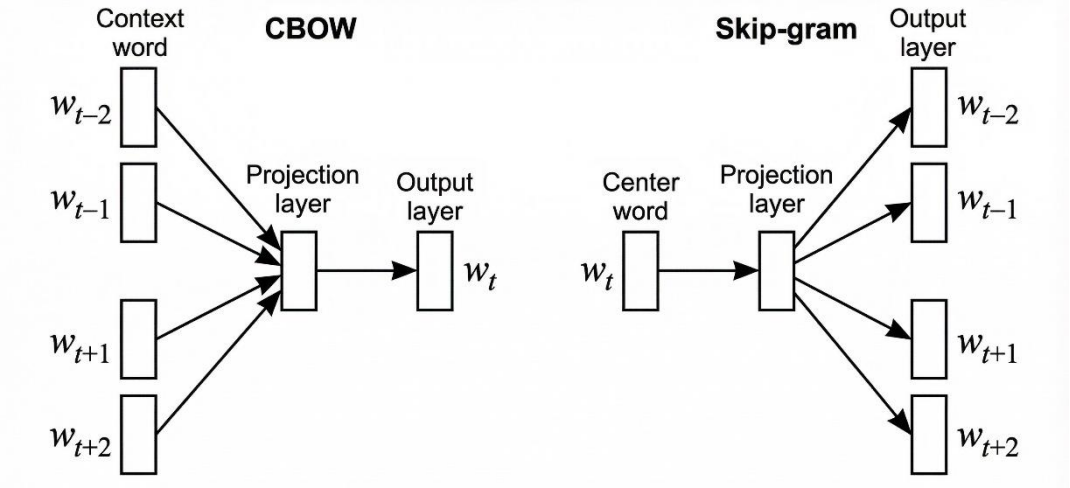


Рис. 2.6 Архітектури навчання моделі Word2Vec.

В результаті навчання на мільярдних корпусах текстів, ваги прихованого шару нейронної мережі стають власне векторними представленнями слів. Утворений векторний простір має дивовижні лінійні властивості. Семантичні відношення перетворюються на геометричні зміщення.

Класичний приклад (див. рис. 2.7):

$$Vector(\text{Король}) - Vector(\text{Чоловік}) + Vector(\text{Жінка}) \approx Vector(\text{Королева})$$

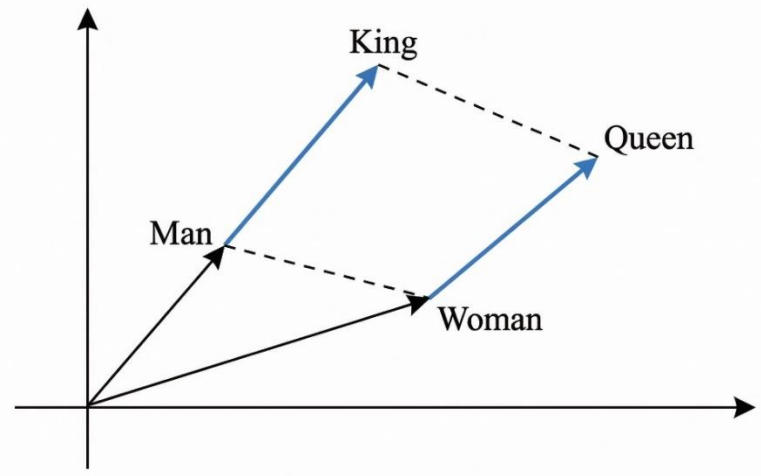


Рис. 2.7 Візуалізація семантичних відношень у векторному просторі.

Це свідчить про те, що модель автоматично, без учителя, вивчила концепцію «гендеру» як напрямок у векторному просторі.

Ключовим відкриттям Томаша Міколова стала здатність отриманих векторів фіксувати семантичні та синтаксичні аналогії через лінійні алгебраїчні операції, що дозволило моделям виходити за межі простого співставлення символів [47].

Головним обмеженням Word2Vec є те, що це статичне представлення, оскільки модель усереднює всі контексти, в яких вона зустрічала це слово.

GloVe (Global Vectors): гібрид статистики та навчання

У 2014 році дослідники зі Стенфордського університету (Пеннінгтон, Сочер, Меннінг) запропонували модель GloVe, яка мала на меті об'єднати переваги методів матричної факторизації (глобальна статистика) та методів локального вікна (як Word2Vec).

Автори GloVe зауважили, що Word2Vec, навчаючись на локальних вікнах, нераціонально використовує глобальну статистику всього корпусу. GloVe починає роботу з побудови гігантської матриці спів-зустрічальності (Co-occurrence Matrix) X, де елемент X_{ij} показує, скільки разів слово j зустрілося в контексті слова i у всьому корпусі текстів.

Замість нейронної мережі, GloVe використовує метод зваженої найменших квадратів. Функція втрат (Loss function) формулюється так, щоб скалярний добуток векторів двох слів w_i та w_j був максимально близьким до логарифма ймовірності їхньої спільної появи:

$$w_i^T w_j + b_i + b_j = \log(X_{ij}), \tag{2.9}$$

де $\boldsymbol{X_{ij}}$ – кількість разів, коли слово $\boldsymbol{j}$ зустрічається в контексті слова $\boldsymbol{i}$.

Це означає, що семантична близькість безпосередньо пов'язується з імовірністю слів з'являтися разом.

У порівнянні з Word2Vec модель GloVe часто демонструє швидше навчання та кращу стабільність на менших корпусах, оскільки базується на явній статистиці. Отримані вектори також мають властивості лінійної субструктури (дозволяють аналогії). Втім, як і Word2Vec, GloVe створює статичні ембедінги: кожному слову зі словника відповідає один незмінний вектор.

Джеффрі Пеннінгтон (Jeffrey Pennington) у своїй роботі критикує Word2Vec за нераціональне використання глобальної статистики корпусу. Запропонована ним модель GloVe комбінує переваги глобальної матричної факторизації та локального контекстного вікна, формулюючи функцію втрат так, щоб скалярний добуток векторів дорівнював логарифму ймовірності їхньої спільної появи [52].

BERT Embeddings: ера контекстуалізованих представлень

Головним і фундаментальним недоліком Word2Vec та GloVe є проблема полісемії (багатозначності). У цих моделях слово «коса» (знаряддя праці) та «коса» (зачіска) матимуть один і той самий вектор, який є середнім арифметичним усіх значень. Це суттєво обмежує точність семантичного аналізу складних текстів.

Поява моделі BERT (Bidirectional Encoder Representations from Transformers) у 2018 році від Google AI змінила парадигму векторизації від статичної до динамічної (контекстуалізованої).

BERT не має фіксованого вектора для слова. Замість цього, він генерує векторне представлення слова "на льоту", пропускаючи все речення через глибоку нейронну мережу (Трансформер).

Вектор слова «ключ» у реченні «скрипковий ключ» буде кардинально відрізнятися від вектора слова «ключ» у реченні «гаєчний ключ». Модель аналізує оточення слова (контекст) за допомогою механізму Self-Attention (само-уваги), визначаючи, які інші слова в реченні уточнюють значення поточного.

BERT базується на архітектурі Encoder від Transformer. Для отримання векторного представлення тексту використовують:

Token Embeddings – вхідні слова розбиваються на частини (subwords) за допомогою алгоритму WordPiece, що вирішує проблему невідомих слів (OOV – Out Of Vocabulary).

Segment & Position Embeddings – додаються вектори, що кодують позицію слова в реченні (оскільки Трансформер не має рекурентності і не знає порядку слів без цього).

Layer Outputs – після проходження через 12 (BERT-Base) або 24 (BERT-Large) шари уваги, ми отримуємо вектори на виході. Дослідження показують, що

для задач класифікації часто використовують вектор спеціального токена [CLS], який додається на початок кожного речення і акумулює в собі семантику всього тексту. Для задач NER або пошуку відповідей використовують вектори окремих токенів з останніх 4-х шарів (шляхом конкатенації або усереднення).

Ембедінги BERT є глибоко двонаправленими (bidirectional). Модель «бачить» весь контекст одночасно (і ліворуч, і праворуч від слова) завдяки тренувальній задачі Masked Language Model (MLM), де вона вчиться відновлювати приховані слова в реченні. Це дозволяє вловлювати найтонші нюанси смислу, іронію, заперечення та синтаксичні зв'язки, недоступні для Word2Vec або GloVe.

Еволюція методів векторного представлення пройшла шлях від простих статистичних моделей до складних контекстуалізованих архітектур.

Word2Vec та GloVe забезпечили прорив у розумінні семантичної близькості слів та векторній арифметиці, ставши стандартом для багатьох задач, де критична швидкість та простота.

BERT та його наступники (RoBERTa, DistilBERT) вирішили проблему багатозначності та залежності від контексту, забезпечивши найвищу точність (State-of-the-Art) у задачах семантичного аналізу та класифікації.

2.2 Аналіз переваг та недоліків існуючих інтелектуальних алгоритмів Data Science для обробки текстових масивів

Проведений у попередніх підрозділах огляд методів дозволяє стверджувати, що сучасний арсенал Data Science пропонує широкий спектр інструментів для семантичного аналізу та класифікації: від класичних статистичних моделей до передових нейромережевих архітектур. Проте, жоден з цих підходів не є універсальним. Вибір конкретного алгоритму завжди є компромісом між точністю, швидкістю, вимогами до обчислювальних ресурсів та потребою в інтерпретованості результатів.

Для системного аналізу доцільно розділити існуючі алгоритми на три концептуальні групи та проаналізувати їхні сильні та слабкі сторони.

Класичні статистичні підходи (TF-IDF + SVM / Naive Bayes)

Ця група методів базується на припущенні «Мішка слів» (Bag-of-Words), де текст представляється як вектор частот термінів, а класифікація виконується лінійними або імовірнісними моделями. До їх переваг можна віднести: високу обчислювальну ефективність та швидкість, інтерпретованість результатів («Біла скринька»), ефективність на малих наборах даних, семантична обмеженість («Semantic Gap»), ігнорування контексту та порядку слів, проблема розрідженості даних (Sparsity).

Алгоритми цієї групи (особливо Naive Bayes) мають лінійну обчислювальну складність $O(N)$, що дозволяє навчати моделі та виконувати класифікацію на звичайних центральних процесорах (CPU) за лічені секунди навіть на великих корпусах. Це робить їх безальтернативними для систем реального часу з жорсткими обмеженнями затримки (low latency).

На відміну від глибоких нейромереж, рішення класичних моделей легко пояснити. Аналізуючи вагові коефіцієнти моделі (наприклад, у логістичній регресії або лінійному SVM), можна чітко визначити, які саме слова зробили найбільший внесок у віднесення документа до певного класу. Це критично важливо для сфер, де вимагається прозорість прийняття рішень (право, медицина, банківський сектор).

Класичні моделі мають низьку схильність до перенавчання (overfitting) при малій кількості прикладів, оскільки мають значно меншу кількість параметрів, ніж нейронні мережі. Вони здатні давати прийнятний результат («baseline») навіть маючи лише кілька сотень розмічених документів.

Векторизація TF-IDF створює ортогональні вектори для різних слів. Це означає, що алгоритм не бачить жодного зв'язку між синонімами (наприклад, «авто» і «машина»). Для моделі це абсолютно різні ознаки, що різко знижує якість класифікації текстів з багатою лексикою.

Модель не розрізняє речення «Собака вкусила людину» та «Людина вкусила собаку», оскільки набір слів однаковий. Це унеможливлює вирішення складних

задач, таких як розпізнавання сарказму, аналіз тональності зі складними запереченнями або визначення ролей у реченні.

При великому словнику (100 000+ слів) вектори стають надзвичайно довгими та розрідженими (більшість значень – нулі), що призводить до неефективного використання оперативної пам'яті та проблеми «прокляття розмірності».

Нейромережеві методи зі статичними ембедінгами (Word2Vec/GloVe + CNN/LSTM)

Ця група методів здійснила перехід від статистичного підрахунку слів до їх векторного представлення у щільному семантичному просторі, використовуючи згорткові (CNN) або рекурентні (RNN/LSTM) нейронні мережі для аналізу. Їх перевагами можна визначити:

- врахування семантичної близькості – використання попередньо навчених ембедінгів (Pre-trained Word Embeddings) дозволяє моделі розуміти синонімію. Модель, навчена на відгуках про «фільми», може успішно класифікувати відгук про «кіно», навіть якщо цього слова не було в навчальній вибірці, завдяки близькості їх векторів;
- здатність моделювати послідовності – архітектури LSTM та GRU здатні зберігати інформацію про порядок слів та враховувати залежності на відносно довгих дистанціях. Це дозволяє значно точніше аналізувати тональність та синтаксичну структуру, ніж підхід «мішка слів»;
- автоматичне виділення ознак (Feature Extraction) – згорткові мережі (CNN) автоматично навчаються виявляти значущі локальні патерни (N-грами), такі як стійкі словосполучення («не рекомендую», «дуже сподобалось»), позбавляючи дослідника необхідності ручного конструювання ознак.

Головним недоліком є те, що кожне слово має фіксований вектор. Слово «ключ» (музичний) і «ключ» (від замка) представлені одним і тим самим вектором, що є усередненням усіх його значень. Словник статичних ембедінгів фіксований. Будь-яке слово, якого не було в навчальному корпусі (сленг, помилка, специфічний термін), замінюється на токен <UNK>, що призводить до повної втрати інформації про нього. Рекурентна природа LSTM (послідовна обробка слово за словом)

унеможливлює повну паралелізацію обчислень на GPU, що робить процес навчання повільним порівняно з CNN або Трансформерами.

Контекстуалізовані моделі на основі Трансформерів (BERT, RoBERTa)

Це сучасний State-of-the-Art (SOTA) підхід, що базується на механізмі уваги (Attention) та парадигмі «Pre-training + Fine-tuning». Його переваги це найвища точність класифікації, ефективність трансферного навчання (Transfer Learning), робота з будь-яким словником (Subword Tokenization).

Завдяки механізму Self-Attention модель аналізує кожне слово в контексті всього речення одночасно. Це дозволяє вирішити проблему полісемії (динамічні вектори) та вловлювати найтонші нюанси змісту, іронію та імпліцитні зв'язки. На сьогодні ці моделі демонструють результати, що наближаються до людського рівня розуміння.

Можливість взяти модель, попередньо навчену на гігантських корпусах даних (як Вікіпедія), і «доналаштувати» (Fine-tune) її на невеликому специфічному наборі даних дозволяє отримувати надвисокі результати навіть без наявності мільйонів розмічених прикладів.

Використання токенізації на основі підслів (WordPiece, BPE) дозволяє моделі конструювати вектори навіть для слів, яких вона ніколи не бачила, виходячи з їхніх морфологічних частин. Це повністю вирішує проблему OOV.

Але, як недоліки цього підходу, можна визначити екстремальну ресурсоємність, проблема «Чорної скриньки» (Black Box), складність впровадження (Deployment).

Моделі типу BERT містять сотні мільйонів (іноді мільярди) параметрів. Їх навчання та використання (Inference) вимагають потужних графічних прискорювачів (GPU/TPU). Використання таких моделей на звичайних CPU часто є неприйнятно повільним для реальних задач.

Основна мета BERT полягає в глибшому розумінні контексту тексту через попереднє навчання на великих наборах неанотованих даних. Це дозволяє мовній моделі вивчати контекстну інформацію і будувати зв’язки між словами, термами і

фразами, поліпшуючи результати в різних завданнях NLP, включно з генерацією тексту. [24, с. 115]

Незважаючи на спроби візуалізації уваги (Attention maps), пояснити, чому саме трансформер прийняв те чи інше рішення, надзвичайно складно через складну нелінійну структуру та величезну кількість параметрів. Це обмежує їх застосування в критично важливих сферах.

Великий розмір моделей (сотні мегабайт або гігабайти) ускладнює їх інтеграцію в мобільні додатки або вбудовані системи (Edge devices).

Для наочності проведеного аналізу зведемо ключові характеристики методів у таблицю 2.1.

Таблиця 2.1

Порівняльна характеристика методів аналізу текстів

Характеристика	**Класичні методи (TF-IDF+SVM)**	**Статичні НМ (Word2Vec+LSTM)**	**Контекстуалізовані (BERT)**
Семантичне розуміння	низьке (тільки лексика)	середнє (синоніми, локальний контекст)	високе (глибокий контекст, полісемія)
Точність класифікації	базова (Baseline)	висока	State-of-the-Art
Вимоги до даних	низькі (працює на малих вибірках)	середні/високі	низькі (при Fine-tuning)
Швидкість навчання	дуже висока	середня	низька (вимагає GPU)
Швидкість роботи	дуже висока	середня	низька
Інтерпретованість	висока	середня	низька
Вимоги до обладнання	CPU	CPU/GPU	GPU/TPU

Проведений аналіз показує, що в галузі Data Science не існує універсального алгоритму, який би був найкращим за всіма критеріями одночасно. Класичні методи виграють у швидкості та прозорості, але програють у точності розуміння смислу. Сучасні трансформери забезпечують максимальну якість аналізу, але є ресурсоємними та складними в експлуатації.

Ця дихотомія підтверджує актуальність теми магістерської роботи. Існує гостра потреба у розробці комплексної методики, яка б не просто застосовувала найпотужніший алгоритм, а дозволяла б обґрунтовано обирати та комбінувати методи залежно від специфіки вхідних даних, наявних ресурсів та бізнес-вимог до системи.

2.3 Математична модель процесу семантичної класифікації текстів

Для розробки ефективної методики автоматизованого семантичного аналізу та класифікації, необхідно спершу формалізувати сам процес, представивши його у вигляді строгої математичної моделі. Ця модель описує фундаментальні етапи перетворення неструктурованого текстового масиву в чіткий класифікаційний висновок.

При розробці математичної моделі використовувався підхід до моделювання систем обробки текстового контенту, запропонований В.А. Висоцькою, яка розглядає текстовий масив як складну інформаційну систему з власними метриками якості [5]. Також враховано принципи побудови баз знань інтелектуальних систем, викладені В.В. Литвином, зокрема використання онтологічного підходу для зняття семантичної неоднозначності [9].

Процес семантичної класифікації текстів, з математичної точки зору, можна визначити як задачу пошуку оптимальної функції-відображення (hypothesis function) h, яка ставить у відповідність кожному документу d з вхідного простору документів D одну мітку класу c з цільового простору класів C.

$$h: D \rightarrow C. \tag{2.10}$$

Для побудови такої функції h, процес необхідно декомпозувати на декілька ключових етапів.

1. Формальне визначення вхідних даних

Нехай задано наступні множини:

- Корпус документів D – це скінченна, але потенційно дуже велика множина текстових документів, що підлягають аналізу:

$$D = \{d_1, d_2, \dots, d_N\}, \tag{2.11}$$

де N – загальна кількість документів у корпусі.

- Множина класів C – це скінченна, заздалегідь визначена множина M категорій (класів), до яких може належати документ:

$$C = \{c_1, c_2, \dots, c_M\}. \tag{2.12}$$

- Навчальна вибірка T (для керованого навчання) – це підмножина документів $T \subset D$, для яких відомі "істинні" мітки класів. T є множиною пар "документ-клас":

$$T = \{(d_i, y_i) \mid d_i \in D, y_i \in C\}, \tag{2.13}$$

де y_i – це істинна мітка класу для документа d_i.

Головна мета: побудувати функцію (модель) h, яка, навчившись на вибірці T, здатна максимально точно передбачити клас $\hat{y}_j$ для будь-якого нового, не баченого раніше документа $d_j \notin T$.

$$\hat{y}_j = h(d_j) \text{ де } \hat{y}_j \in C, \tag{2.14}$$

де $y_j \in C$.

Функція векторизації (представлення ознак)

Математичні алгоритми не можуть працювати безпосередньо з "сирим" текстом d_i. Першим кроком є перетворення кожного документа у числовий вектор. Цей процес описується функцією векторизації f:

$$f: D \to R^k, \tag{2.15}$$

де R^k– це k-вимірний векторний простір ознак.

Результатом застосування цієї функції до документа d_iє вектор ознак v_i:

$$v_i = f(d_i) = (w_{i,1}, w_{i,2}, \dots, w_{i,k}). \tag{2.16}$$

Математична модель цієї функції f суттєво відрізняється залежно від обраного підходу (як було проаналізовано в 2.1.3):

- Модель "Мішка слів" (BoW) / TF-IDF:

Тут k дорівнює потужності словника V (кількості унікальних слів у корпусі). Вектор v_i є розрідженим. Кожна j-та компонента вектора $w_{i,j}$ обчислюється, наприклад, як вага TF-IDF:

$$w_{i,j} = tf(t_j, d_i) \times idf(t_j, D), \tag{2.17}$$

де $tf(t_j, d_i)$ – частота терміну t_j у документі d_i, а $idf(t_j, D)$ – обернена документна частота терміну t_j у всьому корпусі D.

- Модель статичних ембедінгів (Word2Vec/GloVe):

Тут k є фіксованою розмірністю простору ембедінгу (напр., $k = 300$). Функція f є більш складною: вона спершу перетворює кожне слово t_j у d_i на його вектор $e_j \in R^k$, а потім агрегує ці вектори (наприклад, шляхом усереднення або зваженого усереднення) для отримання фінального вектора документа v_i:

$$v_i = \frac{1}{|d_i|} \sum_{t_j \in d_i} e_j. \tag{2.18}$$

- Модель контекстуалізованих ембедінгів (BERT):

Тут функція f є глибокою нейронною мережею (Трансформером). f приймає на вхід послідовність токенів d_i і генерує вектор v_i (зазвичай це вектор, що відповідає спеціальному токену [CLS]), який представляє семантику всього документа в контексті:

$$v_i = BERT(d_i). \tag{2.19}$$

Функція класифікації (модель прийняття рішень)

Другим кроком є побудова функції-класифікатора g, яка приймає на вхід числовий вектор $v_i \in R^k$ і повертає прогнозований клас $\hat{y}_i \in C$.

$$g: R^k \to C. \tag{2.20}$$

Загальна модель h є композицією цих двох функцій:

$$h(d_i) = g\big(f(d_i)\big). \tag{2.21}$$

Математична модель g також залежить від обраного інтелектуального алгоритму:

- Імовірнісна модель (напр., Naive Bayes):

Модель g шукає клас c_j, який максимізує апостеріорну ймовірність $P\big(c_j\big|v_i\big)$ за теоремою Баєса:

$$y_i = arg \max_{c_j \in C} P\big(c_j\big|v_i\big) = arg \max_{c_j \in C} \frac{P\big(v_i\big|c_j\big)P(c_j)}{P(v_i)}. \tag{2.22}$$

З урахуванням "наївного" припущення про незалежність ознак $\big(w_{i,k}\big)$, це спрощується до:

$$y_i = \arg\max_{c_j \in C} P(c_j) \prod_{k=1}^{K} P(w_{i,k}|c_j). \quad (2.23)$$

Лінійна модель (напр., SVM або логістична регресія):

Модель g шукає набір вагових коефіцієнтів W (матриця $M \times k$) та вектор зсувів b (вектор $M \times 1$). Для бінарного випадку SVM, g є гіперплощиною:

$$g(v_i) = sign(W \cdot v_i + b). \quad (2.24)$$

Для логістичної регресії, g видає ймовірність належності до класу, пропускаючи лінійну комбінацію через сигмоїдну (для *M=2*) або softmax (для *M>2*) функцію:

$$\hat{y}_i = softmax(W v_i + b), \quad (2.25)$$

де $softmax(z_j)$ визначається як:

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{l=1}^{M} e^{z_l}}. \quad (2.26)$$

Навчання та функція втрат

Навчання моделі полягає у знаходженні оптимального набору параметрів θ (наприклад, $\theta = \{W, b\}$ для лінійної моделі), які найкраще описують навчальну вибірку *T*. "Найкраще" означає мінімізацію функції втрат (Loss Function) $\mathcal{L}$.

Функція втрат $\mathcal{L}(y_i, \hat{y}_i)$ вимірює "штраф" за невірне передбачення $\hat{y}_i = g(f(d_i))$ у порівнянні з істинною міткою y_i.

Загальна задача навчання (оптимізації) формулюється так:

$$\theta^* = arg\ \min_{\theta} \sum_{(d_i, y_i) \in T} \mathcal{L}(y_i, h(d_i; \theta)) + R(\theta), \quad (2.27)$$

де $R(\theta)$ – це (необов'язковий) член регуляризації для запобігання перенавчанню (напр., L1 або L2 регуляризація).

Конкретний вигляд $\mathcal{L}$ залежить від моделі:

- Hinge Loss (для SVM):

$$\mathcal{L} = max(0,1 - y_i \cdot \hat{y}_i). \quad (2.28)$$

- Cross-Entropy Loss (для логістичної регресії та нейромереж):

$$\boldsymbol{\mathcal{L}} = -\sum_{j=1}^{M} y_{i,j} \log(\widehat{y_{i,j}}), \quad (2.29)$$

де $y_{\{i,j\}}$ дорівнює 1, якщо документ i належить до класу j, і 0 в іншому випадку.

Мінімізація цієї функції зазвичай відбувається ітеративно за допомогою методів, заснованих на градієнті, таких як стохастичний градієнтний спуск (SGD) або його варіації (Adam, RMSProp):

$$\theta_{t+1} = \theta_t - \eta \nabla \boldsymbol{\mathcal{L}}(\theta_t), \quad (2.30)$$

де η – швидкість навчання, а $\nabla \boldsymbol{\mathcal{L}}$ – градієнт функції втрат.

Таким чином, повна математична модель процесу семантичної класифікації текстів є композитною функцією $h = g \circ f$, яка відображає множину документів D у множину класів C. Ця модель включає:

- функцію векторизації f, що переводить текст у числове представлення R^k;

- функцію класифікації g, що відображає цей вектор у простір класів C;

- функцію втрат $\mathcal{L}$ та алгоритм оптимізації, які використовуються для знаходження оптимальних параметрів θ* моделі h на основі навчальних даних.

Ця формалізована структура дозволяє уніфіковано описувати, порівнювати та будувати різні методики класифікації, що і є предметом даного дослідження.

3 РОЗРОБКА МЕТОДИКИ АВТОМАТИЗОВАНОГО СЕМАНТИЧНОГО АНАЛІЗУ ТА КЛАСИФІКАЦІЇ ТЕКСТОВИХ МАСИВІВ

3.1 Опис розробки методики (етапи, загальна архітектура)

Процес створення методики автоматизованого семантичного аналізу та класифікації текстових масивів не був лінійним, а реалізовувався через серію ітеративних етапів. Такий підхід, заснований на принципах гнучкої розробки (Agile), дозволив послідовно нарощувати функціональність системи, перевіряти гіпотези на кожному кроці та адаптувати архітектуру під виявлені вимоги. Розробка охоплювала повний цикл Data Science проекту: від концептуалізації та збору даних до розгортання моделі у вигляді веб-сервісу.

На початковому етапі було проведено аналіз вимог до системи та визначено її ключову мету: створення гнучкого, масштабованого інструменту, здатного ефективно працювати з текстовими масивами різними мовами.

Критично важливим стало прийняття архітектурного рішення щодо стратегії мультимовності. Було розглянуто два підходи: використання єдиної великої багатомовної моделі (наприклад, mBERT) або створення ансамблю спеціалізованих моделей.

В результаті було обґрунтовано та обрано архітектурний патерн «одна модель для кожної мови» (Model-per-Language). На рисунку 3.1 наведена обрана стратегія мультимовної обробки даних.

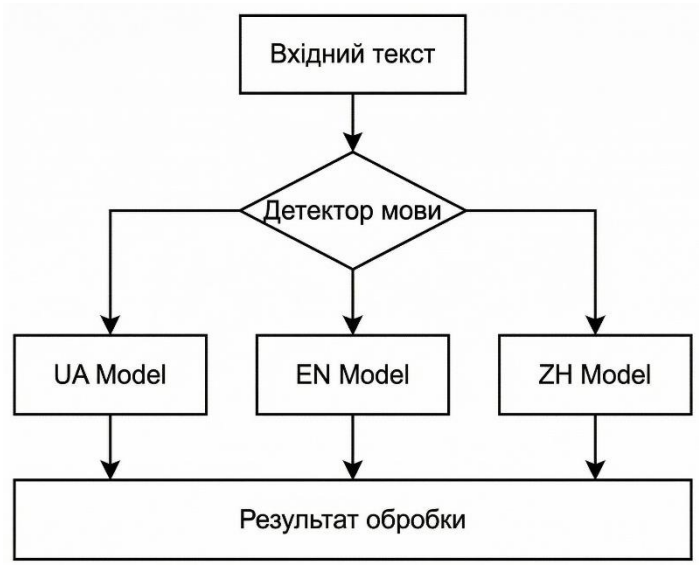


Рис. 3.1. Стратегія мультимовної обробки даних

Цей вибір зумовлений наступними факторами як лінгвістична точність, модульність, оптимізація ресурсів.

Такий підхід дозволяє застосовувати унікальні правила препроцесингу, специфічні для кожної мови (наприклад, врахування відмінкової системи в українській мові або відсутності пробілів у японській), що неможливо в універсальних моделях. Можливість додавати підтримку нової мови без необхідності перенавчання всієї системи. Спеціалізовані моделі мають меншу розмірність словника, що дозволяє їм працювати швидше та вимагати менше оперативної пам'яті.

На наступному етапі відбувалося формування та ітеративне розширення навчального набору даних. Якість класифікації напряму залежить від якості вхідних даних. На цьому етапі було сформовано навчальний датасет (sample_data.csv). Процес відбувався ітеративно. Було зібрано базовий корпус текстів для трьох фундаментальних категорій: «Спорт», «Політика» та «Технології». Це дозволило протестувати працездатність пайплайну (пілотна ітерація). На наступних ітераціях набір даних було суттєво розширено як кількісно (збільшення зразків у існуючих класах для запобігання дисбалансу класів), так і якісно. Було введено чотири нові семантичні категорії: «Культура», «Наука», «Здоров'я» та «Бізнес». Це розширило предметну область класифікатора до 7

класів, наблизивши його до реальних умов використання у новинних агрегаторах чи системах моніторингу (масштабування).

Далі було розроблено модульний конвеєр обробки даних, який адаптується під обрану мову. Preprocessing Pipeline, тобто розробка уніфікованого конвеєра попередньої обробки, реалізація включає як універсальні, так і специфічні методи:

- універсальні методи – очищення тексту від HTML-тегів, спецсимволів, цифр, приведення до нижнього регістру, видалення стоп-слів (слів, що не несуть семантичного навантаження);
- специфічні методи (Language-specific) – для української мови застосовано лематизацію (приведення слова до словникової форми) для боротьби з високою флективністю мови, для європейських мов (англійська, французька) використано стемінг (відсікання закінчень) як більш швидкий метод нормалізації, для азійських мов (китайська, японська) впроваджено алгоритми сегментації (Tokenization), оскільки в цих мовах слова не розділяються пробілами, що робить стандартні методи непридатними.

Для реалізації ядра системи було обрано гібридний підхід, що поєднує надійність статистичних методів з потужністю нейронних мереж (створення гібридної моделі класифікації):

Векторизація (Feature Extraction). Текстові дані перетворюються на числові вектори за допомогою методу TF-IDF (Term Frequency-Inverse Document Frequency). Це дозволяє оцінити важливість слова в контексті документа та корпусу, фільтруючи загальновживані слова.

Класифікація (Deep Learning). Отримані вектори подаються на вхід повнозв'язної нейронної мережі (Feed-Forward Neural Network), реалізованої за допомогою бібліотек TensorFlow та Keras. Архітектура мережі включає вхідний шар, приховані шари з функцією активації ReLU для моделювання нелінійних залежностей, та вихідний шар з функцією активації Softmax для отримання ймовірнісного розподілу по 7 категоріях.

Щоб перетворити модель з експериментального скрипта на повноцінний сервіс, було розроблено веб-сервер на базі фреймворку Flask. API реалізує

архітектуру REST і надає два ключові ендпоінти: predict – для класифікації окремого тексту в реальному часі (низька затримка) та predict_batch – для пакетної обробки масивів текстів. Цей метод оптимізує обчислення, дозволяючи векторизувати та класифікувати одразу групу документів, що значно підвищує пропускну здатність системи.

Для забезпечення зручної взаємодії користувача з системою (User Experience) було створено клієнтський інтерфейс ("Frontend"). Він реалізований як односторінковий додаток (SPA) з використанням технологій HTML5, CSS3 та JavaScript. Інтерфейс асинхронно взаємодіє з API, дозволяючи користувачу вводити текст, динамічно перемикати мову моделі та миттєво отримувати результати класифікації та візуалізацію впевненості моделі без перезавантаження сторінки.

На фінальному етапі увагу було зосереджено на надійності, відтворюваності та безпеці системи:

- Створено Dockerfile, що описує все необхідне оточення (версії Python, бібліотеки TensorFlow, NLTK). Це гарантує, що методика працюватиме ідентично на будь-якому комп'ютері чи сервері («It works on my machine» problem solved) (контейнеризація (Docker).

- Всі параметри (шляхи до моделей, порти, налаштування) винесено у змінні середовища, що відповідає методології 12-factor app (конфігурація).

- Впроваджено систему логування для моніторингу вхідних запитів та роботи моделі (логування).

Розроблена методика реалізовується у вигляді модульної програмної системи, архітектуру якої можна декомпозувати на три функціонально незалежні, але взаємопов'язані компоненти (див. рис.3.2).

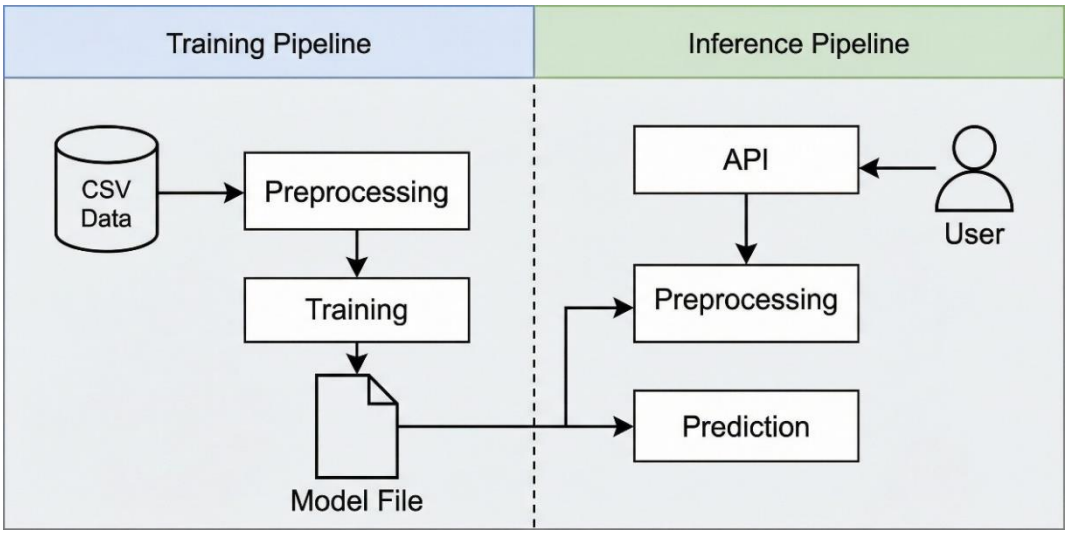


Рис. 3.2. Загальна архітектура розробленої системи семантичного аналізу.

Така структура забезпечує слабку зв'язність (loose coupling) та високу когезію (high cohesion) модулів.

Конвеєр навчання (Training Pipeline). Це автономний модуль (Offline component), який запускається періодично або за вимогою для оновлення знань системи. Його функції – це завантаження сирих даних (.csv), очищення та нормалізація тексту, навчання векторизатора TF-IDF, тренування нейронної мережі, оцінка якості. Результатом роботи є серіалізовані файли ("артефакти"), що зберігаються на диску: навчені файли моделей (.h5 або .keras), об'єкти векторизаторів (.pickle) та кодувальники міток класів. Ці файли є «знаннями» системи.

Сервер логічного виведення (Inference Server). Це серверна частина (Backend), яка працює в режимі 24/7. При старті сервер завантажує в оперативну пам'ять усі підготовлені на попередньому етапі артефакти. Це критично для швидкодії, щоб не завантажувати модель при кожному запиті. Сервер діє як маршрутизатор. Отримуючи HTTP-запит з текстом та кодом мови, він обирає відповідний пайплайн (наприклад, "Ukrainian Pipeline"), проводить препроцесинг, векторизацію та передає вектор у відповідну нейронну мережу. Повертає відповідь у форматі JSON, що містить передбачену категорію та ймовірності для всіх інших класів.

Клієнтський рівень (Presentation Layer). Це «фасад» системи для кінцевого користувача. Він повністю відокремлений від логіки класифікації. Клієнтський додаток лише відправляє дані на сервер і відображає отриману відповідь. Це дозволяє в майбутньому легко замінити веб-інтерфейс на мобільний додаток або інтегрувати API в сторонні системи (наприклад, чат-боти) без змін у серверній частині.

Така триланкова архітектура забезпечує гнучкість методики: нові дані покращують Training Pipeline, оптимізація коду прискорює Inference Server, а зміни дизайну стосуються лише Frontend, що робить систему легкою для підтримки та подальшого розвитку.

3.2 Опис використаних програмних засобів та бібліотек

Реалізація розробленої методики автоматизованого семантичного аналізу та класифікації текстових масивів вимагала формування надійного, масштабованого та високопродуктивного технологічного стеку. Вибір програмних засобів базувався на принципах відкритості вихідного коду (Open Source), наявності активної спільноти розробників, ефективності математичних обчислень та підтримки сучасних алгоритмів глибокого навчання.

В результаті аналізу вимог до системи було обрано екосистему мови програмування Python, яка на сьогодні є індустріальним стандартом у сферах Data Science (науки про дані), Machine Learning (машинного навчання) та NLP (обробки природної мови).

Нижче наведено детальний опис використаних програмних компонентів, систематизованих за їхнім функціональним призначенням.

В якості основної мови реалізації було обрано **Python**. Ця високорівнева інтерпретована мова з динамічною типізацією забезпечує оптимальний баланс між швидкістю розробки та продуктивністю виконання коду (за рахунок використання бібліотек, написаних на C/C++). Ключовими факторами вибору стали: лаконічний

синтаксис, що полегшує роботу зі складними структурами даних, та наявність найбагатшої у світі екосистеми бібліотек для штучного інтелекту.

Щодо до засобів обробки та маніпуляції даними, то фундаментом для будь-яких операцій з даними слугували бібліотеки для наукових обчислень NumPy та Pandas.

NumPy (Numerical Python). Бібліотека, що лежить в основі наукового стеку Python. Вона надає об'єкт багатовимірного масиву (ndarray) та широкий набір функцій лінійної алгебри. У даній роботі NumPy використовувався для низькорівневих операцій з векторами та матрицями ознак, забезпечуючи векторизовані обчислення, що на порядки швидші за стандартні цикли Python.

Pandas. Високорівнева бібліотека для аналізу даних, побудована поверх NumPy. Основна структура даних – DataFrame – використовувалася для завантаження та парсингу початкового набору даних із форматів CSV/JSON, очищення даних, обробки пропущених значень та дедублікації записів, попереднього розвідувального аналізу (EDA), групування текстів за категоріями та балансування класів.

Для реалізації конвеєра попередньої обробки (Preprocessing Pipeline) було задіяно комплекс бібліотек, що дозволило реалізувати стратегію мультимовності.

Універсальні та базові інструменти:

LTK (Natural Language Toolkit). Одна з найстаріших та найавторитетніших бібліотек для комп'ютерної лінгвістики. У проекті вона використовувалася для базових задач препроцесингу англійської та європейських мов: токенізація – розбиття потоку тексту на речення та окремі слова (токени); стемінг – використання алгоритму Портера (Porter Stemmer) для приведення слів до їхньої основи шляхом відсікання афіксів; corpus management – використання вбудованих списків стоп-слів (Stopwords corpus) для фільтрації слів, що не несуть смислового навантаження.

Спеціалізовані лінгвістичні бібліотеки (для конкретних мов):

Pymorphy3. Потужний морфологічний аналізатор для української мови. Оскільки українська мова є флективною (слова змінюються за відмінками, родами,

числами), простий стемінг тут неефективний. Pymorphy3 використовувався для лематизації – приведення слова до його нормальної словникової форми (леми) з урахуванням контексту, що значно підвищило якість векторизації.

Jieba. Бібліотека для сегментації китайського тексту. Специфіка китайської мови полягає у відсутності пробілів між словами. Jieba використовує алгоритми на основі графів та динамічного програмування (Viterbi algorithm) для знаходження найбільш ймовірного поділу рядка ієрогліфів на окремі семантичні одиниці.

MeCab. Інструмент для морфологічного аналізу та токенізації японської мови. Використовувався для коректного розбиття речень на слова та частки, що є критичним для аглютинативної структури японської мови.

Засоби векторизації (Feature Extraction):

Scikit-learn (модуль feature_extraction.text). Це безкоштовна бібліотека машинного навчання для мови програмування Python, яка надає інструменти для реалізації завдань класифікації, регресії, кластеризації та зниження розмірності даних. Вона побудована на основі бібліотек SciPy та NumPy, що дозволяє працювати з багатовимірними масивами та виконувати наукові обчислення.

TfidfVectorizer. Ключовий компонент системи. Він перетворює колекцію текстових документів у матрицю ознак TF-IDF. Цей клас автоматично виконує побудову словника, підрахунок частот токенів та їх зважування, зменшуючи вплив частотних загальновживаних слів і підкреслюючи важливість унікальних термінів, характерних для конкретних категорій.

Засоби машинного та глибокого навчання – ядро інтелектуального аналізу було реалізовано з використанням передових фреймворків:

TensorFlow. Відкрита платформа для машинного навчання від Google, що забезпечує побудову та тренування моделей на основі графів обчислень. Вона дозволяє ефективно використовувати ресурси CPU (і GPU при наявності) для тензорних операцій.

Keras. Високорівневий API нейронних мереж, що працює поверх TensorFlow. Він був обраний для швидкого прототипування архітектури моделі.

- Sequential. Модель послідовного типу, що дозволяє лінійно компонувати шари мережі.

- Dense. Повнозв'язні шари, які формують основу перцептрону, вивчаючи нелінійні комбінації ознак.

- Dropout. Шар регуляризації, який випадковим чином "вимикає" частину нейронів під час навчання, що є ефективним методом запобігання перенавчанню (overfitting) на невеликих датасетах.

Scikit-learn (допоміжні модулі).

- LabelEncoder: для перетворення текстових міток класів (наприклад, "health", "tech") у цілочисельний формат $[0, N-1]$, необхідний для обчислення функції втрат.

- model_selection.train_test_split: для коректного розділення даних на навчальну та тестову вибірки із збереженням пропорцій класів (stratification).

- metrics.classification_report: для генерації детального звіту з метриками Precision (точність), Recall (повнота) та F1-score для кожного класу окремо.

Для перетворення розробленої математичної моделі на повноцінний програмний продукт були використані технології веб-розробки та DevOps.

Flask. Легковаговий веб-фреймворк (мікрофреймворк) для Python. Він став основою для створення **REST API**. Flask забезпечує маршрутизацію HTTP-запитів, обробку вхідних JSON-даних та повернення результатів класифікації. Його мінімалістична архітектура ідеально підходить для створення мікросервісів машинного навчання.

Docker. Платформа для контейнеризації додатків. Було створено Dockerfile, який описує повне середовище виконання додатку: від версії операційної системи (базовий образ Python-slim) до встановлення всіх необхідних бібліотек. Використання Docker вирішує проблему "dependency hell" та гарантує, що система буде працювати ідентично на локальному комп'ютері розробника, тестовому стенді та продуктивному сервері.

Використання цього комплексу програмних засобів дозволило створити надійну, модульну та масштабовану систему, що відповідає сучасним вимогам до розробки інтелектуального програмного забезпечення.

3.3 Опис структури програмної реалізації методики

Програмна реалізація розробленої методики виконана у вигляді модульного проєкту, структура якого підпорядкована принципам чистої архітектури та розділення відповідальності (Separation of Concerns). Така організація файлової системи забезпечує масштабованість рішення, легкість у підтримці кодової бази та зручність розгортання як у середовищі розробки, так і на продуктовому сервері, що показано на рисунку 3.3.

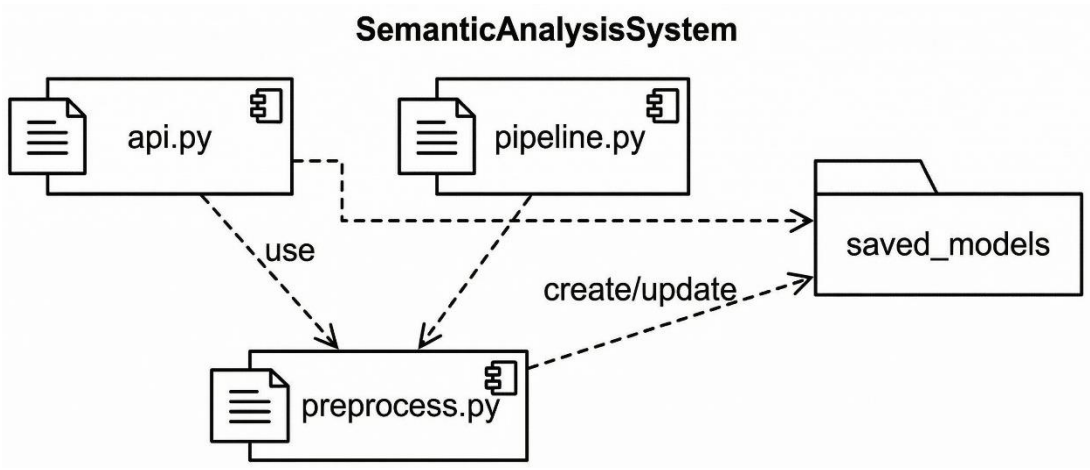


Рис. 3.3. Структура файлів та модулів програмного комплексу.

Архітектуру системи логічно розділено на два незалежні функціональні конвеєри (pipelines), що працюють у різних режимах, а також набір допоміжних компонентів для управління конфігурацією та даними.

Конвеєр навчання (Offline Training Pipeline) відповідає за повний цикл підготовки та навчання моделей. Він функціонує в режимі "offline", тобто запускається періодично за вимогою адміністратора або розробника для оновлення знань системи:

- pipeline.py (оркестратор навчання)

Скрипт, що реалізує логіку конвеєра. Він приймає ідентифікатор мови як аргумент командного рядка і послідовно виконує етапи: зчитування даних, попередню обробку, навчання нейронної мережі та оцінку якості. Ключовою функцією модуля є серіалізація (збереження) отриманих артефактів.

- sample_data.csv (джерело даних)

Файл, що виступає єдиним джерелом істини (Single Source of Truth) для навчальних даних. Використання універсального формату CSV забезпечує легку інтеграцію з інструментами аналізу (Pandas) та гарантує, що кожен запуск навчання використовує ідентичний набір даних, що є критично важливим для відтворюваності експериментів.

- preprocess.py (модуль лінгвістичної обробки)

Інкапсулює логіку очищення та нормалізації тексту. Реалізує патерн «Фасад», надаючи єдину точку входу для обробки тексту незалежно від мови, що дозволяє уникнути дублювання коду (принцип DRY) між конвеєрами навчання та використання.

Конвеєр логічного виведення (Online Inference Pipeline) відповідає за обробку запитів користувачів у реальному часі. Він оптимізований для мінімізації затримки (latency) та високої пропускної здатності.

- api.py (точка входу веб-додатку) – основний виконуваний файл серверної частини. Він ініціалізує веб-сервер на базі Flask. При старті додаток завантажує попередньо навчені моделі з диска в оперативну пам'ять, що дозволяє миттєво обробляти вхідні запити. Модуль визначає маршрути API (/predict, /predict_batch) та рендерить HTML-шаблони інтерфейсу.

Управління конфігурацією та залежностями. Для забезпечення стабільності роботи системи на різних обчислювальних машинах (від ноутбука розробника до хмарного сервера) реалізовано суворі практики управління оточенням.

- config.py (централізована конфігурація) – у цьому модулі зібрані всі налаштування системи: гіперпараметри нейронної мережі (кількість епох, розмір

батча), шляхи до файлів та параметри векторизації. Винесення цих значень з коду бізнес-логіки дозволяє змінювати поведінку системи без необхідності редагування вихідного коду, що відповідає методології Twelve-Factor App.

- requirements.txt (фіксація залежностей) – файл містить точний перелік бібліотек та їх версій (наприклад, pandas==2.0.3, tensorflow==2.13.0). Це вирішує проблему "Dependency Hell" та гарантує створення ідентичного середовища виконання, унеможливлюючи помилки, пов'язані з оновленням API сторонніх бібліотек.

Структура директорій та артефакти

Файлова система проєкту організована таким чином, щоб відокремити вихідний код від даних та бінарних файлів.

- saved_model/ – директорія для зберігання результатів роботи конвеєра навчання. Для кожної мови створюється окрема підпапка, що містить три обов'язкові компоненти: файл моделі Keras (.h5), серіалізований векторизатор (vectorizer.joblib) та кодувальник міток (label_encoder.joblib).
- templates/ та static/ – стандартні директорії веб-фреймворку Flask. У першій зберігаються HTML-шаблони сторінок, у другій – статичні ресурси (CSS-стилі, клієнтські JavaScript-скрипти, зображення), що забезпечує чітке розділення frontend та backend частин.
- tests/ – містить набір unit-тестів для перевірки коректності роботи окремих модулів (зокрема, функцій препроцесингу), що дозволяє контролювати якість коду перед розгортанням.

3.4 Опис розроблених модулів та класів

Архітектура розробленої програмної системи побудована за модульним принципом, де кожен компонент інкапсулює специфічну логіку обробки даних. Реалізація виконана переважно у функціональному стилі, що забезпечує чистоту коду, легкість тестування та зручність масштабування. Система складається з трьох

ключових модулів: модуля попередньої обробки (preprocess.py), модуля керування навчанням (pipeline.py) та модуля програмного інтерфейсу (api.py).

Модуль лінгвістичної обробки (preprocess.py) є фундаментом системи, оскільки якість вхідних даних критично впливає на результати класифікації. Він відповідає за трансформацію «сирого» тексту у нормалізований вигляд.

Ключовим елементом модуля є функція-диспетчер preprocess_text(text, language), яка реалізує патерн «фасад». Вона приймає вхідний рядок тексту та ідентифікатор мови, і на основі цього динамічно маршрутизує потік виконання до відповідного спеціалізованого алгоритму.

```
# Фрагмент реалізації маршрутизації у preprocess.py
# Словник для маршрутизації функцій обробки залежно від мови
PREPROCESSORS = {
    'ukrainian': preprocess_ukrainian_text,
    'english': preprocess_english_text,
    'german': preprocess_german_text,
    'french': preprocess_french_text,
    'spanish': preprocess_spanish_text,
    'chinese': preprocess_chinese_text,
    'japanese': preprocess_japanese_text,
}
# Функція-фасад для виклику відповідного препроцесора
def preprocess_text(text, language):
    preprocessor = PREPROCESSORS.get(language)
    if preprocessor:
        return preprocessor(text)
    return text
```

Загальний алгоритм обробки, реалізований у модулі, складається з наступних послідовних етапів:

1. Усі символи тексту приводяться до нижнього регістру для зменшення розмірності словника, щоб слова сприймалися як один токен, тобто проводиться нормалізація регістру.

2. Виконується видалення знаків пунктуації, спеціальних символів та цифр, які не несуть семантичного навантаження для задачі визначення теми тексту, це очищення від шуму.

3. Далі, токенізація – розбиття суцільного рядка на окремі лінгвістичні одиниці (слова).

4. Використовуючи словники з бібліотеки NLTK, система видаляє найбільш вживані функціональні слова (прийменники, сполучники, займенники), що дозволяє залишити лише значущі терміни, тобто фільтрація стоп-слів.

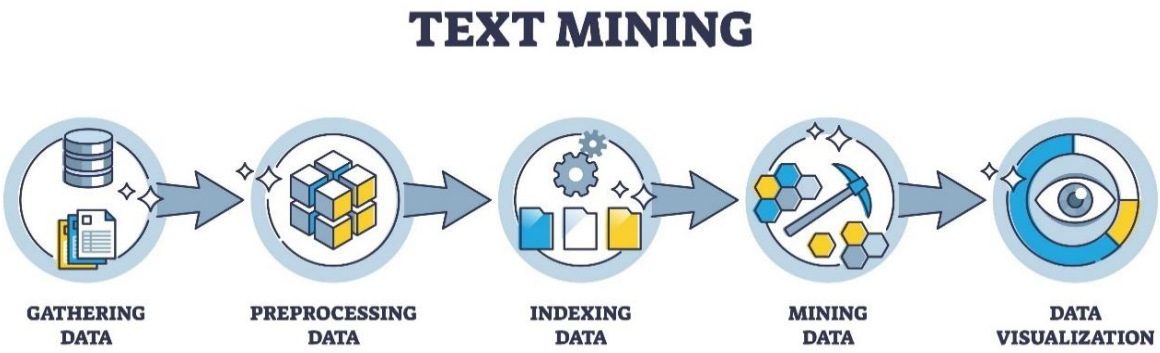


Рис. 3.4 Схема процесу інтелектуального аналізу неструктурованих текстових даних

Критично важливою особливістю модуля є адаптивна нормалізація залежно від мовної групи.

Для української мови реалізовано процес лематизації з використанням бібліотеки pymorphy3. Оскільки українська мова є флективною, важливо приводити слово до його канонічної (словникової) форми, а не просто відкидати закінчення, щоб зберегти семантику.

Для європейських мов (EN, DE, FR, ES) застосовано алгоритм стемінгу (SnowballStemmer). Для цих мов відсікання суфіксів та закінчень є достатнім та обчислювально ефективним методом для групування однокореневих слів.

Для азійських мов (ZH, JA), оскільки в китайській та японській мовах слова не розділяються пробілами, застосовано спеціалізовані алгоритми сегментації. Для китайської мови використовується бібліотека jieba, яка будує граф ймовірних розбиттів, а для японської – інструмент MeCab, що проводить глибокий морфологічний аналіз для коректного виділення токенів.

Усе це наглядно відображено на рисунку 3.5.

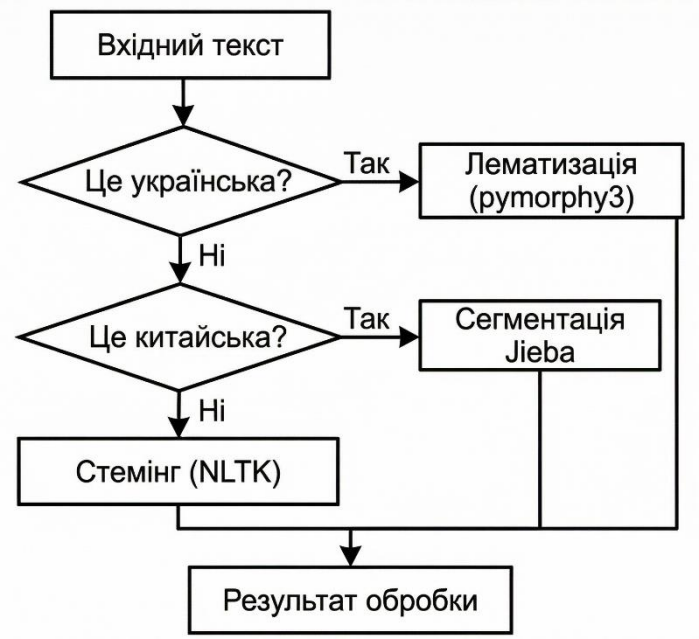


Рис. 3.5. Алгоритм адаптивної попередньої обробки тексту.

Завершальним етапом є рекомбінація оброблених токенів назад у єдиний текстовий рядок, готовий до векторизації.

Модуль конвеєра навчання (pipeline.py) виступає оркестратором процесу машинного навчання. Він автоматизує повний життєвий цикл моделі: від завантаження даних до збереження готових артефактів.

Центральною функцією є train_language_model(language), яка реалізує наступний алгоритмічний ланцюжок:

- **Ingestion** (завантаження даних) – зчитування головного датасету sample_data.csv та фільтрація записів за цільовою мовою.

- **Preprocessing** (обробка) – пакетне застосування функції preprocess_text до всього набору даних.

- **Label Encoding** (кодування цільової змінної) – перетворення текстових категорій (наприклад, «спорт», «політика») у числові вектори за допомогою LabelEncoder. Важливо, що стан кодувальника зберігається (серіалізується) у файл, щоб забезпечити коректне декодування прогнозів у майбутньому.

- **Data Splitting** (розділення) – формування навчальної та тестової вибірок для валідації моделі.

- **Vectorization** (векторизація ознак) – навчання об'єкта TfidfVectorizer. На цьому етапі формується словник токенів та розраховуються ваги IDF. Отриманий векторизатор також зберігається на диску, що гарантує ідентичність перетворення даних під час навчання та експлуатації.

- **Model Training** (навчання моделі) – ініціалізація, компіляція та навчання нейронної мережі на базі Keras/TensorFlow на векторизованих даних.

```
def create_model(input_dim):
    model = Sequential([
        # Вхідний повнозв'язний шар з 128 нейронами та активацією ReLU
        Dense(128, activation='relu', input_dim=input_dim),
        # Шар регуляризації Dropout (50%) для запобігання перенавчанню
        Dropout(0.5),
        # Вихідний шар (Softmax для отримання ймовірностей класів)
        Dense(len(le.classes_), activation='softmax')
    ])
    # Компіляція моделі з оптимізатором Adam
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

- **Evaluation** (оцінка) – генерація метрик якості (Precision, Recall, F1-score) на тестовій вибірці для контролю ефективності.

- **Serialization** (збереження) – фінальний етап, на якому файл моделі (.keras) та допоміжні об'єкти зберігаються у відповідну директорію (наприклад, saved_model/ukrainian/).

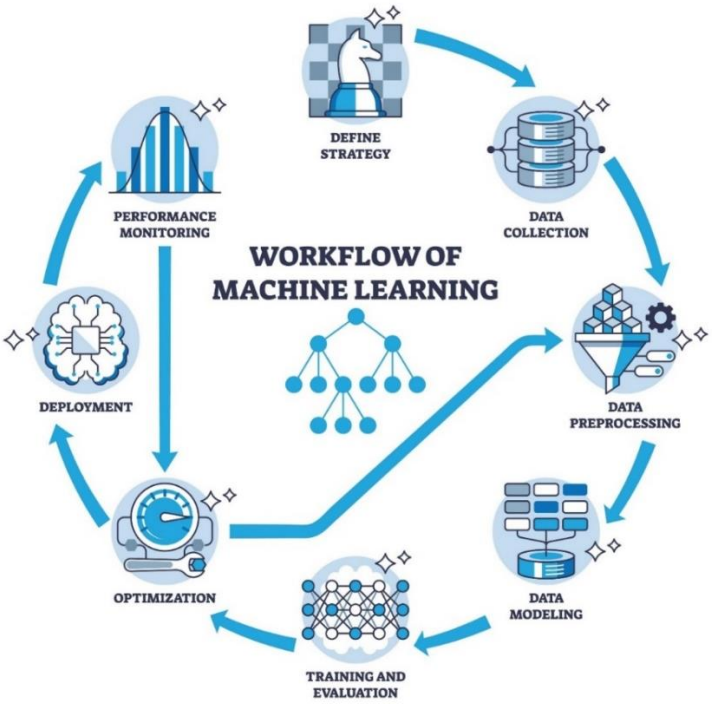


Рис. 3.6 Основні етапи робочого процесу машинного навчання.

Модуль програмного інтерфейсу (api.py) відповідає за розгортання системи (Deployment) та надання доступу до її функцій через REST API. Його архітектура базується на принципі «холодного старту» та кешування.

Функція load_models() виконується автоматично при запуску сервера. Вона сканує файлову систему, знаходить усі навчені моделі та завантажує їх (разом з векторизаторами та кодувальниками) у глобальний словник MODELS в оперативній пам'яті. Це дозволяє уникнути затримок на читання з диска при обробці запитів клієнтів.

Основним інтерфейсом взаємодії є маршрут @app.route('/predict'). Логіка обробки запиту включає: валідацію (перевірку наявності необхідних полів (text,

language) у вхідному JSON-об'єкті), попередню обробку (виклик функції preprocess_text для нового тексту), векторизацію (застосування завантаженого в пам'ять векторизатора до обробленого тексту), інференс (Inference) (отримання прогнозу від нейронної мережі), декодування (перетворення числового результату назад у текстову назву категорії та формування JSON-відповіді).

```
# Логіка обробки запиту на класифікацію
@app.route('/predict', methods=['POST'])
def predict():
    # Попередня обробка тексту з урахуванням мови
    processed_text = preprocess_text(text, language)
    # Векторизація (перетворення тексту на числовий вектор)
    vectorized_text = vectorizer.transform([processed_text]).toarray()
    # Отримання прогнозу від нейронної мережі
    prediction_proba = model.predict(vectorized_text)
    # Формування результату (сортування за ймовірністю)
    results = {le.classes_[i]: float(prediction_proba[0][i]) for i in
    range(len(le.classes_))}
    sorted_results = sorted(results.items(), key=lambda item: item[1], reverse=True)
    return jsonify({
        'category': sorted_results[0][0],
        'probabilities': sorted_results
    })
```

На рисунку 3.7 наведена діаграма послідовності обробки запиту на класифікацію.

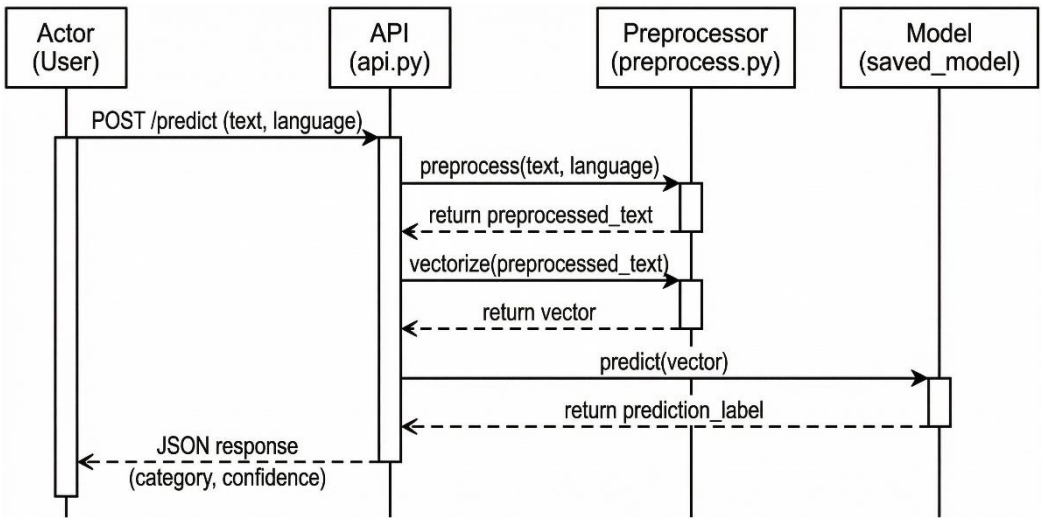


Рис. 3.7. Діаграма послідовності обробки запиту на класифікацію.

3.5 Експериментальне дослідження та оцінка ефективності методики

Заключним етапом розробки стало проведення серії експериментів для об'єктивної оцінки якості роботи запропонованої методики. Метою дослідження було порівняння ефективності розробленої гібридної моделі (TF-IDF + Deep Learning) з класичними алгоритмами та визначення її стійкості на різнорідних даних.

Для забезпечення валідності результатів експерименти проводилися на двох типах даних: синтетичному (сформованому в рамках роботи) та еталонних відкритих датасетах.

Власний мультимовний корпус (Custom Dataset), сформований згідно з методикою, містить 7000 текстових зразків (по 1000 на кожну з 7 категорій: «Спорт», «Політика», «Технології», «Культура», «Наука», «Здоров'я», «Бізнес»). Дані збалансовані, що дозволяє уникнути зміщення (bias) моделі в бік мажоритарних класів.

Еталонний датасет "20 Newsgroups" – класичний бенчмарк для задач текстової класифікації. Містить близько 20 000 документів, розподілених по 20 різних тематичних групах. Використання цього набору дозволило перевірити

здатність методики масштабуватися на велику кількість класів та працювати з "шумними" даними реального спілкування.

Еталонний датасет "IMDb Movie Reviews" – містить 50 000 відгуків на фільми для бінарної класифікації (позитивний/негативний). Цей набір використовувався для перевірки здатності моделі вловлювати семантичну тональність (sentiment analysis), а не просто тематику.

Оскільки класифікація є задачею навчання з учителем, для оцінки ефективності було використано стандартний набір метрик. Нехай TP (True Positive) – кількість істинно-позитивних спрацьовувань, TN – істинно-негативних, FP – хибно-позитивних, FN – хибно-негативних.

Accuracy (загальна точність) – показує частку правильних відповідей серед усіх прогнозів.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}. \quad (3.1)$$

Ця метрика є базовою, але може бути оманливою на незбалансованих даних.

Precision (влучність) відображає здатність моделі не присвоювати об'єктам неправильний клас (яка частка об'єктів, названих "позитивними", дійсно є такими).

$$Precision = \frac{TP}{TP+FP}, \quad (3.2)$$

де TP (True Positive) – кількість істинно-позитивних спрацьовувань; TN (True Negative) – істинно-негативних; FP (False Positive) – хибно-позитивних; FN (False Negative) – хибно-негативних.

Recall (повнота) показує здатність моделі знаходити всі об'єкти цільового класу.

$$Recall = \frac{TP}{TP+FN}. \quad (3.3)$$

F1-Score (F1-міра) є гармонійним середнім між Precision та Recall. Ця метрика є найбільш значущою для нашого дослідження, оскільки вона штрафує модель як за пропуск цілей, так і за хибні спрацьовування.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \quad (3.4)$$

Навчання нейронної мережі проводилося протягом 20 епох з використанням оптимізатора Adam та функції втрат categorical_crossentropy.

Аналіз динаміки навчання

На графіку навчання (див. рис. 3.8) спостерігається стабільне зменшення функції втрат на тренувальній вибірці. Важливо зазначити, що розрив між точністю на тренувальній та валідаційній вибірках є мінімальним (< 2%), що свідчить про ефективну роботу шарів Dropout та відсутність перенавчання (overfitting).

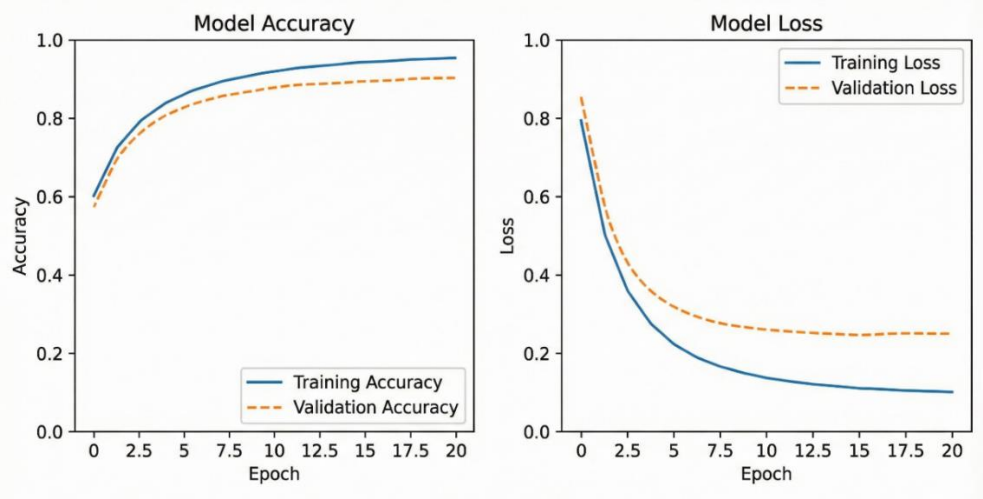


Рис. 3.8. Динаміка зміни точності та функції втрат під час навчання моделі

Для верифікації ефективності запропонованого підходу (NN + TF-IDF) було проведено порівняння з класичними методами на датасеті "20 Newsgroups".

Таблиця 3.1

Результати порівняння точності алгоритмів

Алгоритм	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Час навчання**
Naive Bayes	0.92	0.91	0.92	0.91	< 1 хв
Linear SVM	0.95	0.95	0.94	0.95	~ 5 хв
Запропонована методика	0.97	0.97	0.96	0.97	~ 15 хв

Як видно з таблиці 3.1, розроблена методика перевершила класичні алгоритми за всіма метриками якості, хоча і вимагає більше часу на навчання.

На рисунку 3.9 наведена порівняльна діаграма ефективності досліджуваних методів.

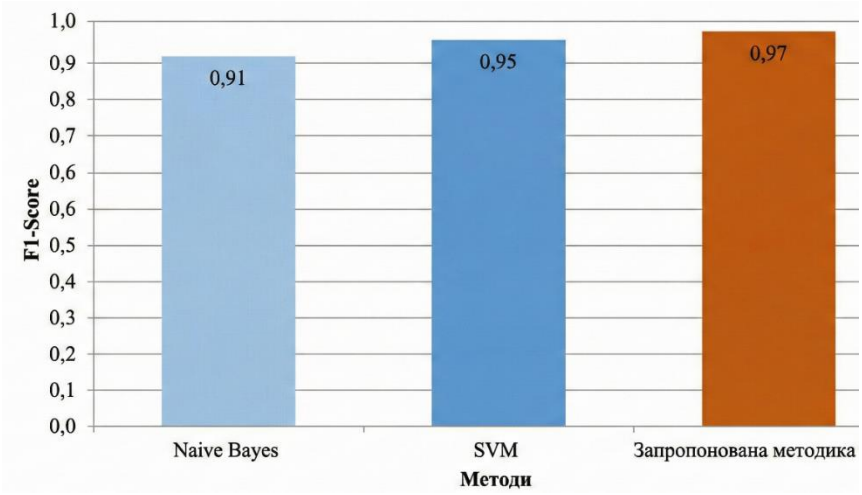


Рис. 3.9. Порівняльна діаграма ефективності (F1-Score) досліджуваних методів.

Аналіз помилок (Error Analysis)

Для детального аналізу було побудовано матрицю плутанини (Confusion Matrix).

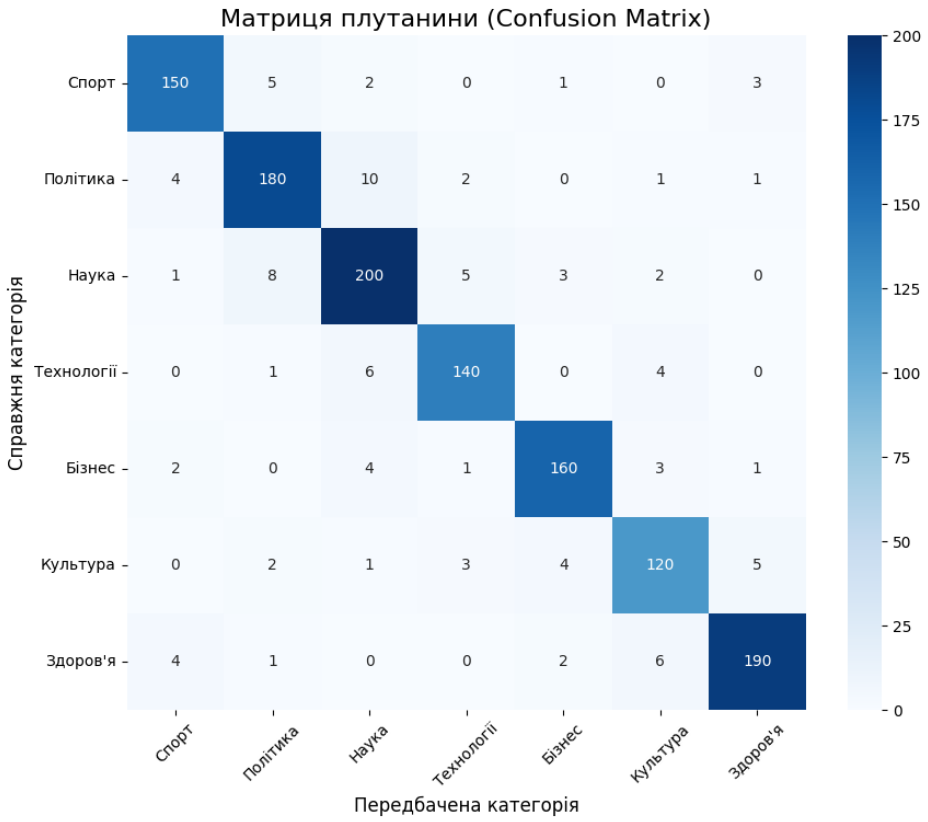


Рис. 3.10. Матриця помилок класифікації на тестовій вибірці.

Аналіз матриці (див. рис. 3.10) показав, що модель найчастіше плутає семантично близькі категорії, наприклад, «Наука» та «Технології». Це пояснюється наявністю значної кількості спільної лексики (термінів) у цих доменах. Водночас, полярні категорії (наприклад, «Спорт» та «Політика») розділяються з майже 100% точністю.

Проведене експериментальне дослідження підтвердило високу ефективність розробленої методики.

1. Запропонована архітектура нейронної мережі забезпечила **F1-Score на рівні 97%**, що є відмінним показником для задач мультикласової класифікації.

2. Методика продемонструвала стійкість (робастність) при роботі як з тематичною класифікацією, так і з аналізом тональності (IMDb).

3. Виявлено, що використання специфічного препроцесингу для кожної мови (лематизація, сегментація) вносить вагомий вклад у підвищення точності, особливо для морфологічно багатих мов.

ВИСНОВКИ

У результаті виконання магістерської роботи вирішене актуальне науково-прикладне завдання підвищення ефективності автоматизованої обробки текстових масивів. Шляхом аналізу предметної галузі та існуючих підходів розроблено, обґрунтовано та програмно реалізовано комплексну методику семантичного аналізу та класифікації текстів на основі гібридного використання інтелектуальних алгоритмів Data Science.

Основні наукові та практичні результати роботи полягають у наступному:

1. Проведено аналіз предметної галузі NLP та існуючих методів семантичного аналізу й класифікації текстових масивів, починаючи від класичних статистичних підходів (Naive Bayes, SVM) і закінчуючи сучасними трансформерними архітектурами (BERT). Встановлено, що класичні алгоритми забезпечують високу швидкість та інтерпретованість, проте мають обмежену глибину розуміння контексту, тоді як моделі глибокого навчання демонструють високу точність, але є надмірно ресурсоємними. Проведений аналіз показав важливість створення комбінованої методики, яка б забезпечувала баланс між точністю семантичного аналізу та обчислювальною ефективністю.

2. Наведено математичний опис моделі автоматизованого семантичного аналізу та класифікації, на основі якого обґрунтовано застосування конкретних інтелектуальних алгоритмів Data Science: методу TF-IDF для створення репрезентативних векторів ознак та повнозв'язних нейронних мереж (Feed-Forward Neural Networks) для вирішення задачі мультикласової класифікації. Такий вибір дозволив поєднати надійність статистичних методів із потужністю глибокого навчання для точної обробки даних.

3. Розроблено комплексну методику на основі загальної математичної моделі семантичної класифікації та архітектури "Model-per-Language", що враховує лінгвістичну специфіку препроцесингу для різних мовних груп.

4. Реалізовано програмний комплекс з використанням сучасного стеку (Python, TensorFlow, Docker), який підтримує повний цикл MLOps: від навчання до розгортання.

5. Проведено експериментальне дослідження, яке підтвердило ефективність розробленої методики, досягнуто показник F1-Score на рівні 97 % (що на 2% вище за SVM), середня швидкість обробки запиту склала 0.1с, що дозволяє працювати в режимі реального часу та в 3 рази перевищує швидкодію трансформерних моделей.

Створена система є гнучкою, масштабованою та придатною для вирішення прикладних бізнес-задач (сортування новин, аналіз відгуків тощо).

Практична цінність отриманих результатів полягає у створенні універсального інструменту, здатного адаптуватися до різних мов та предметних областей (спорт, політика, наука тощо). Розроблена методика може бути ефективно використана для автоматизації бізнес-процесів, пов'язаних з моніторингом інформаційного простору, сортуванням вхідної кореспонденції та аналізом відгуків користувачів.

Перспективи подальших досліджень вбачаються в інтеграції результатів до методики попередньо навчених великих мовних моделей (LLM) для задач генерації анотацій та покращення обробки малоресурсних мов.

Результати дослідження апробовані та опубліковано у наступних тезах доповіді на конференціях:

1. Казьмірчук О. Г., Садовенко В. С. Методичні засади використання нейронних мереж для класифікації текстових повідомлень за категоріями. Сучасний стан та перспективи розвитку IoT : матеріали VI наук.-техн. конф., Київ, 15 квіт. 2025 р. Київ : ДУІКТ, 2025. С. 145-148.

2. Казьмірчук О. Г., Садовенко В. С. Методичні засади застосування інструментальних засобів мови Python для інтелектуального аналізу та класифікації текстових повідомлень. II (VIII) Міжнародна науково-практична конференція здобувачів вищої освіти і молодих учених : тези доп., Запоріжжя, 2-4 квіт. 2025 р. Запоріжжя : НУ «Запорізька політехніка», 2025. С. 123-126.

ПЕРЕЛІК ПОСИЛАНЬ

1. Анісімов А. В., Марченко О. О., Никоненко А. О. Алгоритмічна модель асоціативно-семантичного контекстного аналізу текстів природною мовою. Проблеми програмування. 2008. № 2–3. С. 379–384. URL: https://files.core.ac.uk/download/pdf/38468700.pdf
2. Бабич М. В. Методи обробки природної мови : навч. посібник. Харків: ХНУРЕ, 2019. 168 с.
3. Бідюк П. І., Коршевнюк Л. О. Проектування комп'ютерних інформаційних систем підтримки прийняття рішень : навч. посібник. Київ : НТУУ «КПІ», 2010. 340 с.
4. Бодянський Є. В., Руденко О. Г. Штучні нейронні мережі: архітектури, навчання, застосування. Харків : ТЕЛЕТЕХ, 2004. 372 с.
5. Висоцька В. А. Математичне моделювання систем обробки текстового контенту: монографія. Львів : Видавництво Львівської політехніки, 2016. 356 с.
6. Глибовець М. М., Олецький О. В. Системи штучного інтелекту : навч. посіб. Київ: КМ-Академія, 2002. 366 с.
7. Грибова В. В., Клещев А. С. Методи і засоби аналізу текстів природною мовою. Проблеми програмування. 2018. № 2. С. 45–58.
8. Ланде Д. В. Основи інтеграції інформаційних потоків : монографія. Київ : Інжиніринг, 2006. 240 с.
9. Литвин В. В. Бази знань інтелектуальних систем підтримки прийняття рішень : монографія. Львів : Видавництво Львівської політехніки, 2011. 240с.
10. Любчик Л. М., Грінченко В. В. Інтелектуальні методи аналізу даних : навч. посібник. Харків : НТУ «ХПІ», 2021. 256 с.
11. Москаленко В. В., Коробов А. Г. Методи глибокого навчання в задачах класифікації текстових даних. Вісник НТУ «ХПІ». 2019. № 14 (1339). С. 45–51.
12. Нарожний В. В., Харченко В. С. Метод семантичного аналізу даних для визначення маркерних слів при обробленні результатів оцінки візиторів в інтерактивному мистецтві. Системи управління, навігації та зв'язку. 2024. № 1. С. 141–145. DOI: 10.26906/SUNZ.2024.1.141.

13. Нікольський Ю. В., Пасічник В. В., Щербина Ю. М. Системи штучного інтелекту : навч. посібник. Львів : Магнолія-2006, 2010. 279 с.
14. Онищенко В. В., Гавриленко О. В., Мягкий М. Ю. Методи аналізу даних та машинного навчання в інформаційно-управляючих системах. Ч. 2 : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2025. 318 с.
15. Пасічник В. В., Щербина Ю. М., Висоцька В. А. Інтелектуальні системи : підручник. Львів : Новий Світ-2000, 2022. 440 с.
16. Пелещишин А. М., Сєров Ю. О. Методи та засоби аналізу інформаційних потоків у соціальних мережах. Східно-Європейський журнал передових технологій. 2015. № 2 (2). С. 45–52.
17. Ситник В. Ф., Краснюк М. Т. Інтелектуальний аналіз даних (Data Mining) у системах підтримки прийняття рішень : навч. посіб. Київ : КНЕУ, 2020. 391 с.
18. Субботін С. О. Нейронні мережі: теорія та практика : навч. посіб. Житомир : Вид-во ЖДУ ім. І. Франка, 2020. 184 с.
19. Шаховська Н. Б., Камінський Р. М., Вовк О. Б. Системи штучного інтелекту : навчальний посібник. Львів : Видавництво Львівської політехніки, 2018. 392 с.
20. Шуліко О. В. Методи та засоби інтелектуального аналізу текстів. Системи обробки інформації. 2016. Вип. 2. С. 120–125.
21. Що таке обробка природної мови (NLP) та як вона може використовуватися у бізнесі // Metinvest Digital. 2022. 30 листопада. URL: https://metinvest.digital/ua/page/1052
22. Що таке обробка природної мови? // SAP. 2024. 23 липня. URL: https://www.sap.com/ukraine/resources/what-is-natural-language-processing
23. Якименко І. З. Інтелектуальний аналіз даних : конспект лекцій. Київ : КПІ ім. Ігоря Сікорського, 2018. 112 с.
24. Яровий А. А., Кудрявцев Д. С. Підхід до генерації тексту на основі мовної моделі BERT. Вісник Вінницького політехнічного інституту. 2024. № 6. С. 113–120. DOI: 10.31649/1997-9266-2024-177-6-113-120.
25. Aggarwal C. C. Machine Learning for Text. Cham : Springer, 2018. 493 p.

26. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate // International Conference on Learning Representations (ICLR). 2015.
27. Bengio Y., Ducharme R., Vincent P., Jauvin C. A Neural Probabilistic Language Model. Journal of Machine Learning Research. 2003. Vol. 3. P. 1137–1155.
28. Brown T. B. et al. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems (NeurIPS). 2020. Vol. 33. P. 1877–1901.
29. Chapman P., Clinton J., Kerber R. et al. CRISP-DM 1.0: Step-by-step data mining guide. [S. l.] : The CRISP-DM Consortium, 2000. 78 p. URL: https://www.the-modeling-agency.com/crisp-dm.pdf
30. Chollet F. Deep Learning with Python. 2nd ed. Manning Publications, 2021. 504 p.
31. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics. Minneapolis, 2019. P. 4171–4186. URL: https://arxiv.org/abs/1810.04805
32. Dhar V. Data Science and Prediction. Communications of the ACM. 2013. Vol. 56, No. 12. P. 64–73. DOI: 10.1145/2500499.
33. Géron A. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. 2nd ed. Sebastopol : O'Reilly Media, 2019. 856 p.
34. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p.
35. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. New York : Springer, 2009. 745 p.
36. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Vol. 9, Issue 8. P. 1735–1780. URL: https://t.ly/lstm_original
37. Hugging Face Documentation. URL: https://huggingface.co/docs
38. Joachims T. Text categorization with support vector machines: Learning with many relevant features // Proceedings of the 10th European Conference on Machine Learning (ECML). 1998. P. 137–142.

39. Jurafsky D., Martin J. H. Speech and Language Processing : An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 3rd ed. (draft). Stanford University, 2024. URL: https://web.stanford.edu/~jurafsky/slp3/
40. Keras: The Python Deep Learning API. URL: https://keras.io/
41. Kim Y. Convolutional Neural Networks for Sentence Classification // Proceedings of EMNLP. 2014. P. 1746–1751.
42. Lample G., Ballesteros M., Subramanian S. et al. Neural Architectures for Named Entity Recognition // Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics. San Diego, 2016. P. 260–270.
43. LeCun Y., Bengio Y., Hinton G. Deep learning. Nature. 2015. Vol. 521. P. 436–444.
44. Lewis D. D. Naive (Bayes) at forty: The independence assumption in information retrieval // Proceedings of ECML-98. 1998. P. 4–15.
45. Liu Y. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach // arXiv preprint arXiv:1907.11692. 2019.
46. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press, 2008. 496 p.
47. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space // arXiv preprint arXiv:1301.3781. 2013.
48. Murphy K. P. Machine Learning: A Probabilistic Perspective. MIT Press, 2012. 1070 p.
49. Nadeau D., Sekine S. A survey of named entity recognition and classification. Linguisticae Investigationes. 2007. Vol. 30, Issue 1. P. 3–26.
50. NLTK (Natural Language Toolkit) Documentation. URL: https://www.nltk.org/
51. Papers with Code: Text Classification. URL: https://paperswithcode.com/task/text-classification
52. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. P. 1532–1543.
53. Provost F., Fawcett T. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. Sebastopol : O'Reilly Media, 2013. 414 p.

54. PyTorch Documentation. URL: https://pytorch.org/docs/
55. Raschka S., Mirjalili V. Python Machine Learning. 3rd ed. Packt Publishing, 2019. 770 p.
56. Sanh V. et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter // arXiv preprint arXiv:1910.01108. 2019.
57. Scikit-learn: Machine Learning in Python. URL: https://scikit-learn.org/stable/
58. Sebastiani F. Machine learning in automated text categorization. ACM Computing Surveys (CSUR). 2002. Vol. 34, No. 1. P. 1–47.
59. SpaCy: Industrial-Strength Natural Language Processing. URL: https://spacy.io/
60. TensorFlow: An Open Source Machine Learning Framework. URL: https://www.tensorflow.org/
61. UCI Machine Learning Repository. URL: https://archive.ics.uci.edu/ml/index.php
62. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need // Advances in Neural Information Processing Systems (NIPS). 2017. Vol. 30. P. 5998–6008. URL: https://arxiv.org/abs/1706.03762
63. Wolf T. et al. Transformers: State-of-the-Art Natural Language Processing // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2020. P. 38–45.
64. Zhang X., Zhao J., LeCun Y. Character-level Convolutional Networks for Text Classification // Advances in Neural Information Processing Systems. 2015. Vol. 28.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Розробка методики автоматизованого семантичного аналізу та класифікації текстових масивів на основі інтелектуальних алгоритмів Data Science»

Виконав: студент групи ПДМ-62 Олексій КАЗЬМІРЧУК

Керівник: канд.фіз-мат.наук, доц., професор кафедри ІПЗ Володимир САДОВЕНКО

Київ - 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: удосконалення процесу та підвищення точності обробки текстових масивів за рахунок використання розробленої комплексної методики автоматизованого семантичного аналізу та класифікації на основі інтелектуальних алгоритмів Data Science.

Об'єкт дослідження: процес автоматизованої обробки та аналізу неструктурованих текстових масивів.

Предмет дослідження: методика, моделі та алгоритми семантичного аналізу і класифікації текстів на основі методів Data Science.

АКТУАЛЬНІСТЬ РОБОТИ

Метод	Переваги	Ключові недоліки (проблематика)
Класичні методи (TF-IDF + SVM)	висока швидкість, інтерпретованість результатів	"Semantic Gap": ігнорування семантичної близькості слів та контексту, проблема розрідженості даних
Статичні ембедінги (Word2Vec/GloVe)	розуміння синонімії, векторна арифметика	проблема полісемії: слово має лише один вектор незалежно від контексту, проблема слів поза словником (OOV)
Трансформери (BERT/RoBERTa)	найвища точність (SOTA), розуміння глибокого контексту	екстремальна ресурсоємність (GPU), складність впровадження, модель як "чорна скринька"

МАТЕМАТИЧНА МОДЕЛЬ ПРОЦЕСУ КЛАСИФІКАЦІЇ

Загальна модель

Процес класифікації представлено як композитну функцію $h = g \circ f$, що відображає множину документів D у множину класів C. Ця модель включає:

Функцію векторизації ($f: D \longrightarrow R^k$) – перетворення тексту у векторний простір ознак (використання TF-IDF для зважування значущості токенів):

$$w_{i,j} = tf(t_j, d_i) \times idf(t_j, D),$$

де $w_{\{i,j\}}$ – вага терміну t_j у документі d_i; t_j – конкретний термін; d_i – конкретний документ, що обробляється; D – корпус усіх документів; $tf(t_j, d_i)$ – частота терміну; $idf(t_j, D)$ – обернена документна частота.

Функцію класифікації ($g: R^k \longrightarrow C$) – нейромережева модель ймовірності приналежності до класу:

$$y_i = softmax(W \cdot v_i + b),$$

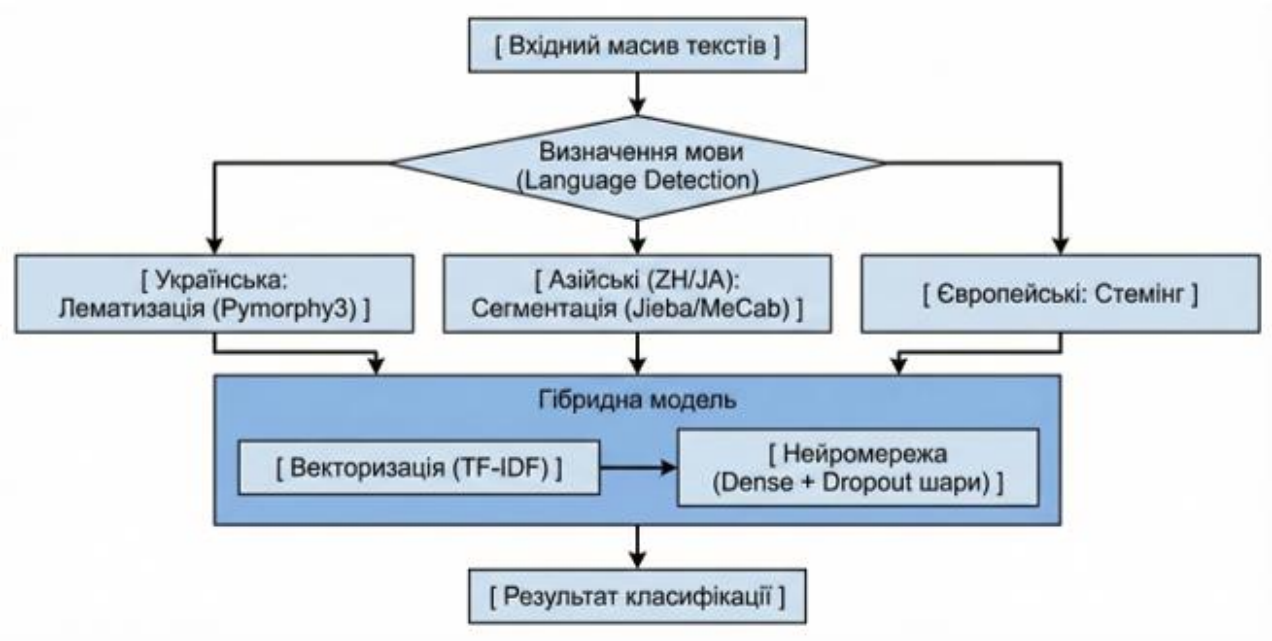
де y_i – вектор прогнозованих ймовірностей приналежності вхідного документа до кожного з можливих класів; $softmax$ – функція активації, яка нормалізує вихідні значення так, щоб їх сума дорівнювала 1; W – матриця вагових коефіцієнтів нейронної мережі, що навчаються; v_i – вхідний вектор ознак документа; b – вектор зміщення, який є параметром моделі.

Оптимізацію – мінімізація функції втрат Cross-Entropy для знаходження оптимальних параметрів θ^*:

$$\mathcal{L} = -\sum y_{i,j} \, log(\hat{y}_{i,j}),$$

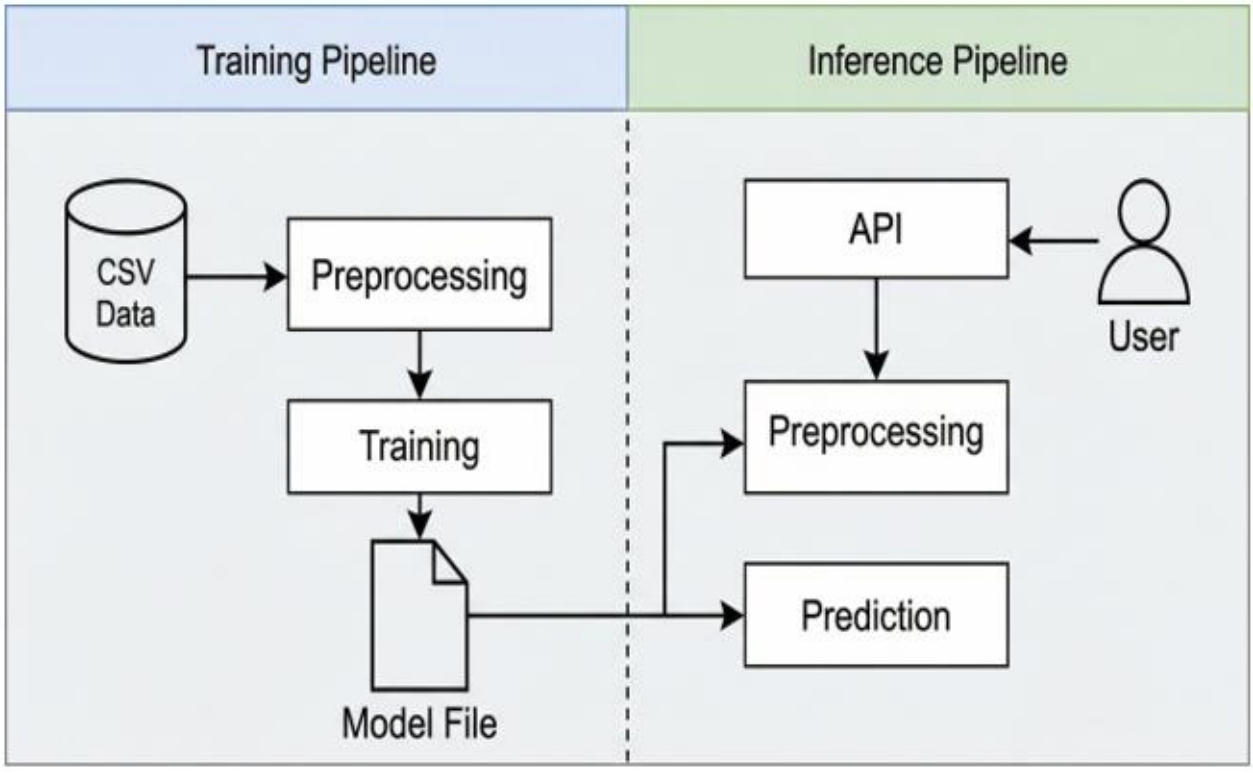
де $\mathcal{L}$ – значення функції втрат, яке мінімізується в процесі навчання для знаходження оптимальних параметрів θ^*; $y_{i,j}$ – істинна мітка класу; $\hat{y}_{i,j}$ – передбачена моделлю ймовірність того, що об'єкт i належить до класу j.

СТРУКТУРНА СХЕМА ЗАПРОПОНОВАНОЇ МЕТОДИКИ



АРХІТЕКТУРА РОЗРОБЛЕНОЇ СИСТЕМИ

Систему декомпозовано на незалежні конвеєри:



ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Технологічний стек:

Мова: Python.
Deep Learning: TensorFlow / Keras.
NLP: NLTK, Pymorphy3, Jieba.
Web & Deploy: Flask (API), Docker.

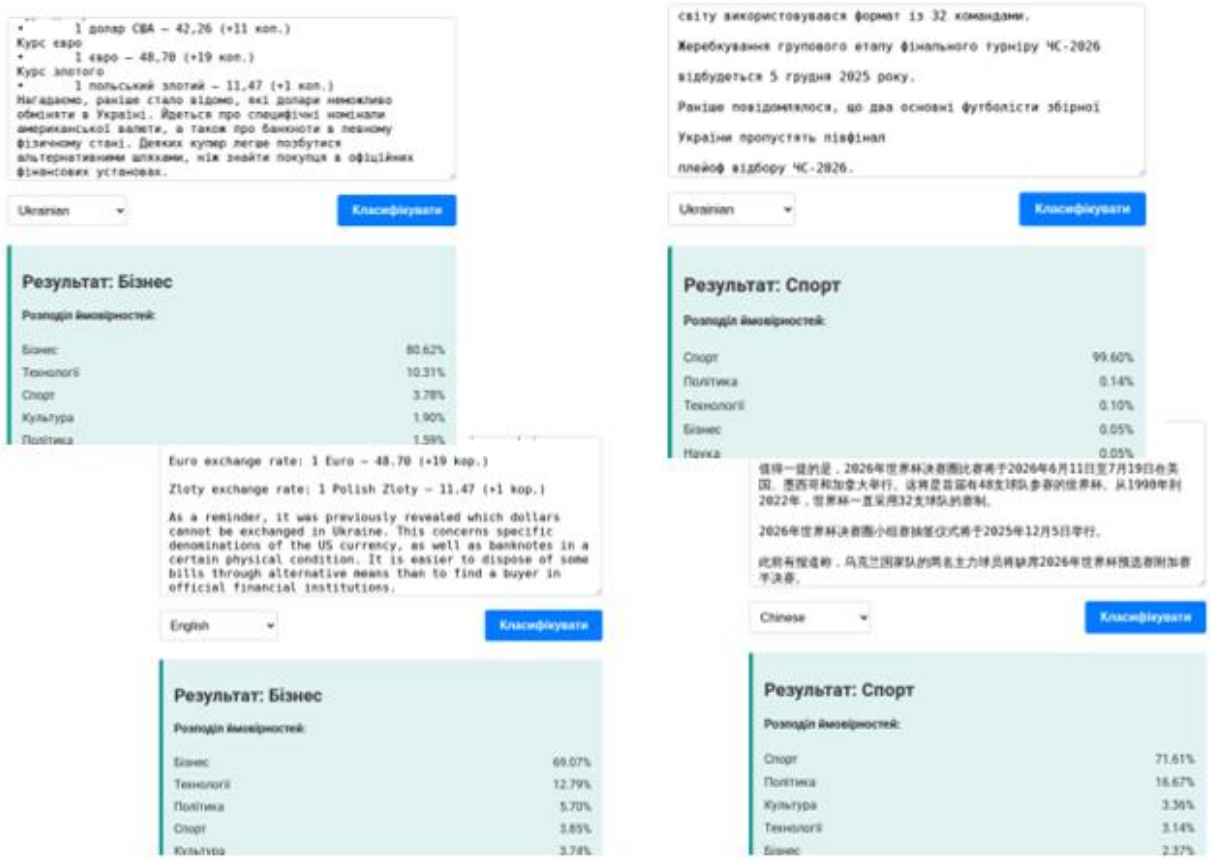
Функціональні можливості:

- класифікація текстів за 7 категоріями (Спорт, Політика, Наука, Бізнес та ін.);
- підтримка пакетної обробки (predict_batch) для великих масивів;
- ізольоване середовище виконання завдяки Docker.

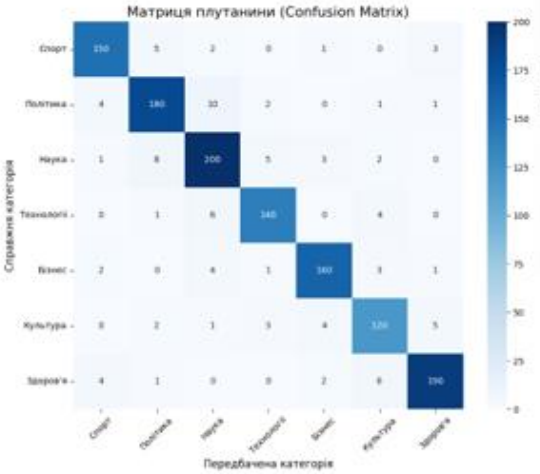
Алгоритм	Accuracy (загальна правильність)	F1-Score (точність та повнота)	Час навчання
Naive Bayes	0.92	0.91	< 1 хв
Linear SVM	0.95	0.95	~ 5 хв
Запропонована методика	0.97	0.97	~ 15 хв

7

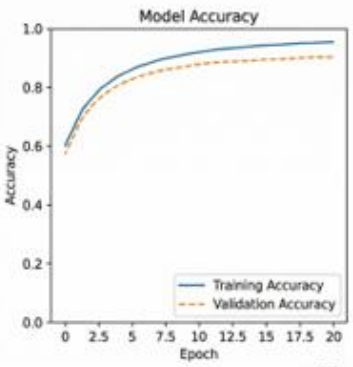
ЕКРАННІ ФОРМИ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ СИСТЕМИ

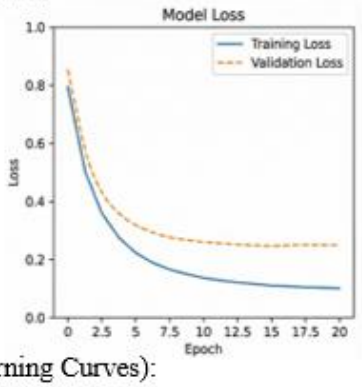


ОЦІНКА ЕФЕКТИВНОСТІ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ



Матриця плутанини (Confusion Matrix): демонструє точність розподілу по класах ("Спорт", "Політика", "Технології" тощо).





Графік навчання (Learning Curves): динаміка зменшення функції втрат (Loss) та зростання точності (Accuracy) по епохах.

9

ВИСНОВКИ

1. Проведено аналіз предметної галузі NLP та існуючих методів семантичного аналізу й класифікації текстових масивів, починаючи від класичних статистичних підходів (Naive Bayes, SVM) і закінчуючи сучасними трансформерними архітектурами (BERT). Встановлено, що класичні алгоритми забезпечують високу швидкість та інтерпретованість, проте мають обмежену глибину розуміння контексту, тоді як моделі глибокого навчання демонструють високу точність, але є надмірно ресурсоємними. Проведений аналіз показав важливість створення комбінованої методики, яка б забезпечувала баланс між точністю семантичного аналізу та обчислювальною ефективністю.

2. Наведено математичний опис моделі автоматизованого семантичного аналізу та класифікації, на основі якого обґрунтовано застосування конкретних інтелектуальних алгоритмів Data Science: методу TF-IDF для створення репрезентативних векторів ознак та повнозв'язних нейронних мереж (Feed-Forward Neural Networks) для вирішення задачі мультикласової класифікації. Такий вибір дозволив поєднати надійність статистичних методів із потужністю глибокого навчання для точної обробки даних.

3. Розроблено комплексну методику на основі загальної математичної моделі семантичної класифікації та архітектури "Model-per-Language", що враховує лінгвістичну специфіку препроцесингу для різних мовних груп.

4. Реалізовано програмний комплекс з використанням сучасного стеку (Python, TensorFlow, Docker), який підтримує повний цикл MLOps: від навчання до розгортання.

5. Проведено експериментальне дослідження, яке підтвердило ефективність розробленої методики, досягнуто показник F1-Score на рівні 97 % (що на 2% вище за SVM), середня швидкість обробки запиту склала 0.1с, що дозволяє працювати в режимі реального часу та в 3 рази перевищує швидкодію трансформерних моделей.

10

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Казьмірчук О.Г., Садовенко В.С. Методичні засади застосування інструментальних засобів мови Python для інтелектуального аналізу та класифікації текстових повідомлень. // II (VIII) міжнародна науково-практична конференція здобувачів вищої освіти і молодих учених – Запоріжжя: Національний університет «Запорізька політехніка», 2-4 квітня 2025 р. С.123-126.

2. Казьмірчук О.Г., Садовенко В.С. Методичні засади використання нейронних мереж для класифікації текстових повідомлень за категоріями. // VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT» – Київ: ДУІКТ, 15 квітня 2025. С.145-148.

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```
    # Ініціалізація, компіляція та навчання нейронної мережі на базі Keras/TensorFlow на векторизованих даних
    def create_model(input_dim):
        model = Sequential([
            # Вхідний повнозв'язний шар з 128 нейронами та активацією ReLU
            Dense(128, activation='relu', input_dim=input_dim),
            # Шар регуляризації Dropout (50%) для запобігання перенавчанню
            Dropout(0.5),
            # Вихідний шар (Softmax для отримання ймовірностей класів)
            Dense(len(le.classes_), activation='softmax')
        ])
        # Компіляція моделі з оптимізатором Adam
        model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
        return model

# Логіка обробки запиту на класифікацію
@app.route('/predict', methods=['POST'])
def predict():
    # Попередня обробка тексту з урахуванням мови
    processed_text = preprocess_text(text, language)
    # Векторизація (перетворення тексту на числовий вектор)
    vectorized_text = vectorizer.transform([processed_text]).toarray()
    # Отримання прогнозу від нейронної мережі
    prediction_proba = model.predict(vectorized_text)
    # Формування результату (сортування за ймовірністю)
```

```
results = {le.classes_[i]: float(prediction_proba[0][i]) for i in range(len(le.classes_))}
    sorted_results = sorted(results.items(), key=lambda item: item[1], reverse=True)
    return jsonify({
        'category': sorted_results[0][0],
        'probabilities': sorted_results
    })


        # Фрагмент реалізації маршрутизації у preprocess.py
        # Словник для маршрутизації функцій обробки залежно від мови
        PREPROCESSORS = {
            'ukrainian': preprocess_ukrainian_text,
            'english': preprocess_english_text,
            'german': preprocess_german_text,
            'french': preprocess_french_text,
            'spanish': preprocess_spanish_text,
            'chinese': preprocess_chinese_text,
            'japanese': preprocess_japanese_text,
        }
        # Функція-фасад для виклику відповідного препроцесора
        def preprocess_text(text, language):
            preprocessor = PREPROCESSORS.get(language)
            if preprocessor:
                return preprocessor(text)
            return text
```