

Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Рекомендовано до захисту:

протокол засідання кафедри

№ ____ від ____ . ____ .2025 р.

Завідувач кафедри ІПСіТ

_____ Олешко О.І.

(підпис)

(прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему «Методи векторизації слів при обробці текстових даних»

Захищено на засіданні ЕК № _____
протокол № _____ від ____ . ____ .2019 р.
Оцінка _____ / _____
Голова ЕК

_____ Фесенко Г.В.
(підпис) (прізвище та ініціали)

Виконав:

студент 4 курсу, групи КС- 43

Спеціальності:

122 – Комп'ютерні науки

(шифр і назва спеціальності підготовки)

Мішуров Михайло Сергійович

(прізвище, ім'я та по батькові, підпис)

Керівник д.т.н. Шматков С.І.

(ступень, звання, прізвище та ініціали, підпис)

Рецензент _____

(ступень, звання, прізвище та ініціали, підпис)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Кафедра інтелектуальних програмних систем і технологій

Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр

Напрямок підготовки комп'ютерні науки

Спеціальність 122

ЗАТВЕРДЖУЮ

Завідувач кафедри


підпис Олешко О.І.
ініціали, прізвище

“ ____ ” _____ 20 ____ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мішурова Михайла Сергійовича
(прізвище, ім'я по батькові студента)

1. Тема роботи Методи векторизації слів при обробці текстових даних

керівник роботи Шматков Сергій Ігорович, доктор наук, професор,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “16”квітня 2025 року № 4101-5/962

2. Строк подання студентом роботи 02.06.2025

3. Перелік питань, які потрібно розробити

Основи Natural Language Processing. Що таке векторизація тексту в NLP і яка вона буває: Bag of words, TF-IDF, Word2Vec, BERT та інші алгоритми. Порівняння алгоритмів векторизації тексту у системах класифікації. Донавчання моделі BERT.

4. План роботи

№ з/п	Назви етапів роботи
1	Формування мети та завдань досліджень
2	Теоретичне дослідження
3	Пошук датасету для донавчання мережі
4	Донавчання мережі BERT
5	Вибір методів оцінки ефективності векторизації
6	Порівняння інших методів векторизації зі створеною мережою
7	Оформлення результатів
8	Перевірка кваліфікаційної роботи на плагіат
9	Підготовка до захисту кваліфікаційної роботи
10	Представлення кваліфікаційної роботи до захисту

5. Дата видачі завдання 22.01.2025

Студент


підпис

Мішуров М.С.
ініціали, прізвище

Керівник роботи


підпис

Шматков С.І.
ініціали, прізвище

АНОТАЦІЯ

Звіт з кваліфікаційної роботи бакалаврського рівня на тему «Методи векторизації слів при обробці текстових даних» містить 53 с., 18 рис., 15 джерел.

Мета роботи: дослідження та порівняльний аналіз методів векторизації слів, визначення їх ефективності на прикладі задачі класифікації тексту, оцінка впливу донавчання нейромережевих методів на якість їх векторів, розробка програмного продукту у вигляді телеграм-бота для пошуку цитат відомих людей з використанням найбільш ефективного алгоритму векторизації.

Об'єкт дослідження: процеси обробки та аналізу текстових даних у контексті вирішення задач природної обробки мови.

Методи дослідження: аналіз, порівняння, розбір, опис.

Використані програмно-апаратні рішення: Python, scikit-learn, Transformers, NLTK, Aiogram, Matplotlib, TensorBoard.

Результати: проведено комплексне дослідження методів векторизації слів від класичних статистичних підходів (Bag of Words, TF-IDF) до сучасних нейромережевих архітектур (Word2Vec, BERT). Експериментально підтверджено перевагу донавченої моделі BERT для критично важливих застосувань, оптимальність Word2Vec для систем реального часу та актуальність TF-IDF для ресурсо-обмежених середовищ. Розроблено та впроваджено Telegram-бот для семантичного пошуку цитат, що демонструє високу якість результатів при адекватному часу відповіді.

Ключові слова: NLP, ВЕКТОРИЗАЦІЯ ТЕКСТУ, КЛАСИФІКАЦІЯ ТЕКСТУ, BAG OF WORDS, TF-IDF, WORD2VEC, BERT, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, СЕМАНТИЧНИЙ ПОШУК, TELEGRAM БОТ.

ANNOTATION

The report on the qualification work of the bachelor's level on the topic "Word vectorization methods in text data processing" contains 53 pages, 18 figures, 15 sources.

Objective: research and comparative analysis of word vectorization methods, determination of their effectiveness on the example of the text classification problem, assessment of the impact of additional training of neural network methods on the quality of their vectors, development of a software product in the form of a telegram bot for searching for quotes of famous people using the most effective vectorization algorithm.

Object of research: processes of processing and analysis of text data in the context of solving natural language processing problems.

Research methods: analysis, comparison, analysis, description.

Software and hardware solutions used: Python, scikit-learn, Transformers, NLTK, Aiogram, Matplotlib, TensorBoard.

Results: a comprehensive study of word vectorization methods from classical statistical approaches (Bag of Words, TF-IDF) to modern neural network architectures (Word2Vec, BERT) was conducted. The advantage of the pre-trained BERT model for critical applications, the optimality of Word2Vec for real-time systems, and the relevance of TF-IDF for resource-constrained environments have been experimentally confirmed. A Telegram bot for semantic citation search has been developed and implemented, demonstrating high quality results with adequate response time.

Keywords: NLP, TEXT VECTORIZATION, TEXT CLASSIFICATION, BAG OF WORDS, TF-IDF, WORD2VEC, BERT, MACHINE LEARNING, NEURAL NETWORKS, SEMANTIC SEARCH, TELEGRAM BOT.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	6
ВСТУП	7
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА МЕТОДИ ВЕКТОРИЗАЦІЇ ТЕКСТУ	9
1.1. Поняття обробки природної мови	9
1.2. Основні етапи попередньої обробки тексту	9
1.3 Поняття векторизації тексту	11
1.4 Bag of Words	12
1.5. TF-IDF	13
1.6. Word2Vec	14
1.7. BERT	15
1.8. Інші методи векторизації тексту	17
РОЗДІЛ 2. ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ВЕКТОРИЗАЦІЇ ТЕКСТУ	20
2.1 Використані бібліотеки та інструменти	20
2.2 Бенчмарк МТЕВ для оцінки якості векторних представлень	22
2.3 Датасет та його попередня обробка	23
2.4 Донавчання моделі BERT	26
2.5 Реалізація методів векторизації	29
2.6 Процедура класифікації	32
2.7 Аналіз якості векторизації та швидкодії	35
2.8 Практичні рекомендації щодо вибору методу векторизації	38
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ПОШУКУ ЦИТАТ НА ОСНОВІ ВЕКТОРИЗАЦІЇ WORD2VEC	39
3.1 Програмні бібліотеки та інструменти	39
3.2 Вибір методу векторизації	40
3.3 Опис датасету та предметної області	40
3.4 Реалізація системи пошуку цитат	41
3.5 Реалізація телеграм бота	44
3.6 Демонстрація функціональності	46

3.7 Переваги та обмеження реалізованої системи	48
ВИСНОВКИ.....	50
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	52

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

BERT	— Bidirectional Encoder Representations from Transformers
BoW	— Bag of Words
BPE	— Byte-Pair Encoding
CBOW	— Continuous Bag of Words
GloVe	— Global Vectors for Word Representation
GPT	— Generative pre-trained transformer
GPU	— Graphics Processing Unit
IDF	— Inverse Document Frequency
IMDb	— Internet Movie Database
MLM	— Masked Language Modeling
MTEB	— Massive Text Embedding Benchmark
NLP	— Natural Language Processing
NLTK	— Natural Language Toolkit
ORM	— Object-Relational Mapping
SBERT	— Sentence-BERT
TF	— Term Frequency
TF-IDF	— Term Frequency-Inverse Document Frequency

ВСТУП

В епоху стрімкого розвитку цифрових технологій та експоненційного зростання обсягів інформації, завдання ефективної обробки та аналізу текстових даних набуває особливої актуальності. Сучасні підходи до обробки природної мови дозволяють комп'ютерним системам розуміти, інтерпретувати та генерувати людську мову, що відкриває широкі можливості для розробки інтелектуальних систем, здатних взаємодіяти з користувачами на якісно новому рівні.

Фундаментальною проблемою в галузі NLP є необхідність представлення текстової інформації у форматі, зрозумілому для комп'ютерних алгоритмів. Векторизація слів, тобто процес перетворення текстових даних у числові вектори, є ключовим етапом, що визначає ефективність подальшої обробки та аналізу тексту. Від якості векторного представлення слів напряду залежить здатність моделей машинного навчання розпізнавати семантичні та синтаксичні особливості тексту, класифікувати документи, виявляти тематичні тренди та виконувати інші завдання, пов'язані з аналізом текстової інформації.

Протягом останнього десятиліття в галузі NLP відбувся стрімкий розвиток методів векторизації слів - від класичних підходів, таких як Bag of Words і TF-IDF, до сучасних неймережевих алгоритмів, зокрема Word2Vec і контекстуальних моделей на основі трансформерів, як-от BERT. Кожен з цих методів має свої переваги та обмеження, що зумовлює необхідність їх детального порівняльного аналізу для визначення оптимального підходу до вирішення конкретних прикладних завдань.

Актуальність роботи зумовлена зростаючою потребою в ефективних методах аналізу текстових даних у різних сферах людської діяльності - від інформаційного пошуку та систем рекомендацій до бізнес-аналітики та розробки віртуальних асистентів. Дослідження та порівняння різних алгоритмів векторизації слів дозволить визначити їх ефективність у контексті

конкретних задач NLP та розробити рекомендації щодо вибору оптимального методу векторного представлення текстової інформації.

Об'єктом дослідження цієї роботи є процеси обробки та аналізу текстових даних у контексті вирішення задач природної обробки мови.

Предметом дослідження є методи та алгоритми векторизації слів при обробці текстових даних, зокрема Bag of Words, TF-IDF, Word2Vec та BERT, їх ефективність та застосовність у задачах класифікації тексту.

Метою дипломної роботи є дослідження та порівняльний аналіз методів векторизації слів при обробці текстових даних, визначення їх ефективності на прикладі задачі класифікації тексту, оцінка впливу донавчання нейромережових методів на якість їх векторів, а також розробка програмного продукту у вигляді телеграм-бота для пошуку цитат відомих людей з використанням найбільш ефективного алгоритму векторизації.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА МЕТОДИ ВЕКТОРИЗАЦІЇ ТЕКСТУ

1.1. Поняття обробки природної мови

Обробка природної мови (Natural Language Processing, NLP) – це галузь штучного інтелекту та комп'ютерної лінгвістики, що займається взаємодією між комп'ютерами та людською (природною) мовою [1]. Основна мета NLP – навчити комп'ютери розуміти, інтерпретувати та генерувати людську мову в корисний спосіб.

Задачі NLP охоплюють широкий спектр застосувань, серед яких:

- машинний переклад (наприклад, Google Translate, DeepL);
- аналіз тональності тексту (sentiment analysis);
- класифікація тексту за тематикою або категоріями;
- відповіді на запитання (question answering);
- розпізнавання іменованих сутностей (named entity recognition);
- генерація тексту та діалогові системи;
- автоматичне реферування та узагальнення тексту;
- інформаційний пошук та екстракція [2].

Однією з фундаментальних проблем у галузі NLP є перетворення текстової інформації у формат, придатний для комп'ютерного аналізу. На відміну від людей, комп'ютери не здатні природним чином розуміти семантику тексту, тому необхідно застосовувати спеціальні підходи для представлення текстових даних у числовому форматі, який можна обробляти алгоритмами машинного навчання.

1.2. Основні етапи попередньої обробки тексту

Перед застосуванням алгоритмів векторизації, які будуть розглянуті пізніше, текст зазвичай проходить кілька етапів попередньої обробки, які суттєво впливають на якість подальшого аналізу [3].

Токенізація – це процес розбиття тексту на окремі одиниці, які називають токенами. Токенами можуть бути слова, символи або підслова. Цей етап є фундаментальним для будь-якого подальшого аналізу тексту, оскільки більшість алгоритмів NLP працюють саме з послідовностями токенів.

Існують різні підходи до токенізації:

- Токенізація за словами – текст розбивається на окремі слова, зазвичай за пробілами та знаками пунктуації.
- Токенізація за символами – текст розбивається на окремі символи.
- Токенізація за підсловами (subword tokenization) – метод, що розбиває слова на менші значущі частини. Цей підхід особливо ефективний для мов з багатим морфологічним складом та для роботи з невідомими словами. Прикладами таких алгоритмів є Byte-Pair Encoding (BPE), WordPiece та SentencePiece, які використовуються в сучасних моделях, таких як BERT та GPT [4].

Розглянемо приклад токенізації за підсловами з використанням BPE. Вхідне слово: "переклад". Результат: ["пере", "клад"].

Лематизація – це процес приведення словоформи до її базової форми (леми) з урахуванням морфологічного аналізу слова та контексту. Лема – це канонічна або словникова форма слова. Для української та інших морфологічно багатих мов лематизація є особливо важливим етапом, оскільки одне і те ж слово може мати багато різних форм. Лематизація допомагає зменшити розмірність простору ознак та об'єднати різні форми одного слова [5].

Приклади лематизації:

- "писав", "пише", "писатиме" → "писати";
- "кращий", "найкращий" → "хороший";
- "студентів", "студентові", "студентами" → "студент".

Стемінг – це спрощений процес приведення слова до його основи (стеми) шляхом відкидання афіксів за допомогою набору евристичних правил.

На відміну від лематизації, стемінг не гарантує отримання коректної лінгвістичної форми і може призвести до неіснуючих слів. Стемінг є менш точним, але більш швидким порівняно з лематизацією. Вибір між стемінгом і лематизацією залежить від конкретної задачі, вимог до точності та доступних обчислювальних ресурсів.

Приклади стемінгу:

- "писав", "пише", "писатиме" → "пис";
- "кращий", "найкращий" → "крас";
- "університетський", "університету" → "університет".

1.3 Поняття векторизації тексту

Векторизація тексту – це процес перетворення текстових даних у числові вектори, які можуть бути використані алгоритмами машинного навчання. Оскільки більшість алгоритмів машинного навчання працюють із числовими даними, векторизація є необхідним етапом для застосування цих алгоритмів до текстової інформації. Основна мета векторизації полягає в створенні таких векторних представлень слів або документів, які б зберігали семантичні та синтаксичні особливості тексту. Ідеально, слова з подібним значенням або контекстом повинні мати подібні векторні представлення в багатовимірному просторі.

Алгоритми векторизації можна умовно розділити на кілька категорій:

- Частотні методи: Bag of Words, TF-IDF, N-grams.
- Прогнозні методи (на основі нейронних мереж): Word2Vec (CBOW і Skip-gram), GloVe, FastText.
- Контекстуальні методи (на основі нейромереж-трансформерів): BERT, GPT, RoBERTa, DistilBERT, XLNet та інші.

Кожен із цих підходів має свої переваги та обмеження, які будуть детально розглянуті у наступних підрозділах.

1.4 Bag of Words

Bag of Words (BoW) – це один із найпростіших і найбільш інтуїтивно зрозумілих методів векторизації тексту [6]. Основна ідея полягає в представленні тексту як неупорядкованого набору слів, де враховується лише частота появи кожного слова, а порядок слів ігнорується. BoW часто використовується як базова модель для порівняння з більш складними алгоритмами векторизації.

Алгоритм векторизації:

- 1) Створити словник усіх унікальних слів у корпусі документів.
- 2) Для кожного документа створити вектор, розмірність якого дорівнює розміру словника.

- 3) Заповнити вектор частотами появи кожного слова в документі (або просто бінарними індикаторами наявності/відсутності слова).

Розглянемо приклад. Нехай, маємо два простих речення. Документ 1: "Я люблю машинне навчання". Документ 2: "Я люблю обробку природної мови". Спочатку формуємо словник усіх унікальних слів: ["Я", "люблю", "машинне", "навчання", "обробку", "природної", "мови"]. Тепер представляємо кожен документ у вигляді вектора частот. Документ 1: [1, 1, 1, 1, 0, 0, 0]. Документ 2: [1, 1, 0, 0, 1, 1, 1].

Переваги алгоритму:

- простота реалізації;
- інтуїтивна зрозумілість;
- висока швидкість обчислень;
- ефективність для простих задач класифікації з невеликими корпусами.

Недоліки алгоритму :

- ігнорування порядку слів, що призводить до втрати важливої семантичної інформації;
- висока розмірність векторів (дорівнює розміру словника);
- відсутність зв'язку між семантично пов'язаними словами;

- проблема розріджених даних;
- чутливість до розміру словника і частоти слів.

1.5. TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) – це статистична міра, яка оцінює важливість слова в документі відносно корпусу документів [8]. Вона поєднує два показники: частоту терміна та обернену частоту документа. Цей метод є потужним покращенням BoW, особливо для задач, де важливо виділити ключові слова, що характеризують документ.

Алгоритм векторизації:

1) Обчислити Term Frequency (TF) – частоту появи слова в документі.
 $TF(t, d) = (\text{кількість появ терміна } t \text{ в документі } d) / (\text{загальна кількість слів у документі } d).$

2) Обчислити Inverse Document Frequency (IDF) – міру унікальності слова в корпусі. $IDF(t) = \log(N / DF(t))$, де N – загальна кількість документів у корпусі, $DF(t)$ – кількість документів, що містять термін t .

3) Обчислити TF-IDF як добуток TF та IDF: $TF-IDF(t, d) = TF(t, d) \times IDF(t).$

Розглянемо той самий приклад з двома документами. Документ 1: "Я люблю машинне навчання". Документ 2: "Я люблю обробку природної мови". Обчислимо TF-IDF для слова "люблю" у першому документі. $TF(\text{"люблю"}, \text{док1}) = 1/4 = 0.25$ (слово "люблю" з'являється 1 раз серед 4 слів). $DF(\text{"люблю"}) = 2$ (слово "люблю" зустрічається в обох документах). $IDF(\text{"люблю"}) = \log(2/2) = \log(1) = 0$. Отже, $TF-IDF(\text{"люблю"}, \text{док1}) = 0.25 \times 0 = 0$. Це показує, що слово "люблю" не є дуже інформативним для розрізнення цих документів, оскільки воно з'являється в обох. Тепер обчислимо TF-IDF для слова "машинне" у першому документі. $TF(\text{"машинне"}, \text{док1}) = 1/4 = 0.25$. $DF(\text{"машинне"}) = 1$ (слово "машинне" зустрічається лише в першому документі). $IDF(\text{"машинне"}) = \log(2/1) = \log(2) \approx 0.301$. Отже, $TF-IDF(\text{"машинне"}, \text{док1}) = 0.25 \times 0.301 \approx$

0.075. У результаті TF-IDF надає вищу вагу словам, які часто зустрічаються в конкретному документі, але рідко – в інших документах корпусу.

Переваги алгоритму:

- врахування важливості слів у контексті всього корпусу;
- зниження ваги загальних слів (наприклад, сполучників, прийменників);
- підвищення ваги термінів, які є характерними для конкретного документа;
- проста інтерпретація результатів.

Недоліки алгоритму:

- як і BoW, не враховує порядок слів та контекст;
- не відображає семантичні зв'язки між словами;
- висока розмірність векторів;
- обмежена ефективність для коротких текстів.

1.6. Word2Vec

Word2Vec – це група алгоритмів для отримання векторних представлень слів, розроблена дослідниками Google на чолі з Томашем Міколовим у 2013 році [9]. На відміну від BoW і TF-IDF, Word2Vec використовує нейронні мережі для створення векторних представлень, які відображають семантичні зв'язки між словами. Word2Vec базується на дистрибутивній гіпотезі, яка стверджує, що слова, які з'являються в подібних контекстах, мають подібне значення. Це дозволяє створювати векторні представлення слів, де семантично пов'язані слова розташовуються близько одне до одного у векторному просторі. Word2Vec став проривом у галузі NLP і заклав основу для подальшого розвитку методів векторизації на основі нейронних мереж. Word2Vec пропонує дві архітектури нейронних мереж, Continuous Bag of Words (CBOW) та Skip-gram.

CBOW передбачає цільове слово на основі контексту. Для кожного слова в корпусі розглядається вікно з оточуючими словами. Нейронна мережа навчається передбачати центральне слово на основі слів контексту. Після навчання вагові коефіцієнти прихованого шару використовуються як векторні представлення слів. Skip-gram працює у зворотному напрямку порівняно з CBOW – модель передбачає контекстні слова на основі поточного слова.

Одна з найцікавіших особливостей векторів Word2Vec – можливість виконувати векторну арифметику, яка відображає семантичні зв'язки між словами. Наприклад, $\text{Vector}(\text{"король"}) - \text{Vector}(\text{"чоловік"}) + \text{Vector}(\text{"жінка"}) \approx \text{Vector}(\text{"королева"})$. Це означає, що вектори здатні захоплювати складні семантичні відношення між словами.

Переваги алгоритму :

- врахування семантичних зв'язків між словами;
- відносно невелика розмірність векторів (зазвичай 100-300);
- можливість виконувати векторну арифметику;
- ефективність для пошуку аналогій і подібних слів;
- хороша узагальнююча здатність.

Недоліки алгоритму:

- потреба у великому корпусі для навчання;
- відсутність контекстуалізації (одне слово має одне векторне представлення незалежно від контексту);
- неможливість обробки невідомих слів;
- обчислювальна складність навчання.

1.7. BERT

Bidirectional Encoder Representations from Transformers (BERT)– це модель представлення мови, розроблена дослідниками Google у 2018 році [10]. На відміну від Word2Vec, BERT створює контекстуалізовані векторні

представлення, де одне і те ж слово може мати різні вектори залежно від контексту, в якому воно використовується.

BERT базується на архітектурі Transformer, яка використовує механізм уваги для обробки послідовностей [7]. Архітектура Transformer має декілька ключових переваг порівняно з іншими підходами. По-перше, паралелізація обчислень значно покращує швидкість навчання та інференсу, оскільки всі позиції послідовності можуть оброблятися одночасно. По-друге, механізм уваги дозволяє моделі безпосередньо моделювати залежності між будь-якими двома позиціями у послідовності, незалежно від їх відстані. По-третє, архітектура демонструє високу ефективність у захопленні довгострокових залежностей без проблем зникаючого градієнта, характерних для RNN. Класична модель Transformer складається з двох основних компонентів: кодувальника (encoder) та декодувальника (decoder). Кодувальник відповідає за перетворення вхідної послідовності в контекстне представлення, тоді як декодувальник генерує вихідну послідовність на основі цього представлення. BERT використовує лише енкодерну частину архітектури Transformer, адаптуючи її для задач розуміння мови. Ключова інновація BERT полягає у двонаправленому навчанні, яке дозволяє моделі враховувати контекст з обох сторін слова. На його основі розроблено численні модифікації та покращення, такі як RoBERTa, DistilBERT, ALBERT та інші.

BERT складається з кількох шарів трансформерів, кожен з яких має підшари самоуваги (self-attention) та повнозв'язні нейронні мережі. Залежно від розміру, розрізняють:

- BERT-base: 12 шарів, 12 механізмів уваги, 768 прихованих нейронів (110 млн параметрів). Саме ця версія буде використана під час подальшого порівняння алгоритмів векторизації.

- BERT-large: 24 шари, 16 механізмів уваги, 1024 приховані нейрони (340 млн параметрів).

BERT використовує два завдання для передтренування:

– Masked Language Model (MLM): частина слів у вхідному тексті маскується (замінюється на спеціальний токен [MASK]), і модель навчається передбачати ці слова на основі контексту.

– Next Sentence Prediction (NSP): модель навчається передбачати, чи є два речення послідовними у тексті.

Для отримання векторних представлень тексту за допомогою BERT використовується підхід CLS Token Embedding. BERT додає спеціальний токен [CLS] на початку кожного тексту. Вектор цього токена на виході останнього шару можна використовувати як представлення всього тексту.

Переваги алгоритму:

- контекстуалізовані векторні представлення (одне слово може мати різні вектори у різних контекстах);
- здатність враховувати семантику та синтаксис на рівні речень;
- висока ефективність у багатьох задачах NLP;
- можливість роботи з невідомими словами завдяки токенізації на рівні підслів;
- передтренована модель може бути донавчена для конкретних задач.

Недоліки алгоритму:

- висока обчислювальна складність, що призводить до потреби в потужних GPU (Graphics Processing Unit) для навчання та інференсу;
- обмеження на довжину вхідного тексту (зазвичай 512 токенів);
- велика кількість параметрів (потреба у значних обсягах пам'яті);
- складність інтерпретації отриманих векторів.

1.8. Інші методи векторизації тексту

Окрім детально розглянутих в цьому розділі методів векторизації, в галузі NLP існує ряд інших підходів, які заслуговують на згадку, хоча і не будуть досліджуватися в рамках цієї роботи. Ці методи можуть бути

корисними для певних специфічних задач або як альтернативи основним підходам.

GloVe (Global Vectors for Word Representation) – це алгоритм векторизації слів, розроблений дослідниками Стенфордського університету у 2014 році. Основна ідея полягає у наступному. GloVe будує матрицю спільної появи слів у корпусі та використовує факторизацію цієї матриці для отримання векторних представлень. Алгоритм намагається мінімізувати різницю між скалярним добутком векторів слів та логарифмом ймовірності їх спільної появи. Переваги: ефективне використання статистичної інформації корпусу, хорошу ефективність на задачах аналогій слів.

FastText – це розширення Word2Vec, розроблене дослідниками Facebook у 2016 році. Основна ідея в тому, що замість представлення слова як атомарної одиниці, FastText розкладає кожне слово на символні n-грами (підрядки фіксованої довжини) і створює векторні представлення для цих n-грам. Вектор слова обчислюється як сума векторів його n-грам. Переваги: здатність створювати вектори для невідомих слів, краща обробка морфологічно багатих мов, ефективність для рідкісних слів.

Doc2Vec – це розширення Word2Vec для створення векторних представлень цілих документів, а не лише окремих слів. Doc2Vec додає до архітектури Word2Vec додатковий "документний вектор", який навчається разом з векторами слів. Цей підхід дозволяє отримувати фіксовані векторні представлення для документів різної довжини. Переваги: прямі векторні представлення документів, врахування порядку слів у документі, ефективність для задач класифікації документів.

Sentence-BERT (SBERT) – це модифікація BERT, спеціально оптимізована для створення семантично значущих векторних представлень речень, розроблена у 2019 році. SBERT модифікує BERT за допомогою сіамських та триплетних мережевих структур для створення векторних представлень речень, які можна порівнювати за допомогою косинусної подібності. Переваги: ефективні обчислення подібності речень, можливість

кластеризації та семантичного пошуку, значно швидша за стандартний BERT для задач порівняння речень.

Серія моделей GPT (Generative Pre-trained Transformer) – це аутогенеративні мовні моделі, розроблені OpenAI, які також можуть використовуватися для векторизації тексту. GPT використовує архітектуру трансформер з односпрямованою увагою (unidirectional attention) для моделювання послідовності тексту. Вектори прихованих станів можуть використовуватися як представлення слів або речень. Переваги: потужні генеративні можливості, ефективність на задачах генерації тексту, можливість few-shot навчання.

РОЗДІЛ 2. ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ВЕКТОРИЗАЦІЇ ТЕКСТУ

Основною метою даного розділу є проведення порівняльного аналізу ефективності різних методів векторизації текстових даних на прикладі задачі класифікації текстів. У попередньому розділі було детально розглянуто теоретичні основи алгоритмів BoW, TF-IDF, Word2Vec та BERT. Тепер необхідно оцінити їх практичну ефективність у реальних умовах застосування. Експериментальне дослідження спрямоване на вирішення наступних завдань:

- порівняння якості векторних представлень, що генеруються різними алгоритмами векторизації;
- оцінка впливу розміру навчальної вибірки на ефективність кожного методу;
- дослідження ефекту донавчання моделі BERT на цільовому датасеті;
- аналіз швидкості роботи різних алгоритмів векторизації;
- визначення оптимального підходу для конкретної задачі класифікації.

Для забезпечення об'єктивності порівняння було обрано задачу бінарної класифікації настроїв відгуків на фільми з датасету IMDb (Internet Movie Database), що є стандартною задачею в галузі обробки природної мови [11]. Ця задача дозволяє оцінити здатність різних методів векторизації захоплювати семантичну інформацію, необхідну для розуміння емоційного забарвлення тексту.

2.1 Використані бібліотеки та інструменти

Для реалізації експерименту використовувалися мова програмування Python та комплекс сучасних бібліотек машинного навчання та обробки природної мови. Для управління залежностями використовується стандартний менеджер пакетів `pip` з файлом `requirements.txt` і віртуальне середовище для ізоляції залежностей проекту. Далі наведено перелік бібліотек.

Transformers (Hugging Face) - основна бібліотека для роботи з попередньо натренованими трансформерними моделями. Використовувалася для завантаження та роботи з BERT-моделями, включаючи BertTokenizer для токенизації, BertModel для отримання векторних представлень та BertForMaskedLM для донавчання моделі на задачі Masked Language Modeling.

Datasets (Hugging Face) - бібліотека для ефективного завантаження та обробки великих датасетів, включаючи IMDb Movie Reviews. Забезпечує оптимізовані операції над даними та інтеграцію з екосистемою Hugging Face.

Scikit-learn - класична бібліотека машинного навчання, що використовувалася для реалізації традиційних методів векторизації (CountVectorizer, TfidfVectorizer), логістичної регресії як базового класифікатора та метрик оцінювання якості (classification_report, f1_score).

NLTK (Natural Language Toolkit) - комплексна бібліотека для обробки природної мови, що використовувалася для токенизації тексту та завантаження списку стоп-слів англійської мови при роботі з Word2Vec.

Gensim - бібліотека для тематичного моделювання та обробки документів, що забезпечила доступ до попередньо натренованої моделі Word2Vec (Google News 300-dimensional vectors).

NumPy - фундаментальна бібліотека для наукових обчислень, що використовувалася для ефективних операцій з масивами.

Matplotlib - основна бібліотека для створення графіків та візуалізації результатів експерименту, включаючи порівняльні графіки F1-score для різних методів векторизації.

TensorBoard - інструмент візуалізації процесу навчання моделей, що використовувався для моніторингу функції втрат під час донавчання BERT. Забезпечив можливість відстеження конвергенції моделі та виявлення потенційного перенавчання.

Всі обчислення проводилися на GPU з CUDA для забезпечення прийнятної швидкості роботи з великими нейронними мережами, а саме при роботі з BERT моделями. Повна конфігурація системи наведена в таблиці 2.1.

Таблиця 2.1 – Конфігурація системи

Назва параметру	Значення
Версія Python	3.12.7
Операційна система	Windows 11
Оперативна пам'ять	32 Гб, DDR4, 2400 МГц
Процесор	Intel Core i7-10850H, 6 ядер, 12 потоків, 2700 - 5100 MHz
Відеокарта	NVIDIA Quadro T1000, 768 CUDA ядер, 1395 – 1455 МГц
Відеопам'ять	4GB, GDDR5, 8000 МГц

2.2 Бенчмарк МТЕВ для оцінки якості векторних представлень

Massive Text Embedding Benchmark (МТЕВ) представляє собою комплексну систему оцінювання якості текстових векторних представлень, розроблену для стандартизації процесу порівняння різних методів векторизації [12]. Цей бенчмарк об'єднує 58 різних датасетів та охоплює вісім основних категорій завдань обробки природної мови:

- Класифікація (Classification) - включає задачі визначення категорії або мітки для текстового документа.
- Кластеризація (Clustering) - групування семантично подібних текстів без попереднього навчання.
- Парне класифікування (Pair Classification) - визначення відносин між парами текстів.
- Повторне ранжування (Reranking) - переупорядкування результатів пошуку відповідно до релевантності.
- Пошук інформації (Retrieval) - знаходження релевантних документів за запитом.
- Семантична подібність тексту (STS - Semantic Textual Similarity) - оцінка ступеня семантичної близькості між текстами.

- Підсумовування (Summarization) - оцінка якості автоматично згенерованих резюме.
- Класифікація біграм (BitextMining) - ідентифікація паралельних речень у білінгвальних корпусах.

Повний перелік завдань та датасетів МТЕВ зображено на рисунку 2.1.

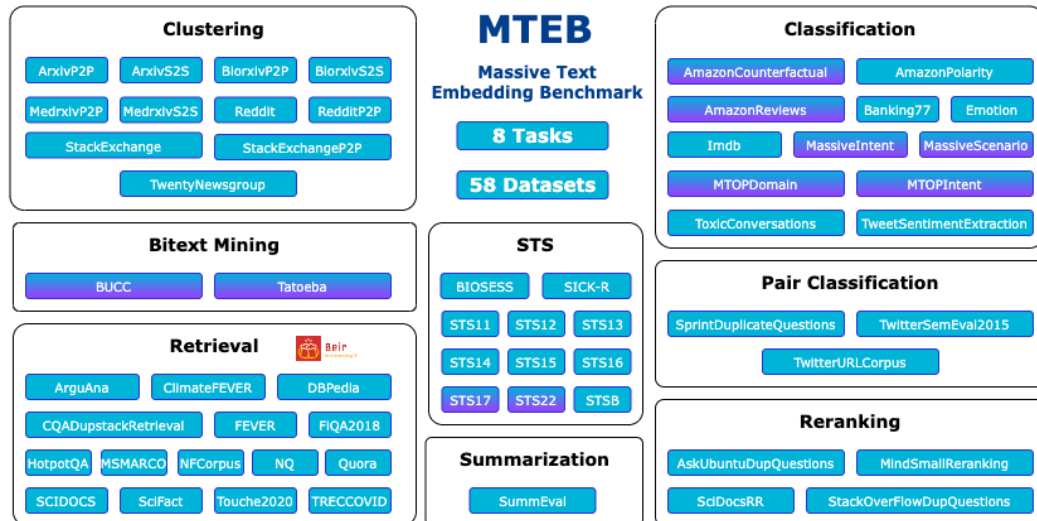


Рисунок 2.1 – Огляд завдань і наборів даних у МТЕВ. Багатомовні датасети позначені фіолетовим кольором

Використання стандартизованих бенчмарків, таких як МТЕВ, дозволяє забезпечити об'єктивність та відтворюваність результатів досліджень у галузі обробки природної мови. Це особливо важливо при порівнянні різних методів векторизації, оскільки дозволяє уникнути упередженості у виборі тестових даних та метрик оцінювання.

2.3 Датасет та його попередня обробка

З бенчмарку МТЕВ для нашого дослідження була обрана задача класифікації тексту з використанням датасету IMDb Movie Reviews. Датасет складається з 25,000 рецензій для навчання та 25,000 для тестування. Кожна рецензія має бінарну мітку: 0 для негативних відгуків (оцінка 1-4 зірки) та 1 для позитивних (оцінка 7-10 зірок). Рецензії з оцінкою 5-6 зірок не включені в

датасет для забезпечення чіткого розділення класів. Вибір датасету обумовлений кількома факторами:

- Репрезентативність задачі. Класифікація настроїв є однією з найпоширеніших задач у практичних застосуваннях NLP, від аналізу соціальних мереж до оцінки відгуків клієнтів.

- Якість датасету. IMDb містить 50,000 ретельно розмічених відгуків з рівномірним розподілом позитивних та негативних прикладів, що забезпечує збалансованість тестування.

- Варіативність тексту. Відгуки на фільми характеризуються широким діапазоном стилів, довжини та складності, що дозволяє оцінити робастність методів векторизації.

- Семантична складність. Задача вимагає розуміння не лише окремих слів, але й контексту, іронії та складних емоційних конструкцій.

Для забезпечення якості вхідних даних було застосовано процедуру попередньої обробки тексту, що включає:

- приведення тексту до нижнього регістру;
- видалення HTML-тегів;
- видалення спеціальних символів, залишаючи лише літери, цифри та базову пунктуацію;
- нормалізація пробілів для усунення зайвих пропусків.

Такий підхід до попередньої обробки забезпечує консистентність вхідних даних для всіх алгоритмів векторизації та зменшує вплив шуму на якість отриманих представлень. Реалізація використовує модуль `re` (regular expression) і зображена на рисунку 2.2.



```
def preprocess_text(text): 5 usages  ⌵ mackaronina
    text = text.lower()
    # Remove HTML tags
    text = re.sub( pattern: r"<.*?>", repl: "", text)
    # Remove special characters
    text = re.sub( pattern: r"[^a-zA-Z0-9.,!? ]", repl: "", text)
    # Remove extra spaces
    text = re.sub( pattern: r"\s+", repl: " ", text).strip()
    return text
```

Рисунок 2.2 – Функція preprocess_text для попередньої обробки тексту

Для об'єктивного порівняння методів векторизації було реалізовано кілька стратегій формування навчальних та тестових вибірок із датасету. Для тренування класифікатора використовувалися різні розміри навчальної і тестової вибірок, які будуть описані далі. Для донавчання BERT використовувалася значно більша вибірка - 15000 зразків для навчання та 2000 для валідації. Це обумовлено специфікою навчання великих мовних моделей, які потребують значних обсягів даних для ефективного донавчання. Всі вибірки формувалися з використанням фіксованого seed=1 для забезпечення відтворюваності результатів. Дані перемішувалися перед відбором для уникнення систематичних зміщень. Код класу ImdbDataset, що реалізує даний функціонал, наведено на рисунку 2.3.

```

class ImdbDataset:
    def __init__(self):
        self.dataset = load_dataset("imdb")
        self.labels = ["neg", "pos"]

    def split_train_test_classifier(self, train_size, test_size):
        train_sample = self.dataset["train"].shuffle(seed=SEED).select(range(train_size))
        test_sample = self.dataset["test"].shuffle(seed=SEED).select(range(test_size))
        train_texts = [preprocess_text(text) for text in train_sample["text"]]
        test_texts = [preprocess_text(text) for text in test_sample["text"]]
        y_train = train_sample["label"]
        y_test = test_sample["label"]
        return train_texts, test_texts, y_train, y_test

    def split_train_test_bert(self, train_size, test_size, tokenize):
        dataset = self.dataset.remove_columns("label")

        def preprocess(example):
            example["text"] = preprocess_text(example["text"])

        dataset = dataset.map(preprocess)
        dataset = dataset.map(tokenize, batched=True)
        train_sample = dataset["train"].shuffle(seed=SEED).select(range(train_size))
        test_sample = dataset["test"].shuffle(seed=SEED).select(range(test_size))
        return train_sample, test_sample

```

Рисунок 2.3 – Клас ImdbDataset

2.4 Донавчання моделі BERT

Fine-tuning (донавчання) є ключовою стратегією в сучасному машинному навчанні, що дозволяє адаптувати попередньо навчені моделі до специфічних завдань або доменів. Цей підхід базується на принципі transfer learning, де знання, отримані моделлю під час навчання на великих загальних корпусах, переносяться та уточнюються на цільових даних. У контексті мовних моделей типу BERT fine-tuning передбачає продовження навчання попередньо тренованої моделі на новому датасеті з використанням значно меншої швидкості навчання. Це дозволяє моделі зберегти загальні мовні патерни, засвоєні під час pre-training, водночас адаптуючи їх до специфіки нового домену або завдання.

Основні переваги fine-tuning включають:

- Економія обчислювальних ресурсів - не потрібно тренувати модель з нуля.
- Використання попередніх знань - модель вже володіє базовим розумінням мови.

- Швидка адаптація - потрібно значно менше епох для досягнення хороших результатів.

- Ефективність на малих датасетах - можна досягти високої якості навіть з обмеженою кількістю цільових даних.

Для дослідження впливу донавчання на якість векторних представлень була реалізована процедура *fine-tuning* попередньо навченої моделі BERT на цільовому датасеті IMDb. Використовувалася стратегія Masked Language Modeling (MLM), яка є стандартним підходом для донавчання BERT на нових доменах без необхідності розмічених даних. Процес донавчання базувався на моделі "bert-base-uncased". Ця модель була попередньо навчена на великих корпусах англomовного тексту і демонструє високу якість на широкому спектрі NLP завдань.

Конфігурація тренування була оптимізована для досягнення балансу між якістю результату та ефективністю обчислень:

- розмір батчу 4 зразки на пристрій (обмежено пам'яттю GPU в 4 Гб);
- швидкість навчання $2e-5$ (стандартне значення для донавчання BERT);
- кількість епох 5 (достатньо для конвергенції на цільовому домені);
- вимірювання та збереження кожні 1000 кроків;
- MLM ймовірність 0.15 (15% токенів маскуються для предиктування).

Стратегія оцінювання включала регулярне тестування на валідаційній вибірці з автоматичним збереженням найкращої моделі за метрикою *evaluation loss*. Повний код реалізації донавчання зі вказанням решти параметрів показано на рисунку 2.4

```
def save_logs(trainer): 1 usage 2 mackaronina
    with open("./train_results/log.txt", "w", encoding="utf-8") as log_file:
        for line in trainer.state.log_history:
            log_file.write(f"{line}\n")

def main(): 3 mackaronina *
    dataset = ImdbDataset()
    tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
    model = BertForMaskedLM.from_pretrained("bert-base-uncased").to(DEVICE)

    def tokenize(example): 4 mackaronina
        return tokenizer(example["text"], padding="max_length", truncation=True).to(DEVICE)

    train_sample, test_sample = dataset.split_train_test_bert(BERT_TRAIN_SIZE, TEST_SIZE, tokenize)
    data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=True, mlm_probability=0.15)
    training_args = TrainingArguments(output_dir="./train_results/trainer_output", evaluation_strategy="steps",
                                     eval_steps=1000, logging_strategy="steps", logging_steps=1000,
                                     save_strategy="steps", save_steps=1000, save_total_limit=5, learning_rate=2e-5,
                                     num_train_epochs=5, per_device_train_batch_size=4, per_device_eval_batch_size=4,
                                     seed=SEED, load_best_model_at_end=True, report_to="tensorboard")
    trainer = Trainer(model=model, args=training_args, data_collator=data_collator, train_dataset=train_sample,
                     eval_dataset=test_sample)
    trainer.train()
    save_logs(trainer)
    model.save_pretrained("./train_results/model")
    tokenizer.save_pretrained("./train_results/model")
```

Рисунок 2.4 – Код реалізації донавчання моделі BERT на задачі MLM на датасеті IMDb

Для візуалізації результатів процесу навчання використано TensorBoard. В результаті отримано графіки, які зображено на рисунку 2.5.

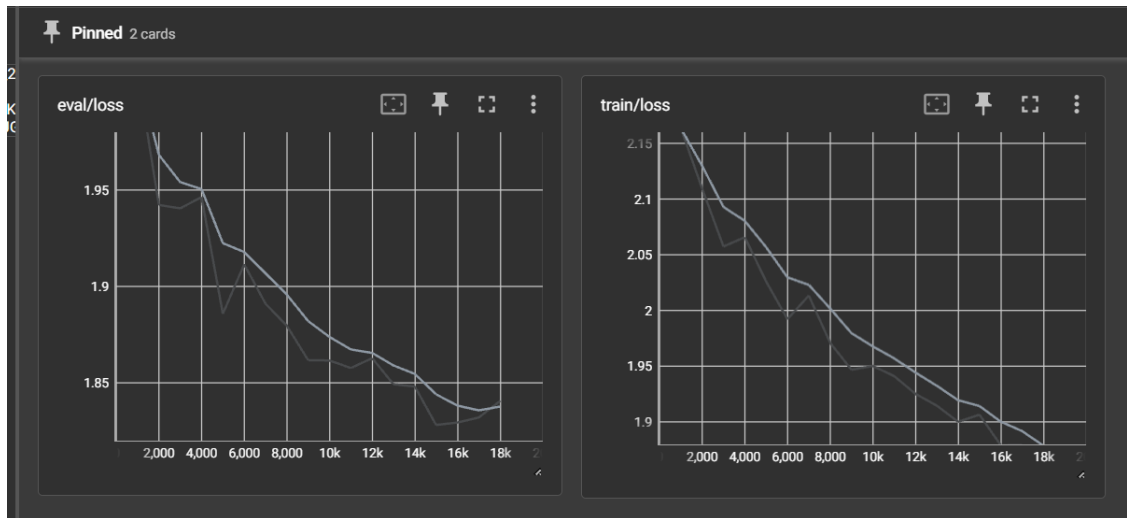


Рисунок 2.5 – Графіки eval/loss та train/loss, візуалізовані за допомогою Tensorboard

За отриманими даними, процес донавчання продемонстрував стабільну конвергенцію. Як видно з графіків втрат, навчальна втрата (train/loss)

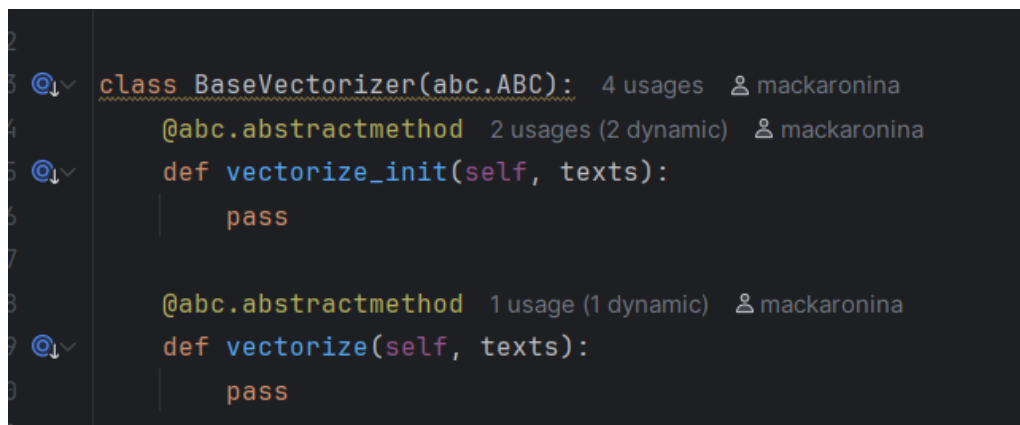
демонструє постійне зниження з початкового значення близько 2.15 до фінального значення приблизно 1.88. Валідаційна втрата (eval/loss) також показує стійке зниження з 1.97 до 1.83, що свідчить про відсутність перенавчання та ефективне засвоєння патернів цільового домену. Плавний характер кривих навчання вказує на стабільність процесу тренування та правильний вибір гіперпараметрів. Відсутність різких коливань або плато говорить про те, що модель ефективно адаптується до специфіки кінорецензій IMDb. Фінальна модель була збережена разом з токенизатором для подальшого використання в експериментах порівняння якості векторизації.

2.5 Реалізація методів векторизації

Для забезпечення справедливого порівняння різних методів векторизації була розроблена уніфікована архітектура експерименту. Всі методи векторизації реалізовано через спільний інтерфейс BaseVectorizer з двома ключовими методами:

- `vectorize_init(texts)` - початкова векторизація зі створенням словника для BoW та TF-IDF, для інших алгоритмів аналогічна `vectorize_text`;
- `vectorize_text(texts)` - векторизація нових текстів з використанням раніше створеного словника.

Такий підхід забезпечує консистентність процедури тестування і виключає можливість витоку інформації з тестової вибірки в процес навчання векторизаторів. Код класу BaseVectorizer наведено на рисунку 2.6.



```

class BaseVectorizer(abc.ABC):
    @abc.abstractmethod
    def vectorize_init(self, texts):
        pass

    @abc.abstractmethod
    def vectorize(self, texts):
        pass

```

Рисунок 2.6 – Абстрактний клас BaseVectorizer

Метод BoW реалізовано через клас `CountVectorizer` з бібліотеки `scikit-learn` [13]. `CountVectorizer` автоматично виконує токенизацію, видаляє знаки пунктуації та створює розріджену матрицю документ-терм. Результуюча матриця має розмірність `[n_documents, n_features]`, де `n_features` - розмір словника. Основною перевагою такої реалізації є її ефективність для розріджених даних та простота інтерпретації. TF-IDF реалізовано через `TfidfVectorizer`, який розширює функціональність `CountVectorizer` додаванням зважування термінів. Код реалізацій BoW та TF-IDF наведено на рисунку 2.7.

```

class BagOfWords(BaseVectorizer):
    def __init__(self):
        self.vectorizer = CountVectorizer()

    def vectorize_init(self, texts):
        return self.vectorizer.fit_transform(texts).toarray()

    def vectorize(self, texts):
        return self.vectorizer.transform(texts).toarray()

class TFIDF(BaseVectorizer):
    def __init__(self):
        self.vectorizer = TfidfVectorizer()

    def vectorize_init(self, texts):
        return self.vectorizer.fit_transform(texts).toarray()

    def vectorize(self, texts):
        return self.vectorizer.transform(texts).toarray()

```

Рисунок 2.7 – Класи `BagOfWords` і `TFIDF`

`Word2Vec` реалізовано з використанням попередньо навченої моделі `Google News`, що містить 300-вимірні вектори для 3 мільйонів слів і фраз. Алгоритм векторизації документа включає такі етапи:

- 1) Токенізація - розбиття тексту на слова за допомогою `NLTK`.
- 2) Фільтрація - видалення стоп-слів та слів, відсутніх у словнику моделі.
- 3) Отримання векторів - для кожного валідного слова генерується 300-вимірний вектор.

4) Усереднення - обчислюється середнє арифметичне всіх векторів слів документа.

Код реалізації Word2Vec наведено на рисунку 2.8.

```
class Word2Vec(BaseVectorizer): 3 usages 2 mackaronina
    def __init__(self): 2 mackaronina
        self.wv = gensim.downloader.load("word2vec-google-news-300")
        nltk.download("stopwords")
        self.stop_words = set(stopwords.words("english"))

    def vectorize(self, texts): 3 usages (1 dynamic) 2 mackaronina
        vectors = []
        for text in texts:
            words = nltk.word_tokenize(text)
            word_vectors = [self.wv[word] for word in words if word in self.wv and word not in self.stop_words]
            vectors.append(np.mean(word_vectors, axis=0))
        return np.array(vectors)

    def vectorize_init(self, texts): 3 usages (2 dynamic) 2 mackaronina
        return self.vectorize(texts)
```

Рисунок 2.8 – Клас Word2Vec

BERT реалізовано для двох варіантів: базової попередньо навченої моделі та донавченої на цільовому датасеті. Процес векторизації BERT включає такі етапи:

1) Токенізація - перетворення тексту в послдовність токенів з додаванням спеціальних токенів [CLS] та [SEP].

2) Кодування позицій - додавання інформації про позицію кожного токена в послдовності.

3) Padding та Truncation - приведення всіх послдовностей до однакової довжини (512 токенів максимум).

4) Forward pass - пропуск через 12 шарів трансформера з механізмами attention.

5) Екстракція представлення - використання векторного представлення токена [CLS] як представлення всього документа.

Ключові особливості реалізації:

– тексти обробляються групами по 64 для оптимізації використання відеопам'яті;

– torch.no_grad() запобігає обчисленню градієнтів під час інференсу;

- тензори переміщуються на GPU для прискорення обчислень;
- результати переносяться назад на CPU для звільнення відеопам'яті.

Код реалізації BERT наведено на рисунку 2.9.

```
class BERT(BaseVectorizer): 3 usages 2 mackaronina
    def __init__(self, model_name): 2 mackaronina
        self.tokenizer = BertTokenizer.from_pretrained(model_name)
        self.model = BertModel.from_pretrained(model_name).to(DEVICE)

    def vectorize(self, texts, batch_size=64): 2 usages (1 dynamic) 2 mackaronina
        predictions = []
        for i in range(0, len(texts), batch_size):
            batch_texts = texts[i:i + batch_size]
            inputs = self.tokenizer(batch_texts, return_tensors='pt', padding=True, truncation=True).to(DEVICE)
            with torch.no_grad():
                outputs = self.model(**inputs)
                batch_predictions = outputs.last_hidden_state[:, 0, :].cpu().numpy()
            predictions.extend(batch_predictions)
        return np.array(predictions)

    def vectorize_init(self, texts): 2 usages (2 dynamic) 2 mackaronina
        return self.vectorize(texts)
```

Рисунок 2.9 – Клас BERT

2.6 Процедура класифікації

Для всіх методів векторизації використовувався однаковий класифікатор - логістична регресія з регуляризацією $C=5$ та максимальною кількістю ітерацій 1000 [14]. Логістична регресія є лінійним алгоритмом класифікації, що використовує логістичну (сигмоїдну) функцію для перетворення лінійної комбінації ознак у ймовірність належності до класу.

Вибір логістичної регресії обумовлений:

- простотою та інтерпретованістю, лінійна природа моделі дозволяє легко аналізувати вплив окремих ознак;
- ефективністю на високовимірних розріджених даних, алгоритм добре працює з великою кількістю ознак, характерною для текстових даних;
- стабільністю результатів на різних типах векторних представлень;
- швидкістю навчання та передбачення, що критично важливо при порівнянні методів векторизації;
- відсутністю власних репрезентативних здібностей, що дозволяє чисто оцінити якість векторних представлень без впливу складності класифікатора.

Параметр регуляризації $C=5$ контролює силу регуляризації - менші значення означають сильнішу регуляризацію. Значення $C=5$ було обрано як компроміс між запобіганням перенавчанню та збереженням здатності моделі до навчання складних патернів. Максимальна кількість ітерацій 1000 забезпечує достатній час для конвергенції алгоритму оптимізації на всіх типах векторних представлень.

Експериментальне дослідження проводилося з використанням різних розмірів навчальних вибірок для оцінки стабільності та масштабованості методів векторизації. Розміри тренувальних вибірок: 200, 400, 600, 800, 1000 зразків. Розмір тестової вибірки: 2000 зразків (фіксований для всіх експериментів)

Така стратегія дозволяє:

- оцінити поведінку методів при обмежених обсягах навчальних даних;
- виявити методи, що ефективно працюють на малих вибірках;
- проаналізувати швидкість покращення якості зі збільшенням навчальної вибірки.

Фіксований розмір тестової вибірки (2000 зразків) забезпечує консистентність порівняння та достатню статистичну потужність для виявлення відмінностей між методами.

Загальний алгоритм експерименту включає наступні кроки:

1) Ініціалізація методів векторизації. Цей крок включає створення екземпляра класу векторизатора і завантаження попередньо навчених моделей (для Word2Vec та BERT).

2) Випадковий відбір тренувальної вибірки та тестової вибірки заданих розмірів, попередня обробка текстів за допомогою функції `preprocess_text()`.

3) Векторизація тренувальної вибірки зі створенням словника (`vectorize_init`).

4) Векторизація тестової вибірки (`vectorize_text`) та фіксація часу, витраченого на неї.

5) Навчання та оцінка класифікатора: створення моделі `LogisticRegression`, навчання на векторах тренувальної вибірки, передбачення міток для тестової вибірки, розрахунок метрик якості (`classification_report`, `F1-score`).

6) Агрегація результатів: збереження `F1-score` для кожного розміру вибірки, розрахунок середнього часу векторизації одного тексту, вивід у консоль детальних звітів класифікації.

7) Візуалізація результатів: побудова графіків залежності `F1-score` від розміру навчальної вибірки, вивід у консоль швидкості роботи всіх методів.

Така методологія забезпечує справедливе порівняння методів векторизації, виключаючи вплив сторонніх факторів та дозволяючи сфокусуватися виключно на якості векторних представлень. Код реалізації навчання та оцінки класифікатора зі збереженням результатів наведено на рисунку 2.10.

```
def train_classifier(x_train, x_test, y_train, y_test, labels): 1 usage & mackaronina
    model = LogisticRegression(C=5, max_iter=1000)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_test = [labels[i] for i in y_test]
    y_pred = [labels[i] for i in y_pred]
    return (classification_report(y_test, y_pred, digits=4, labels=labels),
            f1_score(y_test, y_pred, average='weighted', labels=labels))

def evaluate_model(model_name, model, dataset): 1 usage & mackaronina
    f1_scores = []
    times = []
    for train_size in TRAIN_SIZES:
        train_texts, test_texts, y_train, y_test = dataset.split_train_test_classifier(train_size, TEST_SIZE)
        x_train = model.vectorize_init(train_texts)
        start = time.time()
        x_test = model.vectorize(test_texts)
        end = time.time()
        times.append((end - start) / TEST_SIZE)
        report, f1 = train_classifier(x_train, x_test, y_train, y_test, dataset.labels)
        print(f"Report for {model_name} with train size {train_size}\n{report}\n")
        f1_scores.append(f1)
    time_per_text = np.mean(times)
    return f1_scores, time_per_text
```

Рисунок 2.10 – Функції `train_classifier` і `evaluate_model`

`F1-score` є гармонічним середнім між `precision` (точністю) та `recall` (повнотою) і представляє собою одну з найважливіших метрик для оцінки якості класифікації, особливо у випадках незбалансованих класів [15].

Precision визначається як відношення кількості правильно передбачених позитивних зразків до загальної кількості зразків, передбачених як позитивні: $\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$, де TP - true positives, FP - false positives, FN - false negatives. Recall визначається як відношення кількості правильно передбачених позитивних зразків до загальної кількості фактично позитивних зразків: $\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$. F1-score обчислюється за формулою $\text{F1} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$

Основні переваги F1-score як метрики оцінки:

- балансованість - враховує як точність, так і повноту, запобігаючи оптимізації лише однієї з метрик;
- чутливість до якості - низькі значення precision або recall суттєво знижують F1-score;
- інтерпретованість - значення від 0 до 1, де 1 означає ідеальну класифікацію;
- стандартизованість - широко використовується в спільноті машинного навчання.

Для багатокласової класифікації використовується weighted F1-score, який обчислює F1-score для кожного класу окремо, а потім усереднює їх з врахуванням кількості зразків у кожному класі: $\text{Weighted F1} = \sum(w_i \times \text{F1}_i)$, де w_i - вага класу i (пропорційна до кількості зразків), F1_i - F1-score для класу i . У контексті нашого дослідження weighted F1-score особливо важливий, оскільки дозволяє об'єктивно порівнювати методи векторизації незалежно від можливих незначних дисбалансів у тестових вибірках різних розмірів.

2.7 Аналіз якості векторизації та швидкодії

На рисунку 2.11 зображено отримані графіки залежності F1-score від розміру навчальної вибірки для кожного з методів векторизації. Результати

порівняння методів демонструють суттєві відмінності в ефективності різних підходів.

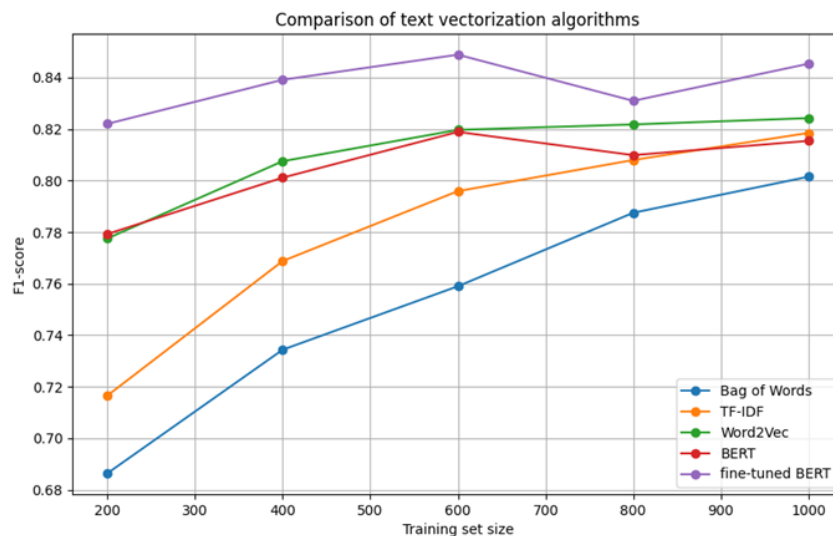


Рисунок 2.11 – Графіки F1-score для кожного з протестованих методів векторизації

Найвища якість досягається донавченою моделлю BERT, яка показує найкращі результати практично на всіх розмірах навчальної вибірки. F1-score цього методу коливається в діапазоні 0.82-0.85, демонструючи стабільно високу якість незалежно від обсягу навчальних даних.

Базова модель BERT показує дещо нижчі, але все ще конкурентоспроможні результати з F1-score в діапазоні 0.81-0.82. Можна відмітити, що різниця між базовою та донавченою версіями BERT не є кардинальною, але стабільно присутня на всіх розмірах вибірки.

Word2Vec демонструє третій за ефективністю результат з F1-score близько 0.77-0.83. Метод показує поступове покращення зі збільшенням розміру навчальної вибірки, що свідчить про його здатність ефективно використовувати семантичну інформацію.

TF-IDF та BoW показують подібні результати в діапазоні 0.69-0.82, причому TF-IDF має незначну перевагу. Обидва методи демонструють суттєве

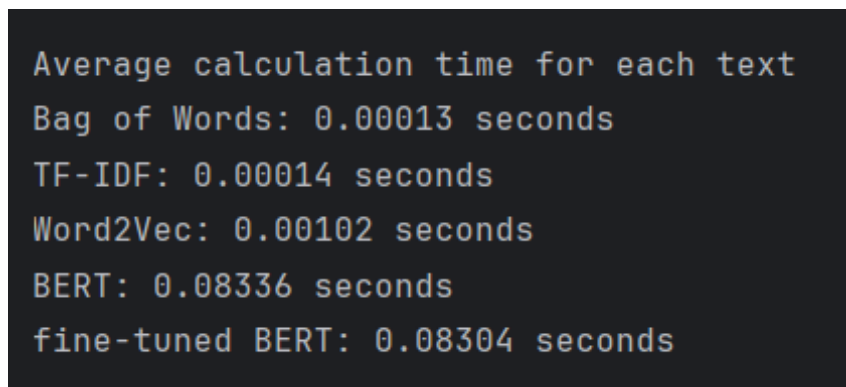
покращення зі збільшенням навчальної вибірки, що характерно для методів, що базуються на статистичних патернах.

Трансформерні моделі демонструють відносну стабільність результатів при різних розмірах вибірки. Це пояснюється тим, що вони використовують багаті попередньо навчені представлення, які вже містять значну кількість семантичної інформації.

Традиційні методи (BoW, TF-IDF) показують значне покращення зі збільшенням навчальної вибірки, особливо при переході від 200 до 600 зразків. Це свідчить про їх залежність від статистичної репрезентативності навчальних даних.

Word2Vec займає проміжне положення, демонструючи помірне покращення з ростом вибірки, що відображає його гібридну природу - використання попередньо навчених векторів слів з необхідністю статистичного усереднення на рівні документа.

На рисунку 2.12 показано результати вимірювання часу векторизації на одиничний текст, що демонструють певні відмінності між методами.



```
Average calculation time for each text
Bag of Words: 0.00013 seconds
TF-IDF: 0.00014 seconds
Word2Vec: 0.00102 seconds
BERT: 0.08336 seconds
fine-tuned BERT: 0.08304 seconds
```

Рисунок 2.12 – Середній час векторизації одного тексту для кожного з протестованих методів

Найшвидшими є традиційні методи BoW (0.00013 с) та TF-IDF (0.00014 с). Їх висока швидкість обумовлена простотою обчислень та можливістю ефективної реалізації через розріджені матриці. Word2Vec показує середню швидкість (0.00102 с), що пов'язано з необхідністю пошуку векторів слів у

словнику та обчислення середнього значення. BERT моделі є найповільнішими: базова модель потребує 0.08336 с, а донавчена - 0.08304 с на текст. Така швидкість обумовлена складністю трансформерної архітектури та необхідністю прямого проходу через 12 шарів attention механізмів.

2.8 Практичні рекомендації щодо вибору методу векторизації

На основі проведеного дослідження можна сформулювати наступні рекомендації для практичного застосування. Для застосувань, де пріоритетом є максимальна якість, рекомендується використання донавченої моделі BERT. Додаткові витрати на fine-tuning виправдовуються стабільним покращенням результатів, особливо при роботі з специфічними доменами. Також можна розглянути використання оптимізованих версій BERT (DistilBERT, ALBERT). Для систем реального часу з помірними вимогами до якості оптимальним вибором є Word2Vec. Метод забезпечує хороший баланс між якістю векторизації та швидкістю обробки. Для прототипування та швидкої розробки традиційні методи TF-IDF та BoW залишаються актуальними завдяки своїй простоті реалізації та налаштування.

Проведене дослідження має певні обмеження, які необхідно враховувати при інтерпретації результатів:

- Обмеженість одним типом завдання - класифікація настроїв може не повністю відображати ефективність методів в інших NLP задачах, таких як пошук інформації або генерація тексту.
- Фокус на англійських даних - результати можуть відрізнятися для інших мов, особливо для мов з обмеженими ресурсами або специфічними лінгвістичними особливостями.
- Використання стандартних гіперпараметрів - додаткове налаштування могло б покращити результати деяких методів, особливо традиційних.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ПОШУКУ ЦИТАТ НА ОСНОВІ ВЕКТОРИЗАЦІЇ WORD2VEC

На основі теоретичного аналізу методів векторизації слів, проведеного в першому розділі, та практичного порівняння алгоритмів на задачі класифікації тексту, описаного в другому розділі, було створено практичну систему для демонстрації ефективності методів обробки текстових даних. Метою третього розділу є розробка та реалізація Telegram-бота для інтелектуального пошуку цитат відомих людей з використанням найбільш оптимального методу векторизації тексту.

3.1 Програмні бібліотеки та інструменти

Для реалізації системи було використано описані у цьому підрозділі бібліотеки та інструменти.

SQLAlchemy (версія 2.x) - найпопулярніша ORM (Object-Relational Mapping) бібліотека для Python. Забезпечує автоматичне створення таблиць, безпечні SQL-запити через ORM, управління сесіями та транзакціями. Використовувалась для збереження цитат у базу даних і подальшої роботи з цією базою.

SQLite - легка вбудована реляційна база даних, яка не потребує окремого серверного процесу. Ідеально підходить для прототипування та невеликих додатків. Забезпечує ACID-властивості транзакцій, підтримує SQL-стандарт та має мінімальні вимоги до ресурсів. Файл бази даних зберігається локально як quotes.db.

Aiogram (версія 3.x) - сучасна асинхронна бібліотека для створення Telegram-ботів. Побудована на основі asyncio та забезпечує високу продуктивність через асинхронну обробку запитів. Підтримує всі сучасні функції Telegram Bot API, включаючи інлайн-режим, клавіатури, медіа-файли та webhook'и. У проекті використовуються класи Bot, Dispatcher для основної

логіки, `InlineQuery`, `InlineQueryResultArticle` для інлайн-функціональності та декоратори для обробки різних типів повідомлень.

`Asyncio` - стандартна бібліотека Python для асинхронного програмування. Забезпечує `event loop` для обробки асинхронних операцій, що дозволяє боту одночасно обробляти багато запитів без блокування. Використовується через `asyncio.run()` для запуску основного циклу бота та `async/await` синтаксис для асинхронних функцій.

`Logging` - стандартний модуль Python для логування подій програми. Налаштований на рівень `INFO` для відстеження роботи бота, включаючи час обробки запитів та можливі помилки. Забезпечує структуроване логування з мітками часу, що критично важливо для моніторингу продуктивності системи.

3.2 Вибір методу векторизації

Згідно з результатами експериментального дослідження, проведеного в другому розділі, метод `Word2Vec` продемонстрував найкраще співвідношення між якістю векторних представлень та обчислювальними витратами. Цей метод забезпечує достатню точність для задач семантичного пошуку при помірних вимогах до обчислювальних ресурсів, що робить його ідеальним вибором для створення реально функціонуючої системи. Предтренована модель займає ~3.6 ГБ на диску та ~4 ГБ в оперативній пам'яті. Векторизований корпус цитат додає лише ~280 МБ ($24000 \times 300 \times 4$ байти), що дозволяє розгортати систему на доступних VPS серверах.

3.3 Опис датасету та предметної області

Для реалізації системи пошуку цитат було обрано датасет `"english_historical_quotes"` користувача `m-ric` з платформи `Hugging Face`. Цей датасет містить понад 24 тисячі історичних цитат англійською мовою від відомих особистостей різних епох та сфер діяльності.

Структура датасету включає наступні поля:

- author - автор цитати (ім'я відомої особистості);
- quote - текст цитати;
- tags - теги, які характеризують тематику та контекст цитати.

На рисунку 3.1 зображено 8 перших записів в датасеті для демонстрації.



quote	author	category
Almost anyone can be an author the business is to collect money and fame from this state of being.	A. A. Milne	["business", "money"]
If you live to be a hundred, I want to live to be a hundred minus one day so I never have to live without you.	A. A. Milne	["love"]
Golf is so popular simply because it is the best game in the world at which to be bad.	A. A. Milne	["best"]
To the uneducated, an A is just three sticks.	A. A. Milne	["education"]
Promise me you'll always remember: You're braver than you believe, and stronger than you seem, and smarter than you...	A. A. Milne	["intelligence"]
One of the dreams of Zionism was to be a bridge. Instead, we are creating exclusion between the East and the West instead...	A. B. Yehoshua	["dreams"]
Major sports are major parts of society. It's not anomalous to have people who love sports come from other parts of that...	A. Bartlett Giamatti	["sports"]
There are many who lust for the simple answers of doctrine or decree. They are on the left and right. They are not confine...	A. Bartlett Giamatti	["society"]

Рисунок 3.1 – Приклад записів в датасеті

Датасет охоплює широкий спектр тем та авторів, включаючи філософів, письменників, політиків, науковців, митців та інших видатних особистостей. Такий різноманітний корпус цитат дозволяє створити систему, здатну знаходити релевантні висловлювання для широкого кола запитів користувачів.

3.4 Реалізація системи пошуку цитат

Клас QuotesSearch є центральним компонентом системи та реалізує повний цикл роботи з корпусом цитат - від ініціалізації до видачі результатів пошуку. Система спроектована з урахуванням принципів ефективності, масштабованості та надійності. Клас використовує патерн singleton-подібної ініціалізації, коли всі векторизовані дані завантажуються один раз при створенні об'єкта та зберігаються в пам'яті для швидкого доступу. Це рішення обумовлене специфікою задачі - корпус цитат є статичним та не змінюється

під час роботи системи. Повний код класу QuotesSearch наведено на рисунку 3.2.

```
class QuotesSearch: 2 usages  ⚡ mackaronina
    @db_connection  ⚡ mackaronina
    def __init__(self, session):
        self.vectorizer = Word2Vec()
        Base.metadata.create_all(engine)
        quotes = session.scalars(select(Quote)).all()
        if len(quotes) == 0:
            dataset = load_dataset(path="m-ric/english_historical_quotes", split="train")
            quotes = [Quote(quote=item["quote"], author=item["author"]) for item in dataset]
            session.add_all(quotes)
            session.commit()
        texts = [preprocess_text(quote.quote) for quote in quotes]
        self.corpus_vectors = self.vectorizer.vectorize_init(texts)
        self.quote_ids = [quote.id for quote in quotes]

    @db_connection  ⚡ mackaronina
    def search(self, input_text, session, num_similar=3):
        if input_text is None or len(input_text) == 0:
            return []
        input_vector = self.vectorizer.vectorize([preprocess_text(input_text)])
        if len(input_vector.shape) != 2:
            return []
        similarities = cosine_similarity(input_vector, self.corpus_vectors).flatten()
        top_indices = similarities.argsort()[::-1][:num_similar]
        similar_texts = [(session.get(Quote, self.quote_ids[i]), similarities[i]) for i in top_indices if
                          similarities[i] > 0.01]
        return similar_texts
```

Рисунок 3.2 – Клас QuotesSearch

Покроково розглянемо процес ініціалізації системи пошуку, реалізований в конструкторі `__init__`. Спочатку створюється об'єкт `Word2Vec`, який завантажує предтреновану модель `word2vec-google-news-300`. SQLAlchemy автоматично створює таблицю `quotes` якщо вона не існує. Використовується декларативний підхід з `mapped_column` для типобезпечного визначення полів. Система перевіряє чи є цитати в базі даних. Якщо база порожня, ініціюється процес завантаження датасету. Використовується `Hugging Face datasets` для завантаження корпусу `"m-ric/english_historical_quotes"`. Всі 24,000+ цитат вставляються в базу одною транзакцією для максимальної ефективності. Використовується `session.add_all()` для мінімізації накладних витрат. Кожна цитата проходить через функцію `preprocess_text()` для нормалізації (видалення HTML,

спецсимволів, приведення до нижнього регістру). Найбільш ресурсоємний етап - перетворення всіх цитат у вектори розмірності 300. Процес займає 1-2 хвилини залежно від продуктивності CPU. Зберігається відповідність між позицією вектору в масиві та ID цитати в базі даних для швидкого доступу.

Тепер розглянемо сам процес пошуку, реалізований методом `search`. Спочатку система перевіряє чи не є запит порожнім або `None`. Це критично важливо для стабільності роботи бота, оскільки Telegram може передавати порожні інлайн-запити. Текст запиту проходить ту ж саму обробку, що й цитати при ініціалізації, `preprocess_text()` забезпечує консистентність векторизації. Далі відбувається векторизація запиту класом `Word2Vec`, алгоритм якої описаний у минулому розділі. Перевіряється чи має результуючий вектор правильну форму (1, 300). Якщо всі слова запиту відсутні в `Word2Vec` моделі, вектор може бути порожнім. Використовується `cosine_similarity()` з `scikit-learn` для одночасного порівняння вектору запиту з усіма 24,000 векторами корпусу.

Косинусна подібність є ключовою метрикою для визначення семантичної близькості між текстами в системі пошуку цитат. Ця метрика обчислює косинус кута між двома векторами в багатовимірному просторі та має ряд важливих властивостей для текстового пошуку. Для двох векторів A та B косинусна подібність обчислюється за формулою: $\cos(\theta) = (A \cdot B) / (\|A\| \times \|B\|)$, де: $A \cdot B$ - скалярний добуток векторів, $\|A\|$ та $\|B\|$ - евклідові норми (довжини) векторів, θ - кут між векторами.

Властивості косинусної подібності:

- діапазон значень від -1 до 1, де 1 означає повну подібність, 0 - ортогональність (відсутність зв'язку), -1 - повну протилежність;
- метрика не залежить від довжини векторів, що важливо для текстів різної довжини;
- може бути розрахована одночасно для великих батчів векторів за допомогою матричних операцій.

Далі `similarities.argsort()` повертає індекси, відсортовані за зростанням подібності. Фільтруються результати з подібністю < 0.01 (1%) для усунення випадкових збігів. Для кожного індексу отримується повний об'єкт `Quote` з бази даних.

Реалізація має певні особливості використання. Якщо всі слова запиту відсутні в `Word2Vec` моделі, повертається порожній список результатів замість помилки. Запити з одного слова обробляються коректно, хоча можуть давати менш точні результати через відсутність контексту. Система розроблена для англійської мови, в результаті чого запити іншими мовами дадуть непередбачувані результати або порожні відповіді.

3.5 Реалізація телеграм бота

Бот реалізований з використанням бібліотеки `aiogram` та працює в інлайн-режимі. Асинхронність забезпечує неблокуючу обробку множинних запитів одночасно. Вибір Telegram як платформи для розгортання системи обумовлений кількома факторами: широка аудиторія користувачів (понад 700 мільйонів активних користувачів), потужний Bot API з підтримкою інлайн-режиму, відсутність необхідності створення окремого мобільного або веб додатку та зручність інтеграції з існуючими чатами користувачів.

Основні компоненти бота:

- Bot instance - основний об'єкт для взаємодії з Telegram API, ініціалізується з унікальним токеном. Токен бота отримується від `@BotFather` в Telegram та забезпечує автентифікацію API запитів.

- Dispatcher - центральний маршрутизатор, який розподіляє вхідні оновлення до відповідних обробників. Dispatcher налаштовано на прослуховування оновлень через long polling механізм.

- Handler functions - асинхронні функції для обробки різних типів повідомлень та запитів.

Частина коду реалізації бота наведено на рисунку 3.3.

```

searcher = QuotesSearch()

bot = Bot(token=BOT_TOKEN)
dp = Dispatcher()

@dp.message(Command("start")) & mackaronina
async def cmd_start(message: types.Message):
    await message.answer("Hi, I'm a bot that you can use to search for quotes via inline mode")

@dp.inline_query() & mackaronina
async def show_autocomplete(inline_query: InlineQuery):
    search_results = searcher.search(inline_query.query)
    inline_results = []
    for quote, similarity in search_results:
        full_text = quote.quote
        author = quote.author
        truncated_text = full_text[:150] + "..." if len(full_text) > 150 else full_text
        inline_results.append(InlineQueryResultArticle(
            id=str(quote.id),
            title=f"{author} ({similarity * 100:.2f}%)",
            description=truncated_text,
            input_message_content=InputTextMessageContent(
                message_text=f'"{full_text}" - {author}'
            )
        ))
    await inline_query.answer(inline_results, cache_time=3600)

```

Рисунок 3.3 – Код обробки команди /start та інлайн-запитів

Команда /start реалізує стандартний паттерн вітання для Telegram-ботів. Інформативне повідомлення надсилається в чат з ботом і пояснює користувачу, як використовувати бота в інлайн-режимі. Для інлайн-запиту користувач вводить юзернейм бота @search_quotes_bot у будь-якому чаті. Telegram відкриває інлайн-інтерфейс з полем для введення запиту. При кожній зміні тексту Telegram надсилає інлайн-запит до бота. Бот обробляє запит та повертає до 50 результатів (в нашому випадку - 3). Користувач бачить результати у вигляді списку з превью, при виборі результату його контент вставляється в чат.

При обробці інлайн-запиту inline_query.query містить текст, введений користувачем після @search_quotes_bot, aiogram автоматично парсить та валідує структуру інлайн-запиту. Виклик searcher.search(inline_query.query) ініціює повний цикл обробки: препроцесинг → векторизація → пошук подібності → ранжування. Результат повертається як список кортежів (Quote

object, similarity score), порожні запити обробляються системою пошуку з поверненням порожнього списку. Форматування результатів використовує клас `InlineQueryResultArticle` - специфічний тип результату для текстових відповідей. Унікальний ID для кожного результату (`str(quote.id)`) забезпечує коректне кешування. `Title` відображає автора та відсоток подібності для швидкої оцінки релевантності. `Description` показує скорочений текст цитати (максимум 150 символів) для попереднього перегляду. `InputTextMessageContent` визначає фінальний текст, який буде відправлений при виборі результату. Параметер `cache_time=3600` налаштовує Telegram на кешування результатів на 1 годину. Кешування зменшує навантаження на сервер та прискорює повторні запити. Telegram автоматично управляє кешем на стороні клієнта.

3.6 Демонстрація функціональності

На рисунку 3.4 продемонстровано приклад роботи системи для запиту "time is money".

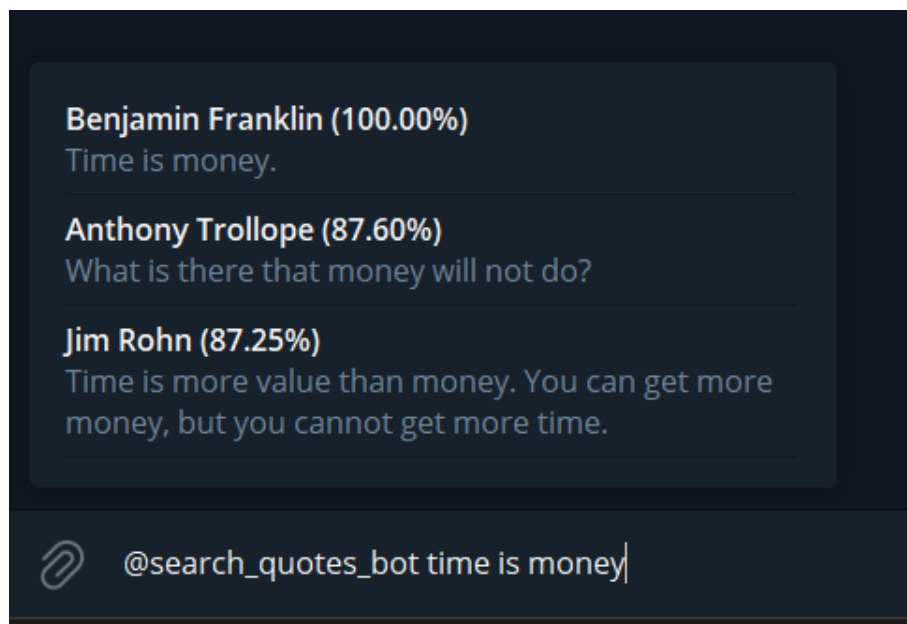


Рисунок 3.4 – Приклад роботи бота в інлайн-режимі для запиту time is money

Система знайшла три найбільш релевантні цитати:

– Benjamin Franklin (100.00%): "Time is money.";

- Anthony Trollope (87.60%): "What is there that money will not do?";
- Jim Rohn (87.25%): "Time is more value than money. You can get more money, but you cannot get more time.".

Результати демонструють високу точність семантичного пошуку. Система знайшла не лише точне співпадіння з запитом (цитата Бенджаміна Франкліна), але й семантично пов'язані висловлювання про гроші та час від інших авторів. На рисунку 3.5 показано результат вибору цитати користувачем.

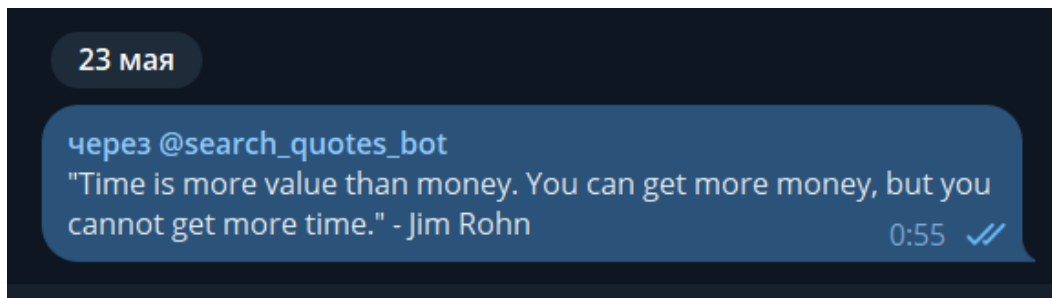


Рисунок 3.5 – Приклад відправлення в чат результату інлайн-запиту

Можна побачити, що система коректно форматує вибране висловлювання, включаючи повний текст цитати та авторство. Рисунок 3.6 демонструє логи роботи системи, які показують час обробки запитів.

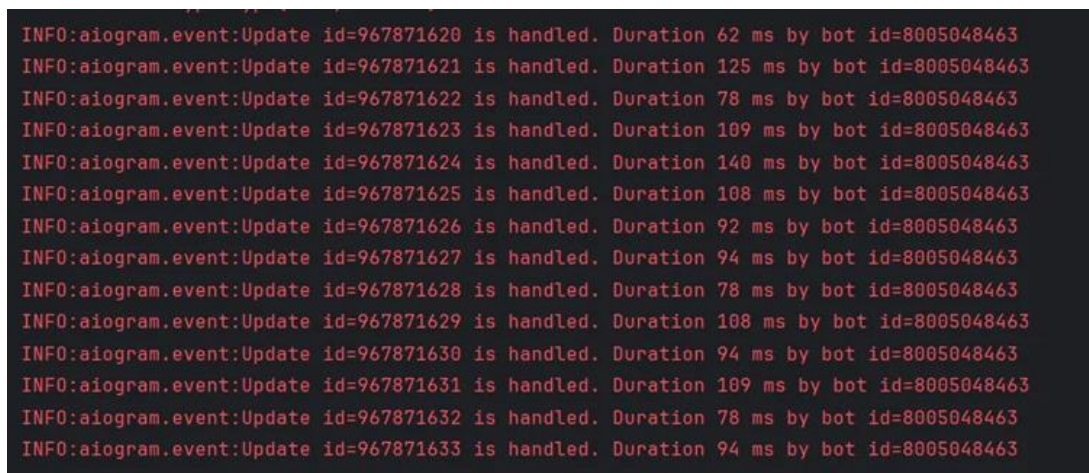


Рисунок 3.6 – Логи інлайн-запитів, які автоматично створює aiogram

Аналіз часових характеристик показує середній час відповіді 80-120 мс і високу стабільність роботи, без значних викидів у часі відповіді. Такі

показники швидкодії є прийнятними для системи реального часу та забезпечують комфортну взаємодію з користувачем. Час відповіді менше 150 мс сприймається користувачами як миттєва реакція системи.

3.7 Переваги та обмеження реалізованої системи

Реалізована система служить практичним підтвердженням ефективності методів векторизації слів у задачах семантичного пошуку та демонструє можливості їх застосування в реальних продуктах. Результати роботи показують, що навіть відносно прості методи векторизації, як Word2Vec, можуть забезпечувати високу якість роботи при правильній реалізації та оптимізації. Створена система може служити основою для подальших досліджень у сфері обробки природної мови та розробки більш складних систем семантичного пошуку та рекомендацій.

Основні переваги системи включають:

- Високу швидкодію: Система забезпечує відповідь за час менше 150 мс, що є критично важливим для інтерактивних додатків.
- Якісний семантичний пошук: Word2Vec дозволяє знаходити не лише точні співпадиння, але й семантично пов'язані цитати.
- Зручний інтерфейс: Інлайн-режим Telegram забезпечує інтуїтивно зрозумілу взаємодію без необхідності переходу в окремий чат з ботом.
- Масштабованість: Архітектура системи дозволяє легко розширювати корпус цитат та змінювати методи векторизації.

Поточні обмеження наступні:

- Мовна підтримка: Система працює лише з англійською мовою через використання англословної моделі Word2Vec та датасету.
- Розмір корпусу: Хоча 24 тисячі цитат є значним обсягом, розширення корпусу могло б покращити покриття різних тем.
- Метод векторизації: Word2Vec, незважаючи на свою ефективність, поступається більш сучасним методам як BERT у розумінні контексту.

– Простота ранжування: Використання лише косинусної подібності може бути недостатнім для складних запитів.

Для подальшого розвитку системи можна розглянути наступні вдосконалення:

– Багатомовність: Інтеграція моделей для інших мов дозволила б розширити аудиторію користувачів.

– Гібридний пошук: Комбінування кількох методів векторизації могло б покращити якість результатів.

– Персоналізація: Врахування історії запитів користувача для покращення релевантності результатів.

– Розширений корпус: Додавання сучасних цитат та висловлювань з різних джерел.

ВИСНОВКИ

У дипломній роботі було проведено комплексне дослідження методів векторизації слів при обробці текстових даних, що включало теоретичний аналіз, практичне порівняння алгоритмів та розробку прикладної системи. Результати дослідження дозволяють зробити наступні висновки.

Теоретичний аналіз показав, що сучасні методи векторизації еволюціонували від простих статистичних підходів до складних нейромережових архітектур. Кожен метод має свої унікальні характеристики: традиційні методи (Bag of Words, TF-IDF) забезпечують швидкодію та простоту реалізації, Word2Vec привносить семантичне розуміння через дистрибутивну гіпотезу, а BERT революціонізує галузь через контекстуальне розуміння та механізми уваги.

Експериментальне дослідження на задачі класифікації настроїв кінорецензій IMDb продемонструвало суттєві відмінності в ефективності різних підходів. На основі отриманих результатів сформульовано наступні рекомендації для практичного застосування. Для критично важливих застосувань з максимальними вимогами до якості рекомендується використання донавченої моделі BERT, незважаючи на підвищені обчислювальні витрати. Для систем реального часу оптимальним вибором є Word2Vec, що забезпечує хороший баланс між семантичним розумінням та швидкістю. Для прототипування та ресурсо-обмежених середовищ традиційні методи TF-IDF залишаються актуальними завдяки простоті реалізації та високій швидкості обробки.

Практична реалізація у вигляді Telegram-бота для пошуку цитат підтвердила теоретичні висновки про ефективність Word2Vec для задач семантичного пошуку. Система демонструє високу якість результатів (точне співпадіння та семантично пов'язані цитати) при прийнятній швидкодії (80-120 мс відповіді).

Дослідження має певні обмеження: фокус на одному типі завдання (класифікація настроїв), використання лише англomовних даних, застосування стандартних гіперпараметрів без додаткового налаштування. Ці обмеження створюють можливості для подальших досліджень. Подальший розвиток напрямку може включати дослідження гібридних підходів, що комбінують переваги різних методів, адаптацію для багатомовних застосувань, а також інтеграцію з новітніми архітектурами типу GPT та T5.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition // URL: https://www.researchgate.net/publication/200111340_Speech_and_Language_Processing_An_Introduction_to_Natural_Language_Processing_Computational_Linguistics_and_Speech_Recognition (дата звернення: 28.05.2025).
2. Manning C. D., Schütze H. Foundations of Statistical Natural Language Processing // URL: <https://nlp.stanford.edu/fsnlp/> (дата звернення: 28.05.2025).
3. Bird S., Klein E., Loper E. Natural Language Processing with Python // URL: <https://www.nltk.org/book/> (дата звернення: 28.05.2025).
4. Kudo T., Richardson J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing // URL: <https://aclanthology.org/D18-2012/> (дата звернення: 28.05.2025).
5. Plisson J., Lavrac N., Mladenic D. A rule based approach to word lemmatization // URL: <https://www.scribd.com/document/425546739/539616e81b02637b1bf90fb667ca2066cf14-pdf> (дата звернення: 28.05.2025).
6. Aggarwal C. C., Zhai C. Mining Text Data // URL: <https://link.springer.com/book/10.1007/978-1-4614-3223-4> (дата звернення: 28.05.2025).
7. Vaswani A., Shazeer N., Parmar N., et al. Attention Is All You Need // URL: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (дата звернення: 28.05.2025).
8. Salton G., Buckley C. Term-weighting approaches in automatic text retrieval // URL: <https://www.sciencedirect.com/science/article/abs/pii/0306457388900210> (дата звернення: 28.05.2025).
9. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space // URL:

https://www.researchgate.net/publication/234131319_Efficient_Estimation_of_Word_Representations_in_Vector_Space (дата звернення: 28.05.2025).

10. Devlin J., Chang M. W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // URL: <https://aclanthology.org/N19-1423/> (дата звернення: 28.05.2025).

11. Maas A. L., Daly R. E., Pham P. T., Huang D., Ng A. Y., Potts C. Learning Word Vectors for Sentiment Analysis // URL: <https://aclanthology.org/P11-1015/> (дата звернення: 29.05.2025).

12. Muennighoff N., Tazi N., Magne L., Reimers N. MTEB: Massive Text Embedding Benchmark // URL: <https://aclanthology.org/2023.eacl-main.148/> (дата звернення: 29.05.2025).

13. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., et al. Scikit-learn: Machine Learning in Python // URL: https://www.researchgate.net/publication/51969319_Scikit-learn_Machine_Learning_in_Python (дата звернення: 29.05.2025).

14. Hosmer D. W., Lemeshow S., Sturdivant R. X. Applied Logistic Regression // URL: <https://onlinelibrary.wiley.com/doi/book/10.1002/9781118548387> (дата звернення: 29.05.2025).

15. Powers D. M. W. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation // URL: https://www.researchgate.net/publication/228529307_Evaluation_From_Precision_Recall_and_F-Factor_to_ROC_Informedness_Markedness_Correlation (дата звернення: 29.05.2025).