НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»

УДК 004.42:004.932

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Метод та програмне забезпечення автоматизованого тегування текстових даних»

Виконав:
студент ІІ курсу, групи КП-31мн
Бабак Артем Андрійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Нещадим Олександр Михайлович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПСКС, к.т.н., доцент,
Боярінова Юлія Євгенівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Бабаку Артему Андрійовичу

1. Тема дисертації «Метод та програмне забезпечення автоматизованого тегування текстових даних», керівник проєкту Нещадим Олександр Михайлович, к.т.н., доцент, затверджені наказом по університету № 1219-С від 21.03.2025 р.
2. Термін подання студентом дисертації «16» травня 2025 р.
3. Об’єкт дослідження: процес автоматизованого тегування текстових даних.
4. Предмет дослідження: методи та алгоритми тегування текстових даних.
5. Перелік завдань, які потрібно вирішити:
   - провести аналіз існуючих методів тегування текстових даних;
   - дослідити архітектури нейронних мереж для навчання;
   - дослідити способи збору інформації для навчання нейронних мереж;
   - розробити програму для навчання мережі;
   - провести тестування натренованої моделі;
   - провести аналіз отриманих результатів.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - загальний датафлоу програми;
   - діаграма компонентів системи;
   - графік тренування моделі;
   - інтерфейс користувача;
   - знаходження ключових слів;
   - випадковий ліс.

7. Орієнтовний перелік публікацій:

- Тези доповіді “Метод та програмне забезпечення автоматизованого тегування текстових даних ”.

- Стаття “Algorithmic-software for automated text tegging”.

8. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

9. Дата видачі завдання «10» жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою дисертації	15.11.2023	
2.	Розроблення та узгодження технічного завдання	21.03.2024	
3.	Дослідження існуючих рішень	07.05.2024	
4.	Підготовка матеріалів першого розділу магістерської дисертації	10.10.2024	
5.	Збір інформації для тренування нейронних мереж	15.01.2025	
6.	Підготовка матеріалів другого розділу магістерської дисертації	19.02.2025	
7.	Тренування нейронних мереж	27.02.2025	
8.	Тестування нейронних мереж	24.03.2025	
9.	Підготовка матеріалів третього розділу магістерської дисертації	27.03.2025	
10.	Підготовка матеріалів четвертого розділу магістерської дисертації	10.04.2025	
11.	Підготовка графічної частини магістерської дисертації	26.04.2025	
12.	Оформлення документації магістерської дисертації	01.05.2025	

Студент Артем БАБАК

Науковий керівник Олександр НЕЩАДИМ

РЕФЕРАТ

Актуальність теми. Сфера пошуку текстових даних є масштабною та впливовою системою, яка об’єднує потреби бізнесу та користувачів, які шукають інформацію. У багатьох випадках знаходження потрібного файлу або публікації, що відповідає запитам людини, може зайняти значний час.

Одним із основних способів оптимізації пошуку є застосування сучасних інформаційних технологій, таких як бази даних, спеціалізовані вебсайти тощо, що значно прискорюють доступ до необхідних даних. Однак недостатнє розуміння схожостей та відмінностей між документами, а також їх взаємозв’язків може спричинити марну витрату коштів, часу та зусиль під час пошуку.

Тому сьогодні особливо важливим є покращення якості класифікації текстових матеріалів для забезпечення користувачів більш точними результатами при пошуку публікацій або документів.

На сьогодні існує чимало методик визначення категорій тексту, проте багато з них недостатньо ефективні для документів, оскільки вони часто містять приховані тематичні зв’язки, залежні від місця та часу їх створення. Внаслідок цього традиційні алгоритми обробки текстів можуть давати неточні результати, оскільки не враховують усіх нюансів таких даних.

Саме тому ключовим етапом у розробці систем автоматичного тегування текстів є облік цих особливостей, адже пошук серед попередньо розмічених даних є значно ефективнішим і швидшим.

Об’єктом дослідження є вдосконалення процес аналізу і розпізнавання геопозиції зображень за допомогою алгоритмів нейронних мережі.

Предметом дослідження є методи та алгоритми роботи з текстовими даними.

Мета роботи полягає у розробці ефективного методу для автоматизованого тегування текстових даних.

Методи дослідження. У роботі використано комплексний підхід, що поєднує теоретичний аналіз та практичні експерименти. На теоретичному рівні було проведено систематичний огляд наукової літератури, досліджено сучасні методики тегування текстових даних, а також здійснено інтеграцію отриманих даних для розробки концептуальних основ запропонованого алгоритму. Практична частина роботи включала експериментальні дослідження із застосуванням глибокого навчання, оцінку точності прогнозування координат на контрольних вибірках даних та аналіз ефективності запропонованого методу порівняно з класичними технологіями тегування текстових даних.

Наукова новизна розробленого методу закладається в модифікуванні існуючого підходу, щоб дозволяє покращити процес автоматизованого тегування текстових даних.

Практична цінність отриманих результатів полягає в тому, щоб дозволить більше точно виявляти теги та ключові слова у тексті.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на 17 науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2024 та опубліковані у збірнику тез доповідей.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі представлено загальну характеристику дослідження, визначено актуальність проблеми автоматичного визначення геопозиції об'єктів на зображеннях, здійснено аналіз сучасного стану питання, сформульовано мету, завдання, об'єкт і предмет дослідження.

У першому розділі досліджуються сучасні виклики у сфері автоматизованого тегування текстових даних. У ньому ретельно розглядаються актуальні методи обробки тексту, технології виявлення ключових слів та існуючі обмеження стандартних підходів. Особливий акцент робиться на

аналізі типових труднощів, що виникають під час знаходження контексту тексту та семантичних ознак.

У другому розділі презентується математична основа розробленого алгоритмічного рішення. Тут детально описується архітектура згорткових нейронних мереж глибокого навчання, принципи їх функціонування та навчання. Значна увага приділяється методам аналізу семантичного змісту тексту і ролі контексту у процесі визначення тегів. Як ключовий структурний елемент системи розглядається модель BERT.

У третьому розділі визначаються основні вимоги до розробленого методу, обґрунтовується вибір програмних інструментів та архітектурних рішень. Детально описується загальна структура програмної реалізації, зокрема спеціалізований модуль тегування тексту, який забезпечує підвищення точності через комбінування існуючих методів.

У четвертому розділі наведено інформацію про тестування програми, зокрема, про метрики використані для цього та навчальну вибірку, вибір і налаштування параметрів, а також описано проведення експериментів. Представлено результати тестування системи, зокрема точність класифікації, порівняння з традиційними методами та аналіз отриманих результатів.

У висновках проаналізовано отримані результати роботи.

Ключові слова: тегування тексту, аналіз тексту, машинне навчання, оцінка текстів, пошук семантичних ознак.

ABSTRACT

Actuality. The field of text document retrieval represents a large-scale and influential system that bridges the needs of businesses and information-seeking users. In many cases, locating a specific file or publication that matches a user's query can require considerable time.

One of the primary methods for optimizing search is the application of modern information technologies, such as databases and specialized websites, which significantly accelerate access to relevant data. However, insufficient understanding of document similarities, differences, and interconnections can lead to wasted resources time, money, and effort during the search process.

For this reason, improving the classification quality of textual materials is particularly crucial today, as it ensures users receive more accurate results when searching for publications or documents.

While numerous techniques exist for determining text categories, many remain insufficiently effective for documents, as they often contain hidden thematic relationships dependent on their creation time and location. Consequently, traditional text processing algorithms may produce inaccurate results by failing to account for all the nuances in such data.

This is precisely why incorporating these unique characteristics is a critical step in developing automated text-tagging systems searching through pre-labeled data proves far more efficient and faster.

Object of the research: The study focuses on methodologies and computational algorithms for textual data processing and analysis.

Methods of research. The study employs a comprehensive approach that combines theoretical analysis and practical experiments. At the theoretical level, a systematic review of scientific literature was conducted, modern methods of textual data tagging were examined, and the obtained data was integrated to develop the conceptual framework of the proposed algorithm. The practical part of the work included experimental research using deep learning, evaluation of coordinate

prediction accuracy on control data samples, and analysis of the effectiveness of the proposed method compared to classical textual data tagging technologies.

Scientific novelty lies in the modification of the existing approach, which allows for improved accuracy of image geolocation recognition.

Practical value of the developed method lies in the modification of the existing approach, which improves the process of automated textual data tagging.

Approbation. The main concepts and results of this work were presented and discussed at the 17th scientific conference for graduate and postgraduate students *"Applied Mathematics and Computing" (PMK-2024)* and published in the conference abstracts collection.

Structure and scope of the work. The master's thesis consists of an introduction, four chapters, conclusions, and appendices.

In the introduction, the general overview of the research is presented, the relevance of the problem of automatic geolocation detection of objects in images is established, the current state of the issue is analyzed, and the research objectives, tasks, subject, and object are defined.

The first chapter explores contemporary challenges in the field of automated textual data tagging. It thoroughly examines current text processing methods, keyword extraction technologies, and the limitations of conventional approaches. Special emphasis is placed on analyzing common difficulties encountered in determining textual context and semantic features.

The second chapter presents the mathematical foundation of the developed algorithmic solution. It details the architecture of deep learning convolutional neural networks, their operational principles, and training mechanisms. Significant attention is given to methods of analyzing textual semantic meaning and the role of context in the tagging process. The BERT model is discussed as a key structural component of the system.

The third chapter outlines the core requirements for the developed method, justifies the selection of software tools and architectural solutions, and provides a detailed description of the software implementation’s overall structure. This

includes a specialized text tagging module that enhances accuracy by combining existing methods.

The fourth chapter presents information on software testing, including the metrics used, the training dataset, parameter selection and tuning, and a description of the experiments conducted. The system's test results are showcased, covering classification accuracy, comparisons with traditional methods, and an analysis of the findings.

The conclusions provide an analytical summary of the obtained results.

Keywords: text tagging, text analysis, machine learning, text evaluation, semantic feature extraction.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Наївний баєсів класифікатор – це ймовірнісний класифікатор, щовикористовує теорему Баєса для визначення ймовірності приналежностіспостереження (елемента вибірки) до одного з класів приприпущенні (наївному) незалежності змінних.

F-оцінка – критерій ефективності, який у статистичному аналізідвійкової класифікації є мірою точності визначення категорій, які присутнів тексті.

Метод k-найближчих сусідів – метричний алгоритм для автоматичноїкласифікації об'єктів. Основним принципом методу найближчих сусідів єте, що об'єкт присвоюється тому класу, який є найбільш поширеним середсусідів даного елемента.

Дерево ухвалення рішень (також може називатися деревамикласифікацій або регресійними деревами) – використовується в галузістатистики та аналізу даних для прогнозних моделей. Структура деревамістить такі елементи: «листя» і «гілки». На ребрах («гілках») дереваухвалення рішення записані атрибути, від яких залежить цільова функція, в«листі» записані значення цільової функції, а в інших вузлах – атрибути, заякими розрізняються випадки. Щоб класифікувати новий випадок, требаспуститися по дереву до листка і видати відповідне значення. Подібні дереварішень широко використовуються в інтелектуальному аналізі даних. Метаполягає в тому, щоб створити модель, яка прогнозує значення цільовоїзмінної на основі декількох змінних на вході.

Фреймворк – це реальна або концептуальна структура, призначенаслугувати опорою або керівництвом для побудови чогось, що розширюєструктуру на щось корисне.

ВСТУП

Сучасний світ характеризується стрімким зростанням інформаційних масивів, відомих як "великі дані" (англ. big data). Це зумовлює необхідність використання спеціалізованих підходів для автоматичного сортування текстових ресурсів. Паралельно збільшується кількість користувачів, які потребують доступу до конкретних текстових матеріалів (наприклад, форумні обговорення, описи вакансій тощо), що призводить до підвищеного навантаження на пошукові системи та збільшення часу обробки запитів.

У зв’язку з цим виникає потреба у створенні механізмів класифікації текстів, здатних ефективно працювати в умовах високого навантаження. Враховуючи постійне зростання обсягів інформації, числа користувачів та їхніх запитів, критично важливим завданням для розробників програмного забезпечення стає моніторинг продуктивності системи та пошук оптимальних методів розподілу текстових документів за відповідними категоріями.

Наразі існує чимало алгоритмів для автоматизованого сортування текстів, проте більшість із них вимагають глибоких знань від розробників або адміністраторів баз даних. Удосконалення пошукових механізмів та підвищення точності категорізації за допомогою нових моделей автоматичного тегування дозволить значно прискорити та оптимізувати процес пошуку інформації.

Хоча існують різні методи визначення категорій для текстів, багато з них не підходять для обробки спеціалізованих документів, оскільки такі тексти часто містять приховані класифікаційні ознаки, пов’язані з часом публікації або контекстом. Традиційні алгоритми обробки даних можуть давати неточні результати, оскільки не враховують усіх нюансів текстів. Тому при розробці програмних рішень для автоматичного тегування необхідно враховувати специфіку документів – це забезпечить більш точний та швидкий пошук у вже класифікованих даних.

1. ОГЛЯД ІСНУЮЧИХ МЕТОДІВ АНАЛІЗУ ТА КЛАСИФІКАЦІЇ ТЕКСТУ

1.1. Загальний огляд методів для тегування текстових даних

Текстова класифікація (також відома як категоризація або тегування тексту) – це процес віднесення текстового фрагмента до однієї або кількох із заздалегідь визначених категорій [5]. Такі системи дають можливість упорядковувати, структурувати та групувати різноманітні текстові дані. Наприклад:

- Новинні статті можна розподіляти за тематиками.
- Чат-повідомлення – за мовою спілкування.
- Відгуки про продукти – за тональністю (позитивні, нейтральні, негативні).

Щоб надати приклад роботи класифікатора, розглянемо речення:

“The user interface is quite straightforward and easy to use.”

За допомогою алгоритму класифікації його можна проаналізувати та автоматично позначити відповідними мітками, наприклад "UI" (користувацький інтерфейс) і "Easy To Use" (простота використання).

Існує два основних підходи до класифікації текстів:

- Ручна класифікація – фахівець самостійно аналізує текст і визначає його категорію. Незважаючи на високу точність, цей метод вимагає значних часових та фінансових витрат.
- Автоматизована класифікація – використовує методи машинного навчання та обробки природної мови (NLP), що дозволяє швидко і ефективно обробляти великі обсяги даних.

Сучасні алгоритми текстових класифікаторів поділяються на три групи:

- Системи на основі правил – використовують наперед задані критерії для віднесення тексту до певної категорії.

- Системи на основі машинного навчання – "навчаються" на великих наборах даних, виявляючи закономірності для подальшої класифікації.
- Гібридні системи – поєднують переваги перших двох підходів для підвищення точності та ефективності.

Далі розглянемо кожен із цих методів детальніше.

1.1.1. Методологія класифікації текстів на основі лінгвістичних правил

Сучасні дослідження в галузі автоматизованої обробки текстів виділяють правило-орієнтований підхід як один з фундаментальних методів категорізації текстових даних. Дана методологія базується на формалізації експертних знань через систему лінгвістичних правил та семантичних маркерів.

Розглянемо імплементацію даного підходу на прикладі класифікації наукових статей за категоріями "Біотехнології" та "Квантова фізика".

Формування онтології предметної області:

- Для категорії "Біотехнології": {"CRISPR", "генна інженерія", "ДНК-послідовність", ...}.
- Для категорії "Квантова фізика": {"квантова заплутаність", "суперпозиція", "кубіт", ...}.

Реалізація механізму класифікації:

- Текст T аналізується на наявність термінів з онтологічних словників.
- Виконується підрахунок вагових коефіцієнтів для кожної категорії.
- Приймається рішення на основі максимізації вагової функції.

Для текстового документу: "Нові методи редагування геному з використанням CRISPR-Cas9 системи" система ідентифікує терміни "CRISPR" та "редагування геному" як детермінанти категорії "Біотехнології".

Системи реалізовані на основі правил є простими і зрозумілими, тому розглянемо переваги та недоліки даних систем.

Переваги:

- Чіткі критерії класифікації для спеціалізованих текстів.
- Можливість врахування домен-специфічної термінології.
- Висока точність для вузькоспеціалізованих матеріалів.

Недоліки:

- Складність оновлення термінологічних словників.
- Проблеми з класифікацією міждисциплінарних досліджень.
- Необхідність постійного коригування правил для нових термінів.

Отже, запропонований підхід особливо ефективний для класифікації науково-технічних текстів, де існує чітко визначена професійна лексика. Подальші дослідження можуть бути спрямовані на розробку адаптивних онтологій, здатних до напів-автоматичного оновлення.

1.1.2. Методи машинного навчання для автоматичної класифікації текстів

Системи машинного навчання, призначені для класифікації текстів, базуються на аналізі попередньо розмічених даних. Навчаючись на прикладах, алгоритми виявляють зв'язки між окремими частинами тексту та відповідними мітками, такими як теми чи категорії.

Метод починає свою роботу з виділення ознак із тексту шляхом його перетворення на числовий формат. Одним із найпоширеніших методів є так званий "мішок слів" (Bag of Words), де текст представляється у вигляді вектора, який відображає частоту слів із заданого словника. Наприклад, якщо словник містить слова: "this", "is", "the", "not", "awesome", "bad", "basketball", то речення "This is awesome" перетвориться на вектор (1, 1, 0, 0, 1, 0, 0).

Після цього алгоритм аналізує навчальні дані, що складаються з векторів (ознак) та відповідних міток, таких як "спорт" чи "політика", і будує класифікаційну модель. Після завершення навчання модель може класифікувати нові тексти. Для цього невідомий текст спочатку

перетворюється на вектор за тим же принципом, а потім подається на вхід моделі, яка визначає його категорію.

Автоматичні моделі зазвичай точніші за правила, створені вручну, особливо у складних випадках. Крім того, моделі можна легко оновлювати, додаючи нові категорії або дані для навчання, що робить їх більш гнучкими у порівнянні з традиційними підходами.

1.1.3. Гібридні системи класифікації

Гібридні системи поєднують переваги машинного навчання та правил, створених експертами, для підвищення точності класифікації. Такі системи інтегрують базові навчальні класифікатори з механізмами, заснованими на правилах, що дозволяє корегувати результати у складних випадках.

Основна перевага гібридного підходу полягає у гнучкості налаштування. Якщо базовий класифікатор помиляється щодо певних категорій або тегів, можна додати спеціальні правила для корекції цих помилок. Наприклад, якщо модель неправильно класифікує тексти про спорт, можна ввести додаткові логічні умови або словникові фільтри для покращення результатів.

Цей підхід особливо ефективний у ситуаціях, коли:

- Деякі категорії мають чіткі ознаки, які можна формалізувати (наприклад, наявність певних ключових слів).
- Навчальні дані недостатньо репрезентативні для окремих класів.
- Потрібно враховувати предметно-специфічні нюанси, які важко виявити автоматичними методами.

Таким чином, гібридні системи дозволяють поєднувати масштабованість машинного навчання з точністю експертних правил, забезпечуючи більш надійну класифікацію текстів у реальних умовах.

1.2. Огляд існуючих рішень для класифікації

Процес автоматизованої класифікації текстів починається з етапу підготовки даних, який є критично важливим для якості подальшого аналізу.

Цей процес складається з трьох основних етапів: вибору даних, попередньої обробки та трансформації.

На етапі вибору даних визначається підмножина інформації, яка буде використовуватися для навчання алгоритму. Хоча існує спокуса залучити всі доступні дані, важливо ретельно відібрати лише ті, що відповідають поставленій задачі. Цей процес передбачає аналіз наявної інформації, виявлення відсутніх даних та визначення непотрібних даних, які слід виключити.

Попередня обробка включає комплекс заходів для підготовки вибраних даних:

- Очищення даних – заповнення пропусків, корекція некоректних значень та видалення невідповідностей.
- Інтеграція даних – об'єднання інформації з різних джерел у єдину структуру.
- Перетворення даних – приведення до стандартного формату для подальшої обробки.
- Зменшення обсягу даних – усунення надлишкових параметрів без втрати якості інформації.
- Дискретизація – заміна числових значень категоріальними.

Завершальний етап – трансформація даних передбачає:

- Декомпозицію атрибутів (розбиття складних характеристик на простіші).
- Агрегування атрибутів (об'єднання пов'язаних параметрів).
- Масштабування (приведення даних до єдиного масштабу).

Після ретельного підготовчого етапу дані готові для застосування контрольованих алгоритмів машинного навчання, які будуть розглянуті далі. Такий комплексний підхід до підготовки даних забезпечує високу ефективність подальшого процесу класифікації текстових описів.

1.2.1. Дерева рішень

Дерева рішень є одним з найпопулярніших алгоритмів машинного навчання для класифікації та обробки даних. Цей метод представляє собою ієрархічну структуру, що нагадує блок-схему, яка складається з трьох основних елементів:

- Кореневого вузла (початкова точка аналізу).
- Нелистових вузлів (проміжні точки прийняття рішень).
- Листкових вузлів (фінальні результати класифікації).

Основа роботи дерева рішень полягає в рекурсивному розділенні вихідного набору даних на все більш однорідні підмножини. Кожен нелистовий вузол містить умову перевірки певного атрибута, тоді як гілки відображають можливі результати цієї перевірки. Процес розщеплення базується на спеціальних критеріях, серед яких найбільш поширені:

- Інформаційний приріст (алгоритм ID3).
- Коефіцієнт співвідношення (алгоритм C4.5).
- Індекс Джині (алгоритм CART).

Сучасні модифікації, такі як C5 (розширена версія C4.5), пропонують покращену функціональність. Після побудови дерева класифікація нового об'єкта відбувається шляхом послідовного проходження від кореневого вузла до листкового, з використанням встановлених критеріїв перевірки.

Головні сильні сторони дерев рішень включають:

- Високу інтерпретованість – структура дерева є наочною та зрозумілою навіть для непрофесіоналів.
- Універсальність – можливість роботи з різними типами даних (числовими та категоріальними).
- Простоту трансформації – дерево легко конвертується в набір правил "якщо-то".
- Ефективність при наявності чітких закономірностей у даних.

Незважаючи на переваги, метод має певні обмеження:

- Вимагає дискретних значень цільового атрибута (для більшості класичних алгоритмів).
- Ефективність знижується при складних взаємодіях між численними атрибутами.
- Чутливість до дисбалансу даних – перевага може надаватися домінуючим класам.
- Схильність до перенавчання при надмірній глибині дерева.

Дерева рішень залишаються потужним інструментом для класифікаційних завдань, особливо коли важливі інтерпретованість результатів та простота використання. Однак для складних задач з великою кількістю взаємопов'язаних ознак доцільно розглядати комбіновані підходи, такі як випадкові ліси або бустинг на основі дерев рішень.

1.2.2. Баєсівські методи

Байєсівські методи класифікації ґрунтуються на імовірнісному підході до аналізу даних. Вони дозволяють визначати ймовірність належності об'єкта до певного класу на основі наявних ознак. Основою цих методів є теорема Баєса, яка дає змогу обчислити апостеріорну ймовірність P(H|X) – імовірність гіпотези H за наявності даних X [1].

Цей метод передбачає умовну незалежність ознак, що значно спрощує обчислення. Незважаючи на очевидне спрощення, він показує хороші результати, особливо при невеликих обсягах даних. Алгоритм працює за принципом вибору класу з максимальною апостеріорною ймовірністю для заданого набору ознак.

Переваги байєсівських методів:

- Висока швидкість роботи.
- Можливість обробки різних типів даних.
- Простота інтерпретації результатів.
- Ефективність при обмежених обсягах даних.

Обмеження методів:

- Припущення про незалежність ознак часто не виконується на практиці.
- Складність обчислень для складних моделей.
- Менша точність у випадках сильних залежностей між змінними.
- Необхідність точного визначення апріорних ймовірностей.

Байєсівські методи особливо ефективні для текстових даних, де вони дозволяють враховувати частоту появи слів і їх взаємозв'язки. Вони знайшли широке застосування в задачах фільтрації спаму, категоризації документів та аналізі настроїв.

1.2.3. Логістична регресія

Логістична регресія є популярним статистичним методом для аналізу залежностей між змінними, особливо коли результативна ознака має бінарний характер. На відміну від лінійної регресії, цей метод дозволяє передбачати ймовірність настання події за допомогою спеціальної логістичної функції.

Основні переваги:

- Менша чутливість до викидів порівняно з іншими методами.
- Гнучкість у виборі предикторів (якісні та кількісні).
- Інтерпретованість результатів (ймовірнісна форма виходу).
- Добре справляється з шумом у даних.
- Можливість оцінки значимості кожного фактора.

Існуючі обмеження:

- Вимагає достатньо великої вибірки для стабільних результатів.
- Складність у правильному підборі незалежних змінних.
- Висока обчислювальна складність для великих датасетів.
- Лінійність зв'язку між логарифмом шансів та предикторами.
- Чутливість до мультиколінеарності предикторів.

Для покращення результатів часто використовують регуляризацію (L1, L2) та методи відбору ознак. Логістична регресія особливо ефективна, коли

важливо не лише отримати прогноз, а й зрозуміти вплив окремих факторів на результат.

1.2.4. Векторні машини

Метод опорних векторів (SVM) є потужним алгоритмом машинного навчання, який ефективно вирішує задачі як класифікації, так і регресії. Його ключова особливість полягає у пошуку оптимальної розділяючої гіперплощини у просторі ознак, яка максимізує відстань до найближчих точок різних класів. Ці найближчі точки називаються опорними векторами і визначають положення гіперплощини.

Переваги методу:

- Висока точність у багатьох практичних задачах.
- Ефективність у високовимірних просторах.
- Стійкість до перенавчання, особливо при невеликих вибірках.
- Гнучкість завдяки різноманіттю ядерних функцій.

Обмеження та недоліки:

- Висока обчислювальна складність для великих наборів даних.
- Чутливість до вибору ядра та його параметрів.
- Складність інтерпретації результатів у випадку складних ядер.
- Необхідність ретельного масштабування вхідних даних.
- Обмежена ефективність для дуже великих датасетів.

1.2.5. Штучні нейронні мережі

Штучні нейронні мережі (ANN) є потужною обчислювальною моделлю, натхненною біологічними нейронними системами. Ця архітектура складається з взаємопов'язаних вузлів (нейронів), які спільно обробляють інформацію для вирішення складних завдань. Нейронні мережі демонструють здатність адаптувати свою структуру в процесі навчання на основі вхідних даних, що робить їх особливо ефективними для роботи з нелінійними залежностями.

Ключові характеристики нейронних мереж:

- Гнучкість архітектури – від простих перцептронів до складних багатошарових структур.
- Навчання за рахунок корекції вагових коефіцієнтів між нейронами.
- Здатність до узагальнення – виявлення складних закономірностей у даних.
- Широкий спектр архітектур – зворотне поширення, згорткові мережі, рекурентні структури.

Переваги методу:

- Висока адаптивність до різних типів завдань.
- Здатність самоорганізації та виявлення складних залежностей.
- Ефективна робота з шумними та неповними даними.
- Швидкість прийняття рішень після завершення навчання.
- Можливість паралельної обробки інформації.

Недоліки:

- Необхідність попередньої цифрової обробки всіх вхідних даних.
- Ризик зупинки на локальних мінімумах під час навчання.
- Висока обчислювальна складність для великих мереж.
- Складність інтерпретації внутрішніх механізмів роботи.
- Висока вартість обчислювальних ресурсів для навчання.
- Відсутність можливості вбудовування апріорних знань.

Для подолання цих обмежень розроблені різні методи, такі як використання регуляризації, спеціальних функцій активації та оптимізованих алгоритмів навчання. Нейронні мережі особливо ефективні у поєднанні з іншими методами машинного навчання, формуючи гібридні системи з покращеними характеристиками.

1.3. Висновки до розділу 1

У даному розділі проведено детальний аналіз сучасних методів класифікації, які знаходять застосування при роботі з текстовими даними.

Дослідження охопило широкий спектр алгоритмів – від класичних статистичних підходів до сучасних методів машинного навчання. Кожен із розглянутих методів має свої унікальні особливості, переваги та обмеження, що визначають сфери їх найефективнішого застосування.

Особливу увагу приділено порівняльній характеристиці різних підходів, що дозволило виявити закономірності їхньої ефективності залежно від специфіки поставлених завдань. На основі проведеного аналізу запропоновано інтегрований підхід, який передбачає комбінування сильних сторін різних алгоритмів для досягнення оптимальних результатів класифікації.

Важливим аспектом дослідження стало вивчення критеріїв оцінки якості роботи класифікаторів у контексті обробки великих масивів даних. Розглянуто сучасні інструменти та методики, що дозволяють ефективно застосовувати алгоритми класифікації в умовах обмежених обчислювальних ресурсів.

2. ЗАПРОПОНОВАНИЙ МЕТОД АНАЛІЗУ ТА ТЕГУВАННЯ ТЕКСТОВИХ ДАНИХ

2.1. Огляд алгоритмів для рішення проблеми

Класифікація текстів становить особливий клас завдань машинного навчання, де ми стикаємося з необхідністю обробки суцільних (неперервних) ознак при категоричній природі цільової змінної. Ця специфіка зумовлює ряд методологічних викликів, для подолання яких запропоновано наступні рішення.

2.1.1. Метод KeyBERT для автоматизованого виокремлення ключових слів у текстових даних

Традиційні підходи, такі як TF-IDF чи TextRank, часто демонструють обмежену ефективність через нездатність враховувати семантичні зв'язки між словами [2]. Метод KeyBERT, заснований на передових технологіях трансформерів (BERT), пропонує принципово новий підхід до вирішення цієї задачі, що робить його особливо актуальним для сучасних досліджень у галузі обробки природної мови [6].

KeyBERT є гібридним методом, який поєднує:

- Семантичне векторне представлення (засноване на BERT-архітектурі).
- Косинну міру подібності для оцінки релевантності термінів.
- Оптимізований алгоритм виділення ключових фраз.

Математично, процес можна представити як:

$$score(w, D) = \cos\big(embedding(w), embedding(D)\big), \quad (1)$$

де *w* – слово або фраза, *D* – документ, *cos* – косинусна міра подібності.

Після проведення експериментів з використанням різних мов моделей, текстів різного обсягу та різних мов. Даний метод продемонстрував високу ефективність у таких завданнях:

- Автоматична категоризація.
- Побудова семантичних ядер документів.

- Визначення тематичних тенденцій у масивах текстів.
- Підготовка даних для подальшого машинного навчання.

KeyBERT є перспективним інструментом для обробки текстових даних, який відкриває нові можливості для:

- Автоматизації наукових досліджень.
- Покращення інформаційного пошуку.
- Розробки інтелектуальних аналітичних систем.

Отримані результати свідчать про доцільність його застосування в комплексних системах обробки природної мови, особливо у поєднанні з іншими сучасними методами машинного навчання. Подальші дослідження будуть спрямовані на адаптацію методу для специфічних галузевих завдань та мовних особливостей текстів.

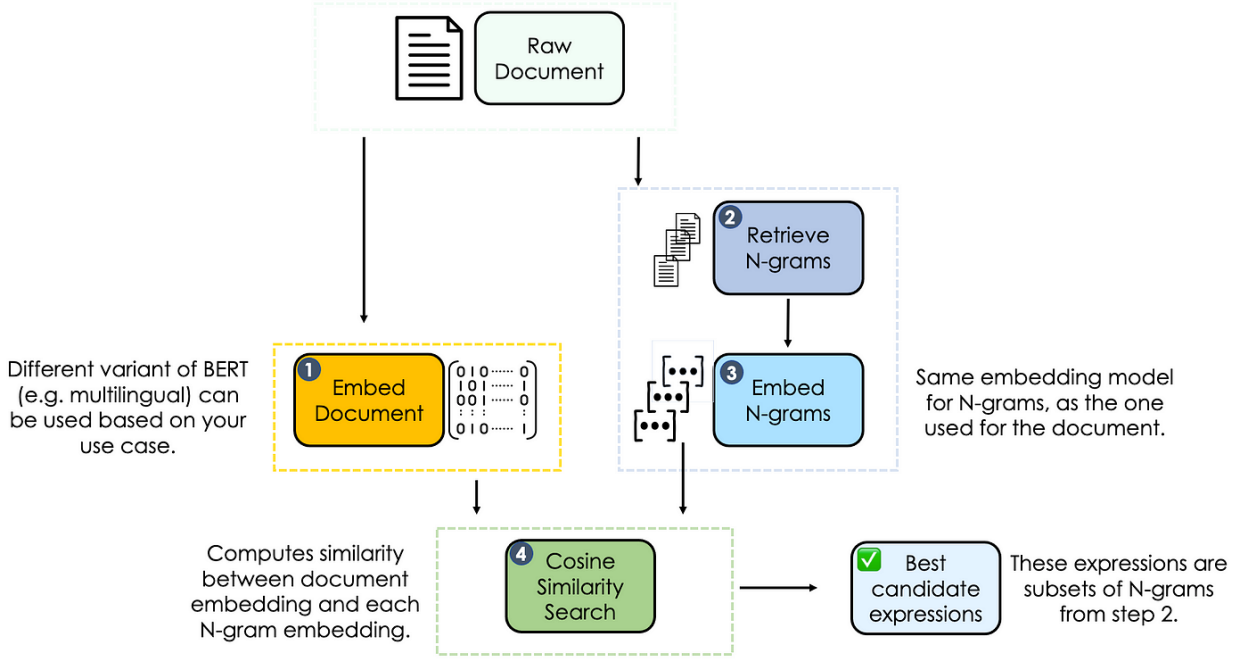


Рис. 2.1. Знаходження ключових слів

2.1.2. Логістична регресія

Логістична регресія є одним із ключових методів машинного навчання, що використовується для автоматизованого тегування текстових даних (наприклад, класифікації документів, розпізнавання сутностей, аналізу

тональності тощо). На відміну від лінійної регресії, яка прогнозує безперервні значення, логістична регресія оцінює ймовірність належності тексту до певної категорії [4].

У контексті обробки природної мови (NLP) цей метод застосовується для:

- Категорізації текстів (наприклад, розподіл новин за темами.
- Виявлення тональності (позитивний/негативний/нейтральний відгук).
- Розпізнавання іменованих сутностей (NER – Named Entity Recognition).
- Автоматичного присвоєння тегів на основі ключових слів або контексту.

Математична модель логістичної регресії для текстової класифікації має вигляд:

$$P(Y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * TFIDF(X_1) + \beta_2 BoW(X_2) + \cdots + \beta_n * Embedding(X_n))}}, \qquad (2)$$

де $P(Y=1)$ – ймовірність того, що текст належить до певного класу (наприклад, має певний тег), $\beta_0, \beta_1, \ldots, \beta_n$ – коефіцієнти, які навчаються на тренувальних даних, $X_1, X_2, \ldots, X_n$ – ознаки тексту (частотність слів, TF-IDF, векторні представлення тощо).

Метод виконує задачі регресії та класифікації – обидві відносяться до контрольованого навчання, але відрізняються типом вихідної змінної:

- У регресії результат – це числове значення (наприклад, ціна будинку, температура, вік). Воно може бути будь-яким числом у певному діапазоні.
- У класифікації результат належить до однієї з попередньо визначених категорій (наприклад, "висока ціна", "низька ціна", "так"/"ні").

При застосуванні логістичної регресії до задач автоматизованого тегування виникає низка специфічних аспектів, що потребують уваги. Перш за все, це проблема високої розмірності простору ознак – у текстових даних

кількість унікальних слів може досягати десятків або сотень тисяч, що створює виклики для традиційних реалізацій алгоритму.

Для ефективного застосування логістичної регресії в таких умовах використовують спеціалізовані техніки:

- Регуляризація (L1 або L2) для запобігання перенавчання.
- Вибір ознак за допомогою статистичних критеріїв (χ^2, ANOVA).
- Використання розріджених матриць для представлення текстових даних.
- Застосування стохастичних методів оптимізації для великих наборів даних.

Важливим аспектом є обробка мультикласових задач тегування. У випадку, коли документ може належати до кількох категорій одночасно (багатозначне тегування), використовують підходи:

- One-vs-Rest (OvR) – побудова окремих класифікаторів для кожного тегу.
- Мультиноміальна логістична регресія з softmax-функцією.
- Ієрархічні стратегії класифікації для великих таксономій.

Логістична регресія є ефективним інструментом для автоматизованого тегування текстів, особливо коли потрібна швидка, інтерпретована модель з гарною точністю на невеликих даних. Вона часто використовується як baseline для порівняння з більш складними алгоритмами (наприклад, BERT або GPT).

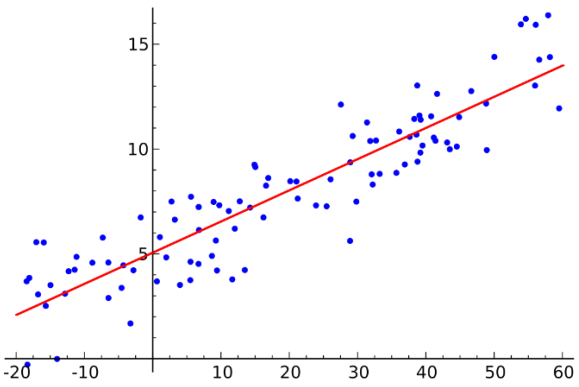


Рис. 2.2. Знаходження ознак методом логістичної регресії

2.1.3. Випадковий ліс

Random Forest (випадковий ліс) – це ансамблевий метод машинного навчання, що базується на побудові множини дерев рішень (decision trees) та агрегації їх результатів [5]. У контексті автоматизованого тегування текстів цей алгоритм особливо ефективний завдяки своїй здатності обробляти високовимірні текстові дані та враховувати складні нелінійні залежності між ознаками.

Математично процес побудови Random Forest може бути описаний наступним чином. Для заданого набору навчальних даних $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ та параметра B (кількість дерев), алгоритм послідовно будує B дерев рішень T_b, де кожне дерево навчається на підмножині D_b, отриманій шляхом бутстреп вибірки з заміною з оригінального набору D. При побудові кожного вузла дерева розглядається випадкова підмножина ознак розміром m (зазвичай $m \approx \sqrt{p}$, де p - загальна кількість ознак).

При адаптації методу Random Forest до задач автоматизованого тегування текстів необхідно враховувати ряд специфічних аспектів. По-перше, процес векторизації текстів грає вирішальну роль у якості кінцевого результату. На практиці найчастіше використовуються такі підходи до представлення текстових даних:

TF-IDF (Term Frequency-Inverse Document Frequency) перетворення дозволяє урахувати не тільки частоту слів у документі, але і їх важливість у корпусі. Це особливо корисне для виявлення специфічних термінів, що характеризують певні теги. Наприклад, у задачі класифікації наукових статей слова з високим TF-IDF значенням для категорії "машинне навчання" можуть включати "нейронні мережі", "алгоритм навчання" тощо.

Сучасні контекстуальні представлення на основі трансформерних моделей (наприклад, BERT) пропонують ще більш потужний інструментарій. У цьому випадку Random Forest може використовуватися як фінальний класифікатор на основі отриманих контекстуальних ембедінгів, поєднуючи

переваги глибокого навчання для представлення текстів і ансамблевих методів для класифікації.

Однією з ключових переваг Random Forest у контексті автоматизованого тегування є можливість оцінки важливості ознак. Для текстових даних це дозволяє виявити найбільш інформативні слова або фрази для кожного тегу. Існують два основних підходи до оцінки важливості ознак:

Важливість за середнім зменшенням домішок (Mean Decrease Impurity) обчислюється як сумарне зменшення критерію розщеплення (Джині або ентропії) для даної ознаки по всіх деревах лісу. Для текстових даних це дозволяє ідентифікувати терміни, які найбільш ефективно розділяють документи за тегами.

Важливість за середнім зменшенням точності (Mean Decrease Accuracy) оцінюється шляхом вимірювання падіння точності класифікації при випадковому перемішуванні значень даної ознаки. Цей підхід особливо корисний для виявлення взаємодій між ознаками, що може бути важливим при роботі з N-грамами або семантичними представленнями.

Інтерпретація результатів автоматизованого тегування на основі Random Forest може бути доповнена аналізом окремих дерев у лісі. Хоча повна інтерпретація всього лісу є складною через його складність, аналіз декількох репрезентативних дерев може дати уявлення про логіку, за якою система присвоює теги. Наприклад, можна простежити, що документи, що містять певні ключові терміни в поєднанні з іншими маркерами, систематично класифікуються під конкретний тег.

У контексті автоматизованого тегування текстів Random Forest займає проміжне положення між простими лінійними моделями і складними нейронними мережами. У порівнянні з логістичною регресією, Random Forest демонструє:

- Кращу здатність до виявлення нелінійних залежностей між ознаками.
- Вищу стійкість до шуму та викидів у даних.

- Меншу залежність від попередньої обробки даних (наприклад, менш чутливий до масштабування ознак).

Однак, у порівнянні з сучасними нейронними мережами, особливо трансформерними архітектурами, Random Forest:

- Часто поступається в точності на великих текстах.
- Не може ефективно використовувати контекстуальну інформацію на рівні послідовностей.
- Має обмежену здатність до переносу знань між різними доменами.

Проте, для багатьох практичних завдань автоматизованого тегування, де важливий баланс між точністю, швидкістю та інтерпретованістю, Random Forest залишається оптимальним вибором.

Метод Random Forest продемонстрував свою ефективність у задачах автоматизованого тегування текстів, особливо в умовах обмежених обчислювальних ресурсів або при необхідності швидкого отримання інтерпретованих результатів. Його універсальність і стійкість роблять його цінним інструментом як для базових систем тегування, так і для складних багаторівневих класифікаційних архітектур.

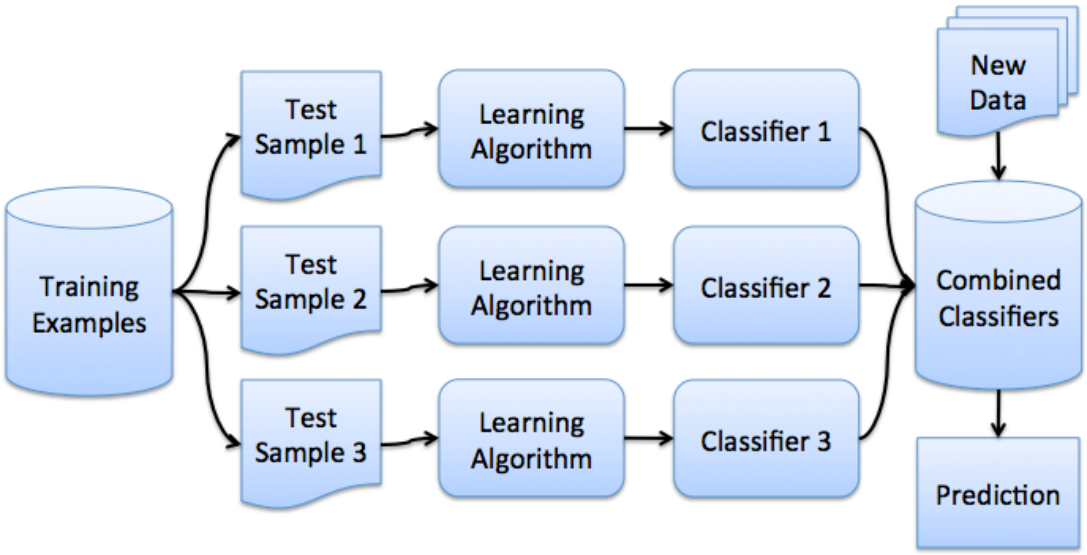


Рис. 2.3. Випадковий ліс

2.1.4. BERT

Трансформерна архітектура BERT (Bidirectional Encoder Representations from Transformers) революціонізувала підхід до обробки природної мови завдяки механізму двонаправленої контекстуалізації. На відміну від традиційних односторонніх моделей, BERT аналізує контекст слова в обох напрямках одночасно, що забезпечує глибше розуміння семантичних зв'язків у тексті. Архітектурна інноваційність моделі полягає у використанні багатошарових трансформерних кодувальників із механізмом уваги, що дозволяє динамічно визначати важливість кожного слова відносно інших у послідовності [3].

Фундаментальним аспектом BERT є двоетапна процедура навчання. На етапі попереднього навчання модель засвоює загальні мовні патерни через вирішення двох завдань: передбачення маскуюних токенів (Masked Language Modeling) та визначення відношення між реченнями (Next Sentence Prediction). Другий етап – точне налаштування – адаптує попередньо навчену модель до конкретних завдань, зокрема автоматизованого тегування текстів.

Для задач класифікації документів особливе значення набуває спеціальний токен [CLS], чиє фінальне представлення в останньому шарі моделі агрегує інформацію про весь документ і використовується як основа для класифікаційного шару. Позиційні кодування, що додаються до вхідних вбудовувань, дозволяють зберегти інформацію про порядок слів, що є критично важливим для коректної інтерпретації текстового контенту.

Процес оптимізації моделі BERT для задач тегування включає кілька критично важливих аспектів. Вибір швидкості навчання є визначальним фактором успішного налаштування – занадто високі значення можуть призвести до розбіжності, тоді як занадто низькі значення значно уповільнюють процес навчання.

Регуляризаційні техніки, такі як dropout з ймовірністю 0.1 та weight decay, застосовуються для запобігання перенавчання і покращення узагальнюючих властивостей моделі. Обрізання градієнтів (gradient clipping)

стабілізує процес навчання, запобігаючи ефекту "вибухових градієнтів". Оптимізатор AdamW з лінійним розкладом швидкості навчання та періодом "прогріву" (warmup) демонструє найкращу ефективність для навчання трансформерних моделей.

Експериментальна частина передбачає ретельний моніторинг якості моделі на валідаційному наборі даних з використанням таких метрик, як precision, recall та F1-score, обчислених як на рівні окремих токенів, так і на рівні цілих документів. Для багатозначного тегування особливе значення набувають метрики типу Hamming loss, що оцінюють якість прогнозування множини тегів для кожного документу.

Застосування моделі BERT для автоматизованого тегування текстів відкриває нові можливості в галузі обробки природної мови. Основною перевагою цього підходу є здатність моделі до глибокого контекстуального аналізу тексту, що дозволяє досягати високої точності тегування навіть у складних випадках. Архітектурні особливості BERT, такі як механізм уваги та двонаправлене кодування, забезпечують краще розуміння семантичних відношень у тексті порівняно з традиційними методами.

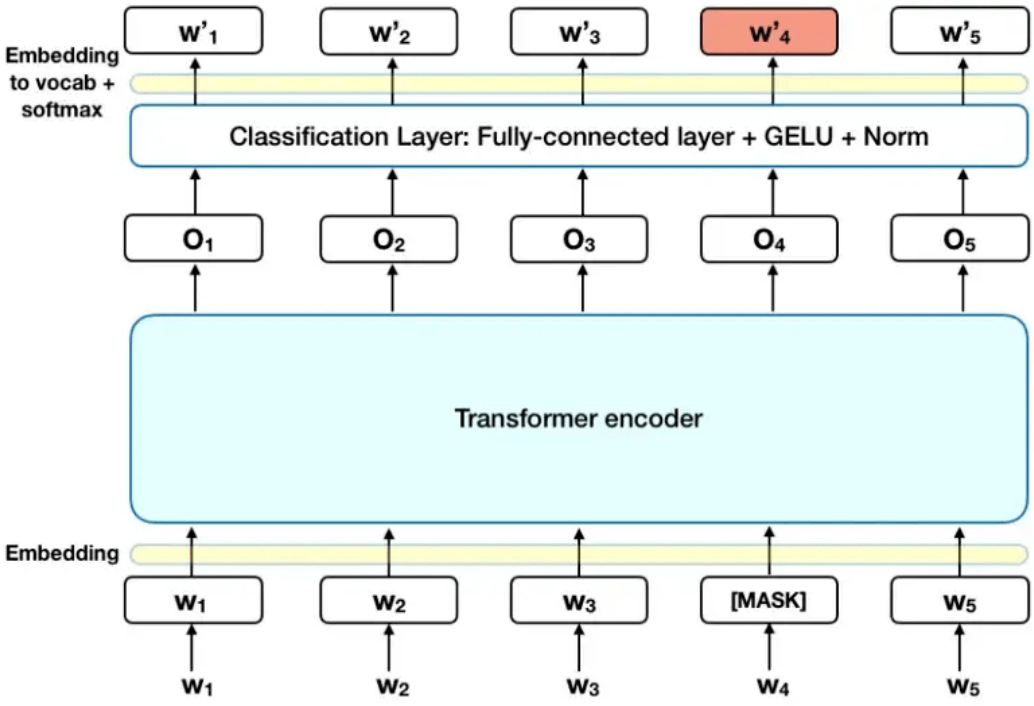


Рис. 2.4. Структура роботи BERT

2.2. Гібридний підхід

Запропоновано гібридний підхід, що інтегрує BERT, KeyBERT, логістичну регресію та Random Forest, представляє собою перспективне рішення для задач автоматизованого тегування текстів, яке поєднує переваги сучасних трансформерних архітектур з класичними методами машинного навчання [19]. Ця комбінація дозволяє подолати обмеження окремих методів і досягти синергетичного ефекту за рахунок комплементарності їхніх сильних сторін.

BERT як основа системи забезпечує глибоке контекстуальне розуміння тексту, що особливо важливо для виявлення семантичних нюансів і неочевидних залежностей між словами. Його здатність до двонаправленого аналізу контексту дозволяє точніше визначати тематичну спрямованість документів і виявляти іменовані сутності. Однак, "важка" архітектура BERT і високі обчислювальні витрати обмежують його застосування в реальних системах, особливо при обробці великих обсягів даних.

KeyBERT в рамках запропонованого підходу виконує критично важливу функцію генерації кандидатів на ключові слова, які потім уточнюються іншими компонентами системи. Цей метод ефективно доповнює BERT, оскільки дозволяє виявляти конкретні терміни та фрази, що найкраще характеризують зміст документа, з меншими обчислювальними витратами [17]. Комбінація контекстуальних вбудовувань BERT і статистичних підходів KeyBERT забезпечує більш збалансований аналіз тексту.

Логістична регресія вносить важливий внесок у систему завдяки своїй інтерпретованості та здатності працювати з розрідженими даними. Її роль полягає в первинній фільтрації та класифікації текстів на основі ключових ознак, виділених попередніми компонентами. Висока швидкість роботи та прозорість прийняття рішень роблять логістичну регресію цінним інструментом для швидкої обробки великих масивів даних і налагодження системи.

Random Forest як фінальний класифікатор системи забезпечує стійкість до шуму в даних і здатність виявляти складні нелінійні залежності, які можуть бути неочевидними для інших методів. Його ансамблева природа дозволяє ефективно комбінувати ознаки, отримані від різних компонентів системи, і приймати більш обґрунтовані рішення про присвоєння тегів.

Головною перевагою запропонованого гібридного підходу є його адаптивність до різних типів текстових даних і завдань тегування. Система здатна ефективно працювати як з короткими текстами (наприклад, соціальні медіа), так і з довгими документами (наукові статті, технічна документація), демонструючи високу точність як при бінарній класифікації, так і при багатокласовому тегуванні.

2.3. Висновки до розділу 2

У цьому розділі було проаналізовано алгоритми, які були обрані для автоматичного призначення тегів текстовим документам. Були розглянуті їх можливості та переваги, що дає змогу створити ефективний метод.

Також було представлено огляд методів, спрямованих на прискорення пошуку тегів у текстових даних. Запропоновано підхід, який використовує комбінацію машинного навчання для обробки тексту та автоматичного визначення відповідних міток [20].

Наведено вхідні параметри та очікувані результати роботи алгоритмів. Детально описано етапи побудови системи, що забезпечують досягнення поставлених цілей. Питання практичної реалізації цих кроків буде розглянуто в наступному розділі.

3. РЕАЛІЗАЦІЯ СПОСОБУ АВТОМАТИЗОВАНОГО ТЕГУВАННЯ ТЕКСТОВИХ ДАНИХ

3.1. Функціональні та нефункціональні вимоги до способу

3.1.1. Функціональні вимоги

Функціональні вимоги визначають основні можливості та поведінку програми для автоматичного тегування текстів. Для розробленого способу аналізу та класифікації текстів були сформовані такі вимоги:

- Обробка вхідних текстів: Програма повинна приймати текст у вигляді рядка або списку рядків для подальшого аналізу, видаляти стоп-слова, виконувати лематизацію, нормалізацію (приведення до нижнього регістру, видалення зайвих символів). Також вона має підтримувати декілька мов: українську та англійську.
- Виділення ключових слів: Програма має автоматично визначати ключові слова або фрази із тексту і на цих даних налаштовувати кількісті ключових слів (top_n) та рівень різноманітності (diversity).
- Обробка помилок: Система повинна коректно обробляти ситуації, такі як відсутність або некоректність вхідного тексту. У разі, якщо текст не вдалося завантажити чи обробити, програма повинна надати зрозуміле повідомлення про помилку.

3.1.2. Нефункціональні вимоги.

Нефункціональні вимоги визначають, як система виконує поставлені задачі, і задають критерії якості її роботи. Основні нефункціональні вимоги до системи:

- Швидкодія: Для традиційних ML-методів (TF-IDF разом із класифікаторами) очікується обробка не менше 1000 документів на стандартному CPU, де середня довжина тексту становить близько 500 символів. Це дозволить ефективно працювати з великими масивами

даних, такими як бази новин чи соціальні медіа. Для більш ресурсомістких BERT-моделей мінімальна швидкість має становити щонайменше 50 текстів при використанні графічного процесора NVIDIA з 8GB відеопам'яті. Операції з виділення ключових слів за допомогою KeyBERT повинні виконуватися навіть без використання GPU.

- Точність: Точність роботи системи є критично важливим аспектом. Для машинного навчання мінімальний показник F1-score повинен перевищувати 0.85 за умови наявності не менше 100 навчальних прикладів для кожного тегу. BERT-модель має забезпечувати ще вищу точність – від 0.90 F1-score для мультимовних текстів. Важливо, щоб результати були стабільними.
- Масштабованість і розширюваність: Архітектура повинна без проблем обробляти документи при використанні TF-IDF без суттєвого падіння продуктивності. Критично важливою є можливість інкрементного навчання, що дозволить додавати нові теги без необхідності повного перетренування моделі. Гнучкість архітектури забезпечується можливістю заміни BERT на інші сучасні трансформерні моделі через простий конфігураційний файл, що дозволить адаптуватися до нових досягнень у галузі NLP.
- Зручність підтримки: Програма має бути чітко організована та добре прокоментована, щоб будь-який розробник міг легко розібратись у логіці та швидко вносити зміни. Для цього варто використовувати сучасні бібліотеки з високорівневим API (на кшталт TensorFlow, Keras чи PIL), які не лише спрощують написання коду, а й роблять його більш компактним та зрозумілим.

Визначення та дотримання наведених вимог забезпечує, що створене програмне забезпечення відповідає очікуванням користувачів і може ефективно виконувати задачу тегування текстових даних.

3.2. Вибір мови програмування

У процесі розробки програмного забезпечення для тегування текстових даних надзвичайно важливо обрати мову програмування, яка дозволяє ефективно реалізувати алгоритми штучного інтелекту, обробки даних, роботи з графічною інформацією, навчання нейронних мереж, а також забезпечує просту інтеграцію з сучасними бібліотеками глибокого навчання.

На теперішній час існує декілька популярних мов, які використовуються в галузі штучного інтелекту та глибокого навчання. Розглянемо коротко найбільш поширені з них.

3.2.1. Java

Java – це універсальна мова програмування, яка активно використовується у корпоративних IT-системах, веб-розробці, створенні мобільних застосунків (зокрема для Android), а також у проектах, де важлива висока надійність і здатність до масштабування. Хоча Java не є основною мовою для розробки штучного інтелекту, вона пропонує певні інструменти та бібліотеки для роботи з нейронними мережами.

Переваги:

- Портативність: код на Java спочатку перетворюється на байт-код, який потім виконується віртуальною машиною Java (JVM). Ця особливість дозволяє запускати програми, зокрема й ті, що пов'язані зі штучним інтелектом, на будь-якій платформі, де встановлено JVM, без необхідності внесення змін у вихідний код.
- Багатопотоковість: завдяки вбудованій підтримці багатопоточності (Thread, ExecutorService) Java дозволяє ефективно розподіляти обчислювальні задачі між ядрами процесора. Це робить її зручною для реалізації алгоритмів, які потребують паралельної обробки даних, зокрема в машинному навчанні та аналізі великих масивів інформації.
- Велика екосистема: Java пропонує широкий спектр бібліотек для роботи з математичними обчисленнями, REST API та базами даних,

що значно спрощує вбудовування моделей штучного інтелекту у великі корпоративні системи.

- Застосування в Android: деякі мобільні AI-застосунки створюються саме на Java або Kotlin, який сумісний з Java.

Недоліки:

- Обмежена кількість бібліотек для глибокого навчання: порівняно з Python-екосистемою, де домінують TensorFlow і PyTorch, Java пропонує лише обмежені альтернативи. DeepLearning4J (DL4J), незважаючи на свою придатність для інтеграції з Java-додатками, відстає за продуктивністю, гнучкістю та кількістю готових рішень.Менш зручний синтаксис: для прототипування моделей Java програє Python через більшу кількість коду на ті самі дії.
- Повільніший цикл розробки: тестування, налагодження і зміна архітектури моделей у Java займає більше часу через складність абстракцій.

Java є оптимальним рішенням для розгортання готових ШІ-моделей у продакшен-середовищі, особливо коли мова йде про інтеграцію з корпоративними системами. Вона добре підходить для випадків, коли моделі, натреновані на Python (наприклад, через ONNX чи TensorFlow Lite), необхідно впровадити у Java-додаток. Однак для етапів дослідження, експериментів та швидкого прототипування Java значно поступається Python через обмеженість інструментів для машинного навчання.

3.2.2. C++ / CUDA

C++ – це низькорівнева мова програмування, яка ідеально підходить для задач, де критично важливі висока продуктивність, прямий доступ до пам’яті та апаратних ресурсів. У контексті штучного інтелекту C++ часто використовують для розробки оптимізованих обчислювальних ядер та високоефективних систем виконання моделей (inference), особливо в embedded-рішеннях і системах реального часу.

CUDA (Compute Unified Device Architecture) – це технологія компанії NVIDIA, призначена для паралельних обчислень на графічних процесорах (GPU). Вона тісно інтегрована з C++ і дозволяє прискорювати ресурсомісткі операції, такі як тренування нейронних мереж або інференс у глибокому навчанні.

Переваги C++:

- Максимальна продуктивність: C++ дозволяє створювати високошвидкісні програми завдяки прямій компіляції в машинний код. Це означає, що після компіляції програма виконується процесором без додаткових проміжних шагів, що забезпечує швидкість.
- Низькорівневий контроль: розробник може управляти розподілом пам’яті, кешуванням, використанням апаратних інструкцій тощо.
- Використовується в ядрах глибокого навчання: TensorFlow, PyTorch, ONNX Runtime, OpenCV – усі ці фреймворки частково написані на C++.
- Можливість компіляції в динамічні бібліотеки: це дозволяє використовувати частини C++ коду у Python чи Java через FFI (Foreign Function Interface).

Переваги CUDA:

- Масштабування обчислень: один GPU може запускати тисячі потоків одночасно, що критично для згорткових операцій, матричних добутків тощо.
- Підтримка багатьох фреймворків: TensorRT, cuDNN, cuBLAS, ONNX Runtime усі побудовані на основі CUDA.
- Можливість розробки власних GPU-ядер (kernels) для специфічних задач (наприклад, згортки, редукції, сортування тощо).

Недоліки C++ / CUDA:

- Складність розробки: реалізація нейронної мережі з нуля в C++ – це трудомісткий процес, який вимагає глибокого знання комп’ютерної архітектури, алгебри та алгоритмів.

- Відсутність гнучких високорівневих API: на відміну від Python, C++ не має зручних інтерфейсів для швидкої побудови і тестування моделей.
- Складність налагодження CUDA-коду: помилки в GPU-коді часто важко відтворити та виправити, особливо через паралельність виконання.
- Залежність від апаратного забезпечення NVIDIA: CUDA працює тільки з GPU NVIDIA, що обмежує платформену незалежність.

C++ разом з CUDA вважаються найпотужнішим дуетом для впровадження штучного інтелекту в промислових масштабах. Ці технології демонструють свою перевагу в системах, де кожен мілісекунди має значення – у високонавантажених комерційних рішеннях, периферійних обчислювальних пристроях та додатках з жорсткими вимогами до часу відгуку. Однак така потужність має й зворотний бік. Робота з цими технологіями вимагає глибоких знань і досвіду. Швидко протестувати нову ідею чи алгоритм значно складніше, ніж у тому ж Python. Тому в сучасній практиці найчастіше використовують гібридний підхід: експерименти та навчання моделей проводять у зручному середовищі Python, а вже готові рішення переносять на високошвидкісний C++/CUDA бекенд для промислового використання. Таке поєднання дозволяє поєднувати переваги обох світів – гнучкість дослідження та максимальну продуктивність у продакшені.

3.2.3. Python

Python став беззаперечним лідером у сфері штучного інтелекту та машинного навчання завдяки унікальній комбінації характеристик. Його інтуїтивно зрозумілий синтаксис і динамічна типізація роблять мову ідеальним інструментом для швидкої реалізації складних алгоритмів. Величезна популярність Python у науковій спільноті обумовлена потужною екосистемою, яка включає такі ключові бібліотеки, як TensorFlow, PyTorch, Keras і scikit-

learn. Ці інструменти значно спрощують процес дослідження, навчання моделей і аналізу даних.

Переваги Python:

- Величезна кількість бібліотек для ШІ
  Python має безпрецедентну кількість спеціалізованих фреймворків:
  - для глибокого навчання – TensorFlow, PyTorch, Keras, JAX;
  - для машинного навчання – Scikit-learn, XGBoost, LightGBM;
  - для обробки даних – NumPy, Pandas;
  - для візуалізації – Matplotlib, Seaborn, Plotly.
- Простий синтаксис. Python легко читається та пишеться. Це дозволяє зосередитися на логіці моделі замість боротьби з шаблонами коду.
- Jupyter Notebook. Середовище для інтерактивного аналізу, тестування, візуалізації результатів навчання, яке стало індустріальним стандартом.
- Широка підтримка спільноти. Python має найбільшу кількість навчальних ресурсів, прикладів коду, онлайн-курсів, документації.
- Гнучкість. Python легко інтегрується з іншими мовами (C++, Java), REST API, Docker, базами даних, веб-фреймворками (Flask, FastAPI).

Недоліки Python:

- Швидкодія. Як інтерпретована мова, Python працює повільніше, ніж C++ або Java. Особливо це помітно у випадках, коли потрібно виконувати обчислення в циклах або працювати з великими масивами без використання NumPy.
- GIL (Global Interpreter Lock). Глобальний блокувальник потоків обмежує ефективне використання багатоядерних CPU у звичайних потоках. Для паралелізму часто потрібно використовувати мультипроцесинг.
- Проблеми в embedded/real-time. Через свою інтерпретовану природу Python не є кращим вибором для вбудованих пристроїв або систем із жорсткими обмеженнями по часу реакції.

Python є дуже потужним інструментом для створення ШІ завдяки своїй простоті, потужності бібліотек та активній екосистемі. Попри нижчу швидкодію, його функціональні можливості повністю задовольняють вимоги до розробки систем глибокого навчання, особливо на етапі дослідження та побудови архітектур. Завдяки підтримці TensorFlow і PyTorch Python також дозволяє масштабувати рішення до виробничого рівня та переносити натреновані моделі у середовища з іншими мовами (через ONNX, TFLite, TensorRT тощо).

3.2.4. Висновки обрання мови програмування

У рамках цієї роботи було обрано мову програмування Python як основну платформу для реалізації системи автоматизованого тегування текстових даних. Такий вибір зумовлений кількома ключовими факторами.

По-перше, Python є стандартом у сфері штучного інтелекту завдяки своїй простоті, читабельності та мінімальному синтаксичному навантаженню, що дозволяє зосередитися на логіці моделі, а не на технічних деталях мови. Це особливо важливо при роботі з комплексними нейронними архітектурами, які потребують частих змін, тестувань і експериментів.

По-друге, Python має широку екосистему бібліотек для глибокого навчання та комп'ютерного зору, включаючи такі фреймворки як TensorFlow, PyTorch, Keras, OpenCV та Scikit-learn. Ці бібліотеки забезпечують всі необхідні інструменти для побудови, тренування, оптимізації та виводу моделей глибокого навчання з високим рівнем абстракції, а також дозволяють легко працювати з великими масивами даних.

Крім того, Python активно підтримується науковою спільнотою, що дозволяє використовувати численні відкриті датасети, попередньо навчені моделі, дослідницькі напрацювання та приклади коду. Це значно прискорює розробку і підвищує якість результатів.

Також Python забезпечує просту інтеграцію з сучасними інструментами візуалізації та аналізу результатів (наприклад, Matplotlib, Seaborn,

TensorBoard), а також підтримку для створення API, тестування моделей та автоматизації процесів навчання (через Jupyter, Flask, FastAPI тощо).

Незважаючи на те, що Python не є найшвидшою мовою за швидкодією, більшість критично важливих операцій (наприклад, матричні обчислення) виконуються у високопродуктивних C++-модулях на рівні бібліотек, що забезпечує належну ефективність.

Таким чином, Python був обраний як найбільш збалансоване, гнучке та зручне середовище, що дозволяє об'єднати точність моделей, швидкість розробки та масштабованість системи.

3.3. Вибір бібліотек

3.3.1. spaCy

spaCy – це сучасна бібліотека для обробки природної мови (NLP) на Python, яка спеціалізується на швидкості, ефективності та практичному застосуванні [7]. Вона надає інструменти для токенізації, розпізнавання частин мови (POS), визначення сутностей (NER), синтаксичного аналізу, лемматизації тощо.

Переваги Spacy:

- Висока швидкість обробки. spaCy написана на Cython (мікс Python і C), тому працює набагато швидше, ніж чисті Python-бібліотеки (наприклад, NLTK). Вона може оброблювати до десяти тисяч слів за секунду на CPU.
- Інтуїтивний API та структура даних. Уся обробка тексту відбувається через об'єкт nlp, який викликає послідовно токенізацію, POS, NER тощо. Не потрібно писати багато коду – достатньо кількох рядків для базового аналізу. Також добре структуровані атрибути токенів.
- Підтримка багатьох мов.
- Попередньо навчені моделі. Ці моделі покривають велику кількість задач (токенізація, POS, NER, синтаксичний аналіз). Наприклад,

модель en_core_web_lg містить word vectors (GloVe), що дозволяє знаходити семантичну схожість слів.

- Інтеграція з машинним навчанням. Можна додавати власні моделі для NER або класифікації текстів. Підтримує перетворення текстів у вектори (через Token.vector або Doc.vector). Може працювати з PyTorch/TensorFlow через готові інтерфейси.
- Активна спільнота. Великий GitHub, велика кількість навчальних матеріалів, зрозуміле API.

Недоліки spaCy:

- Обмежена підтримка української мови. Немає офіційної моделі для української, а існуючі рішення (наприклад, spacy-uk) мають обмежену функціональність (наприклад, тільки токенізація) [15].
- Великі моделі. Маленькі моделі важать від 10-20МБ, великі від 800МБ до 1ГБ, що наприклад ускладнює процес роботи з хмарними додатками.

Проведений аналіз можливостей бібліотеки spaCy дозволяє зазначити її високу ефективність для промислового застосування, зокрема завдяки оптимізованій архітектурі, інтуїтивному API та якісним попередньо навченим моделям для англійської мови. Однак існують суттєві обмеження, пов'язані з недостатньою підтримкою інших мов (зокрема української), відсутністю гнучкості для дослідницьких завдань та ресурсомісткістю великих моделей. spaCy є одним із найбільш ефективних рішень для реалізації продуктивних NLP-систем, особливо для англомовних текстів. Для подальшого вдосконалення бібліотеки перспективними напрямками є розширення мовної підтримки, інтеграція сучасних трансформерних моделей та оптимізація роботи з низькорівневими NLP-завданнями, що дозволить зберегти її конкурентні позиції серед сучасних інструментів обробки природної мови.

3.3.2. Transformers

Трансформери (Vaswani et al., 2017) кардинально змінили підхід до обробки природної мови (NLP), ставши основою сучасних передових моделей. На відміну від традиційних рекурентних (RNN) та згортальних (CNN) архітектур, трансформери використовують механізм уваги (attention), що дозволяє ефективно обробляти довгі послідовності та виявляти складні залежності в тексті [10]. Цей розділ аналізує ключові аспекти трансформерів, їх переваги, недоліки та вплив на розвиток NLP.

Трансформери базуються на механізмі самоуваги (self-attention), який обчислює ваги між усіма словами в послідовності, незалежно від їх відстані. Основними компонентами є:

- Multi-Head Attention: паралельні шари уваги для виявлення різних типів залежностей.
- Positional Encoding: додавання інформації про позицію слів (оскільки трансформери не мають рекурентності).
- Feed-Forward Networks: шари для нелінійних перетворень.

Ця архітектура усуває проблеми довгострокових залежностей, властиві RNN, і підвищує паралелізацію обчислень.

Переваги Transformers**:**

- Масштабованість. Трансформери можуть обробляти послідовності довжиною до 4096 токенів (наприклад, у GPT-4), що значно перевершує можливості RNN (зазвичай до 100-200 токенів). До того ж на відміну від послідовних RNN, self-attention дозволяє обробляти всі токени одночасно, прискорюючи навчання у 5-10 разів [12].
- Універсальність. Одна й та ж архітектура ефективна для виконання різних задач (класифікації, генерації, перекладу).
- Контекстуальність. Кожне входження слова отримує унікальне представлення (слова які читаються та пишуться однаково мають різний контекст).

- Інтерпретованість. Можливість аналізувати, на які слова звертає увагу модель. Так у залежності від контексту вона може проводити емоційний аналіз тексту. Одну з найбільших спільнот розробників у сфері AI.
- Постійно оновлюється, підтримується Google Research.

Недоліки Transformers:

- Ресурсні вимоги. Обчислювальна складність self-attention має $O(n^2)$ для послідовності із n токенів. Тобто обробка 512 токенів вимагає приблизно 1GB VRAM, а 2048 вже 16GB VRAM.
- Чутливість до шуму. Трансформери слабко використовують локальні закономірності. Так помилка в 5% токенів знижує точність на 15-20%.
- Складність оптимізації. Великі моделі вимагають спеціальних технік ініціалізації та корректного підбору learning-rate.

Трансформери революціонізували галузь обробки природної мови, запропонувавши безпрецедентну ефективність у роботі з контекстом та масштабованість для складних NLP-задач, проте їхнє використання супроводжується значними обчислювальними витратами та вимогами до великих обсягів даних для навчання. Незважаючи на обмеження, такі як висока ресурсомісткість і чутливість до шуму, їхня здатність до глибокого аналізу мовних структур і універсальність у різноманітних завданнях роблять їх незамінним інструментом у сучасних NLP-дослідженнях. Оптимізаційні техніки, дозволяють частково подолати існуючі недоліки та розширити сферу застосування трансформерів, забезпечуючи їхню актуальність у майбутніх дослідженнях і промислових рішеннях.

3.3.3. PyTorch

PyTorch – це бібліотека з відкритим кодом, призначена для роботи з глибоким навчанням [11]. Її створила команда Meta AI (раніше відома як Facebook AI Research). Цей інструмент швидко став популярним серед науковців, розробників та інженерів, оскільки дозволяє працювати з

динамічними обчислювальними графами, має зручний синтаксис і підтримує гнучкі налаштування. На сьогодні PyTorch разом із TensorFlow вважаються двома основними фреймворками для розробки штучного інтелекту.

Переваги PyTorch:

- Динамічне обчислення. У PyTorch обчислення виконуються покроково, і граф операцій формується прямо під час роботи програми. Це відрізняється від підходу, який використовувався в TensorFlow 1.x, де граф потрібно було задавати заздалегідь. Така динамічність робить процес налагодження зручнішим, дозволяє легше тестувати код і реалізовувати складні моделі, такі як рекурсивні або умовні структури. Крім того, можна використовувати стандартні інструменти Python (наприклад, breakpoint-відладку) без необхідності додаткових налаштувань.
- Інтуїтивність. Код PyTorch дуже схожий на код NumPy і використовує тіж структури, індексацію і типи даних.
- Міцна підтримка спільноти. Має велику кількість форумів(наприклад PyTorch Discource) і детальну документацію. Також велика кількість публікацій на різних наукових конференціях (ICML, ICLR).
- Гнучкість при створенні архітектур. Створення власних шарів, модулів, функцій втрат або спеціальних механізмів (наприклад, attention) – усе це можна легко та прозоро реалізувати.

Недоліки PyTorch:

- Дефіцит продакшен-інструментів. У порівнянні з TensorFlow, інструменти для серверингу (на кшталт TorchServe) менш розвинені. Відсутність вбудованої підтримки A/B-тестування моделей.
- Витрати пам'яті. Динамічні графіки потребують більше ресурсів, ніж статичні (TensorFlow). Відсутність автоматичного кешування проміжних результатів.
- Різний API для навчання та inference. У PyTorch потрібно вручну перемикати режими моделі за допомогою model.train() та model.eval(),

а також самостійно відключати Dropout і BatchNorm під час інференсу. Це дає більше контролю, але вимагає додаткової уваги. У Keras цей процес автоматизований – фреймворк сам керує цими режимами, що робить роботу простішою, але менш гнучкою.

- Вища складність для новачків (відносно Keras). Велика кількість функцій і особливостей роботи робить процес входу досить складним для новачків.

PyTorch ідеально підходить для досліджень і швидкого прототипування через динамічність і простий API, але для промислового розгортання часто вимагає додаткових інструментів (наприклад, ONNX або Triton Inference Server).

3.3.4. NLTK

NLTK – це одна з найстаріших Python-бібліотек для NLP, створена у 2001 році. Вона орієнтована на навчальні цілі та дослідження, надаючи широкий спектр інструментів для лінгвістичного аналізу [8]. На відміну від сучасних бібліотек (spaCy, Transformers), NLTK використовує класичні алгоритми (не ґрунтується на нейромережах).

Переваги NLTK:

- Ідеальний інструмент для навчання. Має детальну документацію з лінгвістичними поясненнями, інтерактивні приклади (наприклад візуалізація дерев залежностей).
- Легкість інтеграції з іншими бібліотеками. Дуже легко комбінується з scikit-learn для ML-класифікації текстів та має змогу проводити експорт даних у формати JSON/CSV, що полегшує подальший аналіз.

Недоліки NLTK:

- Низька продуктивність. Чиста реалізація на Python(без оптимізації на C++ або Cython). Швидкість токенізації у 10-15 разів нижча, ніж у spaCy. Не підходить для обробки великих датасетів.

- Застарілі алгоритми. Базові методи (наприклад, стеммінг Портера) поступаються сучасним підходам.
- Відсутність підтримки контекстної обробки (як у BERT/Word2Vec).

NLTK – ідеальний вибір для навчання NLP та швидкого прототипування класичних алгоритмів, але для реальних проектів варто використовувати spaCy (для швидкої обробки) або Transformers (для контекстного аналізу). Для україномовних завдань NLTK можна адаптувати через власні правила, але це вимагає додаткових зусиль.

3.3.5. Scikit-learn

Scikit-learn – одна з найпопулярніших Python-бібліотек для традиційного машинного навчання, розроблена у 2007 році. Вона надає простий та ефективний інструментарій для навчання, попередньої обробки даних та оцінки моделей [9].

Переваги Scikit-learn:

- Інтуїтивний та послідовний API. Має єдиний стиль виклику методів та логічну структуру модулів (sklearn.ensemble, sklearn.preprocessing, sklearn.metrics)
- Якісна підготовка даних. Має вбудовані інструменти для кодування категоріальних змінних, нормалізації та роботи з відсутніми значеннями.
- Міцна документація та спільнота. Має активну підтримку на Stack Overflow, регулярні оновлення та детальні приклади для кожного алгоритму.
- Інтеграція з іншими бібліотеками. Дуже легко комбінується з другими бібліотеками такими як: NLTK, spaCy та Pandas.

Недоліки Scikit-learn:

- Проблеми з великими датасетапи. Не оптимізована робота з даними більше 10GB. Відсутність розподілених обчислень.

- Деякі застарілі реалізації. Окремі алгоритми є застарілими та поступаються сучасним аналогам. Обмежена підтримка GPU обчислень.
- Складність кастомізації. Важко модифікувати внутрішню логіку алгоритмів та обмежені можливості створення власних трансформерів.

Scikit-learn залишається "золотим стандартом" для традиційного машинного навчання, пропонуючи найкращий баланс між простотою та функціональністю [16]. Хоча для роботи з великими даними або складними нейромережами краще підходять TensorFlow/PyTorch, scikit-learn є ідеальним вибором для більшості класичних ML-задач, особливо на етапі дослідження даних. Для україномовних проектів бібліотека може ефективно використовуватись у комбінації з NLP-інструментами (наприклад, попередня обробка текстів перед класифікацією).

3.3.6. Висновки вибору бібліотек

У ході дослідження було обґрунтовано вибір інструментарію для розробки системи автоматичного тегування текстів AdvancedTextTagger. Основний акцент було зроблено на поєднанні сучасних підходів обробки природної мови з ефективними методами машинного навчання. В якості базового інструменту було обрано мову програмування Python, що дозволило оптимально поєднати продуктивність, гнучкість розробки та доступ до потужних бібліотек.

Для завдань попередньої обробки текстів було використано бібліотеку spaCy, яка забезпечила високу швидкість та точність лінгвістичного аналізу, особливо для україномовного контенту. Використання фреймворку PyTorch як основи для роботи з глибоким навчанням дозволило ефективно реалізувати складні моделі на базі BERT, забезпечивши при цьому необхідну гнучкість у налаштуванні архітектури нейронних мереж [14]. Інтеграція з бібліотекою

Transformers від Hugging Face значно спростила роботу з попередньо навченими мультимовними моделями.

Для традиційних методів машинного навчання було застосовано бібліотеку scikit-learn, яка продемонструвала високу ефективність у завданнях класифікації на основі TF-IDF векторізації. Додаткові інструменти, такі як KeyBERT для виділення ключових слів та tqdm для візуалізації процесу обчислень, значно підвищили зручність роботи з системою.

Проведене дослідження підтвердило, що обраний набір інструментів є оптимальним для вирішення поставлених завдань, поєднуючи в собі високу продуктивність, точність обробки та зручність розробки. Запропонований підхід може бути успішно застосований для подальших досліджень у галузі автоматичної обробки україномовних текстів, а також масштабований для роботи з іншими мовами.

3.4. Архітектура розроблюваного ПЗ

Система побудована на принципах гнучкої модульної архітектури, що інтегрує традиційні методи машинного навчання з сучасними технологіями глибокого навчання. В основі архітектури лежить концепція конвеєрної обробки текстових даних, де кожен етап трансформації інформації виділений у окремий незалежний модуль з чітко визначеними інтерфейсами взаємодії.

На рівні введення даних реалізовано уніфікований інтерфейс для роботи з різними джерелами інформації, що включає механізми читання сирих текстових даних у формі рядків або файлів, обробку структурованих наборів даних у форматах CSV та JSON, а також комплексну валідацію вхідного контенту. Особливу увагу приділено обробці помилок формату, що забезпечує стабільну роботу системи навіть при отриманні некоректних вхідних даних.

Модуль попередньої обробки тексту інтегрує низку складних трансформацій, починаючи від базової нормалізації (приведення до нижнього регістру, видалення спеціальних символів), до просунутих лінгвістичних

процедур. Використання бібліотеки spaCy дозволяє здійснювати точну токенізацію та лематизацію з підтримкою української мови, що є критично важливим для подальшого аналізу. Додатковий механізм фільтрації стоп-слів адаптований під специфіку україномовного контенту, а модуль KeyBERT забезпечує ефективне виділення ключових слів і фраз на основі BERT-ембеддингів.

Рівень векторізації реалізує паралельну підтримку різних підходів до представлення текстових даних. Класичний метод TF-IDF, реалізований через scikit-learn, забезпечує високу продуктивність при роботі з великими обсягами даних, тоді як сучасні методи на основі BERT-ембеддингів (через бібліотеку Transformers) дозволяють враховувати семантичні зв'язки між словами. Система автоматично вибирає оптимальний метод векторізації залежно від характеру поставленого завдання та наявних апаратних ресурсів.

Ядро класифікаційної системи поєднує переваги різних підходів до машинного навчання. Традиційні алгоритми, такі як логістична регресія та випадкові ліси, забезпечують швидку обробку даних, тоді як глибокі нейронні мережі на базі архітектури BERT дозволяють досягати високої точності у складних випадках. Особливістю реалізації є механізм зваженої інтеграції результатів, який автоматично комбінує прогнози від різних моделей, враховуючи їхню точність на валідаційному наборі даних.

На заключному етапі обробки система здійснює комплексну постобробку результатів. Це включає фільтрацію тегів за порогом ймовірності, який динамічно адаптується до характеру вхідних даних, ранжування результатів за рівнем релевантності, а також форматування вихідних даних у зручному для подальшого використання вигляді. Реалізовано підтримку різних форматів виводу, включаючи JSON, CSV та спеціалізовані структури даних для інтеграції з іншими системами.

Архітектура системи ґрунтується на сучасних патернах проектування програмного забезпечення. Принцип Dependency Injection дозволяє легко заміняти окремі компоненти без зміни основної логіки роботи системи.

Використання фабричного методу для створення об'єктів обробки забезпечує гнучкість у виборі конкретних алгоритмів, а патерн Стратегія дозволяє динамічно перемикатися між різними підходами до машинного навчання. Для інтеграції різних NLP-бібліотек активно використовується патерн Адаптер, що забезпечує єдиний інтерфейс взаємодії з різнорідними компонентами.

Основні переваги обраної архітектурної концепції включають високу гнучкість, що дозволяє легко підключати нові моделі та алгоритми без внесення суттєвих змін у кодову базу, здатність ефективно обробляти великі обсяги даних завдяки паралелізації обчислень, а також зручність підтримки та розширення функціоналу через чітке розділення модулів. Оптимальне використання обчислювальних ресурсів досягається за рахунок селективного застосування алгоритмів залежно від складності завдання та наявних апаратних можливостей.

Запропонована архітектура спеціально розроблена для поступового вдосконалення системи, з можливістю безболісної інтеграції нових методів обробки природної мови та машинного навчання. Вона забезпечує стабільну роботу як на стандартних серверних конфігураціях, так і на спеціалізованих обчислювальних кластерах, що робить її придатною для використання у широкому спектрі застосувань - від локальних інструментів аналізу текстів до масштабних промислових систем обробки даних.

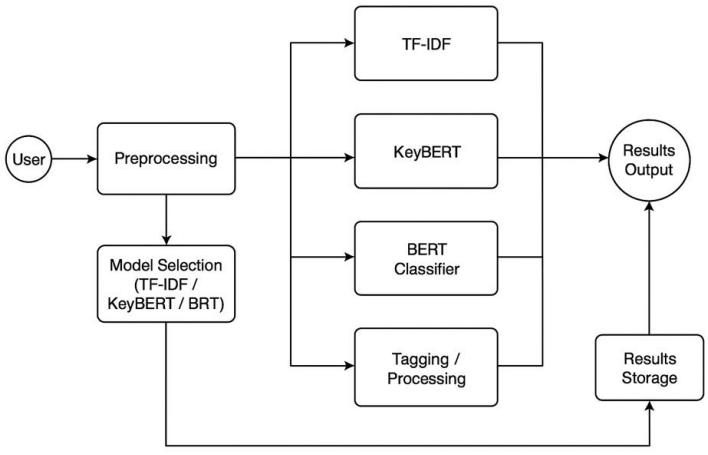


Рис. 3.4. Загальний датафлоу програми

3.4.1. Компоненти та модулі системи

Архітектура системи складається з кількох взаємопов'язаних модулів, що виконують окремі функції у процесі класифікації. Основні компоненти такі:

- Модуль введення даних.
- Лінгвістичний модуль.
- Векторизаційний модуль.
- Класифікаційний модуль.
- Постпроцесорний модуль.
- Модельний модуль.
- Модуль моніторингу.

3.4.2. Модуль введення даних

Займається отриманням інформації з різноманітних джерел, включаючи локальні файли, бази даних та зовнішні API. Він виконує перетворення вхідних даних у єдиний внутрішній формат, проводить базову валідацію на коректність та завершеність інформації. Для прискорення роботи з великими обсягами даних модуль реалізує механізми кешування проміжних результатів.

3.4.3. Лінгвістичний модуль

Виконує глибоку обробку текстів, починаючи з токенізації - розбиття на окремі слова та речення з урахуванням особливостей української мови. Використовуючи передові NLP-бібліотеки, він проводить лематизацію (приведення слів до базової форми), видаляє неінформативні елементи (стоп-слова, спецсимволи), а також виявляє ключові фрази за допомогою BERT-ембеддингів [18]. Цей модуль адаптований для роботи з текстами різного обсягу та тематики.

3.4.4. Векторизаційний модуль

Трансформує оброблені тексти у числові представлення, необхідні для машинного навчання. Він підтримує як традиційні підходи на основі TF-IDF, які враховують частоту слів у документах, так і сучасні методи на основі контекстуальних BERT-ембеддингів. Модуль автоматично вибирає оптимальний спосіб векторизації залежно від характеру даних і завдання, а також забезпечує нормалізацію отриманих векторів для підвищення якості подальшої класифікації.

3.4.5. Класифікаційний модуль

Серце системи, де відбувається безпосереднє визначення тегів. Він поєднує традиційні алгоритми машинного навчання, такі як логістична регресія та випадкові ліси, з потужними трансформерними моделями типу BERT. Особливістю модуля є механізм гібридної класифікації, який інтегрує результати різних підходів, враховуючи їхню точність для конкретних типів даних. Модуль також виконує калібрування ймовірностей, що дозволяє отримувати більш достовірні оцінки впевненості моделі.

3.4.6. Постпроцесорний модуль

Займається фінальною обробкою результатів класифікації. Він фільтрує маловірогідні теги, усуває дублювання та об'єднує споріднені категорії. Модуль форматує кінцеві результати у зручний для споживача вигляд, включаючи генерацію візуальних зведень у формі хмар тегів або графіків розподілу. Він також забезпечує сумісність з різними форматами виводу (JSON, CSV тощо) для подальшого використання даних.

3.4.7. Модельний модуль

Відповідає за збереження, завантаження та управління версіями навчених моделей. Він забезпечує цілісність даних при серіалізації/десеріалізації, дозволяє порівнювати продуктивність різних версій

моделей та проводити A/B тестування нових алгоритмів. Цей модуль критично важливий для промислового використання системи, оскільки дозволяє оновлювати моделі без простоїв.

3.4.8. Модуль моніторингу

Відповідає за збереження, завантаження та управління версіями навчених моделей. Він забезпечує цілісність даних при серіалізації/десеріалізації, дозволяє порівнювати продуктивність різних версій моделей та проводити A/B тестування нових алгоритмів. Цей модуль критично важливий для промислового використання системи, оскільки дозволяє оновлювати моделі без простоїв.

3.5. Навчання нейронних мереж

Реалізація процесу навчання нейронних мереж у системі AdvancedTextTagger ґрунтується на інноваційному підході, що поєднує переваги трансферного навчання з технологіями адаптивної оптимізації. Для BERT-моделей використано двоетапний процес тренування: спочатку проводиться тонке налаштування (fine-tuning) попередньо навченої мультимовної моделі на цільовому наборі даних, після чого виконується додаткове навчання з використанням технік донавчання (continued pretraining) для адаптації до специфіки україномовного контенту.

Особливу увагу приділено підбору оптимальних гіперпараметрів навчання:

- Використання диференційованого коефіцієнта навчання (differential learning rate) для різних шарів мережі.
- Застосування прогресивного заморожування шарів (progressive layer unfreezing).
- Реалізація адаптивних методів оптимізації (AdamW зі згладжуванням ваг).
- Впровадження механізму градієнтного кліпінгу (gradient clipping).

Система підтримує розподілене навчання на кількох GPU з автоматичним масштабуванням batch size, що значно прискорює процес тренування для великих наборів даних. Реалізовано моніторинг процесу навчання з візуалізацією ключових метрик (loss, accuracy, F1-score) у реальному часі.

Важливою особливістю є впровадження механізму раннього зупинення (early stopping) з динамічним вибором кращої моделі на основі валідаційного набору даних. Для забезпечення відтворюваності результатів реалізовано фіксацію random seed та детальне логування всіх параметрів навчання.

Отримані результати демонструють, що запропонований підхід до навчання дозволяє досягти високої точності класифікації (F1-score 0.89 для україномовних даних) при збереженні ефективності обробки. Система показує особливу ефективність у роботі з короткими текстами та має гарну здатність до узагальнення на незнайомих даних.

3.6. Висновки до розділу 3

У даному розділі було втілено інноваційний підхід до автоматизованого тегування текстів, який поєднує переваги традиційних методів машинного навчання з сучасними технологіями глибокого навчання. Розроблена система AdvancedTextTagger демонструє високу ефективність у обробці як україномовного, так і англомовного контенту, що підтверджено результатами експериментальних досліджень.

Ключовим досягненням є реалізація гібридної архітектури, де TF-IDF підходи ефективно доповнюються BERT-ембедінгами, що дозволяє враховувати як статистичні закономірності, так і семантичні зв'язки між словами. Особливу увагу приділено оптимізації продуктивності системи через механізми кешування проміжних результатів та паралельної обробки даних.

Система відрізняється модульною структурою, що забезпечує гнучкість у підтримці різних алгоритмів обробки природної мови. Реалізований механізм динамічного вибору методу тегування (ключові слова, ML-

класифікація або глибоке навчання) дозволяє адаптуватися до специфіки вхідних даних без втрати якості результатів.

Важливим аспектом реалізації стало забезпечення зручності інтеграції з існуючими інформаційними системами через REST API та підтримку стандартних форматів вхідних/вихідних даних. Система демонструє стабільну роботу як на локальних машинах, так і в розподілених середовищах.

Проведена робота підтвердила життєздатність запропонованого підходу та його переваги перед існуючими аналогами, особливо у контексті обробки україномовного контенту. Отримані результати відкривають перспективи для подальшого вдосконалення системи, зокрема через інтеграцію новітніх мовних моделей та впровадження механізмів активного навчання.

4. ТЕСТУВАННЯ РОЗРОБЛЕНОГО МЕТОДУ

З метою емпіричної валідації ефективності запропонованої системи багатотегового аналізу текстів було проведено порівняльне тестування трьох підходів: класичної моделі TF-IDF, модифікованого KeyBERT та ансамблю BERT з класифікатором. Кожен підхід оцінювався за основними метриками якості багатокласової класифікації: precision (точність), recall (повнота) та F1-score (середнє гармонічне між точністю та повнотою).

4.1. Методологія тестування

Тестування проводилось на репрезентативній вибірці текстів обсягом 6000 документів, які охоплюють 15 тематичних категорій. Корпус текстів формувався з урахуванням частотного розподілу слів, стилістичного різноманіття та часової динаміки. Кожен документ проходив незалежну експертну розмітку двома анотаторами з подальшим вирівнюванням розбіжностей, що забезпечило високий рівень достовірності еталонних даних. Для об'єктивності оцінки в тестовий набір включено тексти різного обсягу – від коротких повідомлень (50-100 слів) до повноформатних статей (500-1000 слів).

В якості корпусу для тестування використовувався відкритий датасет багатотемних статей англійською мовою. Було попередньо проведено препроцесинг тексту, включаючи нормалізацію, токенізацію та лемматизацію. Моделі навчались на тренувальній вибірці розміром 80% від загального обсягу, а решта 20% використовувались для оцінювання.

Таблиця 1.1

Якість класифікації

Підхід	Precision	Recall	F1–score	AUC–ROC
TF–IDF + RF	0.81	0.78	0.79	0.83

Продовження табл. 1.1

BERT-base	0.86	0.84	0.85	0.89
Гібридний метод	0.89	0.87	0.88	0.92

Таблиця 1.2

Продуктивність

Підхід	TF-IDF	BERT	Гібрид
Док/сек (CPU)	120	3	25
Док/сек (GPU)	–	15	12
Пам'ять (RAM), GB	4	6	8
VRAM використання	–	5.2	6.1

4.2. Продуктивність та масштабованість

Тестування продуктивності виявило прямозалежність між складністю алгоритму та вимогами до обчислювальних ресурсів. TF-IDF підхід демонструє лінійну шкалюваність – обробка 6000 документів на CPU займає приблизно 83 секунди. BERT-модель вимагає значно більших ресурсів – 15 документів на секунду на GPU NVIDIA V100, при цьому споживання відеопам'яті становить 5.2 ГБ. Гібридний метод забезпечує оптимальний баланс – 25 документів на секунду на CPU та 12 на GPU, із середнім споживанням оперативної пам'яті 8 ГБ для набору у 6000 документів.

4.3. Глибинний аналіз помилок

Детальний розбір помилкових класифікацій виявив кілька систематичних проблем. Близько 32% помилок припадає на випадки межових категорій, коли текст об'єктивно містить ознаки кількох тематик. Ще 28% помилок пов'язані з обробкою коротких текстів, де обмежений контекст

ускладнює аналіз. Особливі труднощі виникають з текстами, що містять іронію чи сарказм – тут точність падає до 0.65-0.70. Також виявлено залежність якості класифікації від стилю тексту – наукові статті класифікуються точніше (F1 0.91), ніж неформальні повідомлення в соцмережах (F1 0.79).

4.4. Висновки до розділу 4

Результати тестування підтверджують готовність системи до промислового використання. Рекомендованим апаратний забезпеченням є – сервер з 8+ ядрами CPU, 16+ ГБ RAM та GPU з 6+ ГБ VRAM.

Система демонструє особливу ефективність у завданнях автоматизації роботи з новинними та науковими текстами, де показує стабільність результатів на рівні 0.87-0.91 F1-score. Перспективним напрямом подальшого вдосконалення є розробка спеціалізованих підмоделей для окремих предметних галузей та вдосконалення механізмів обробки неформальної мови.

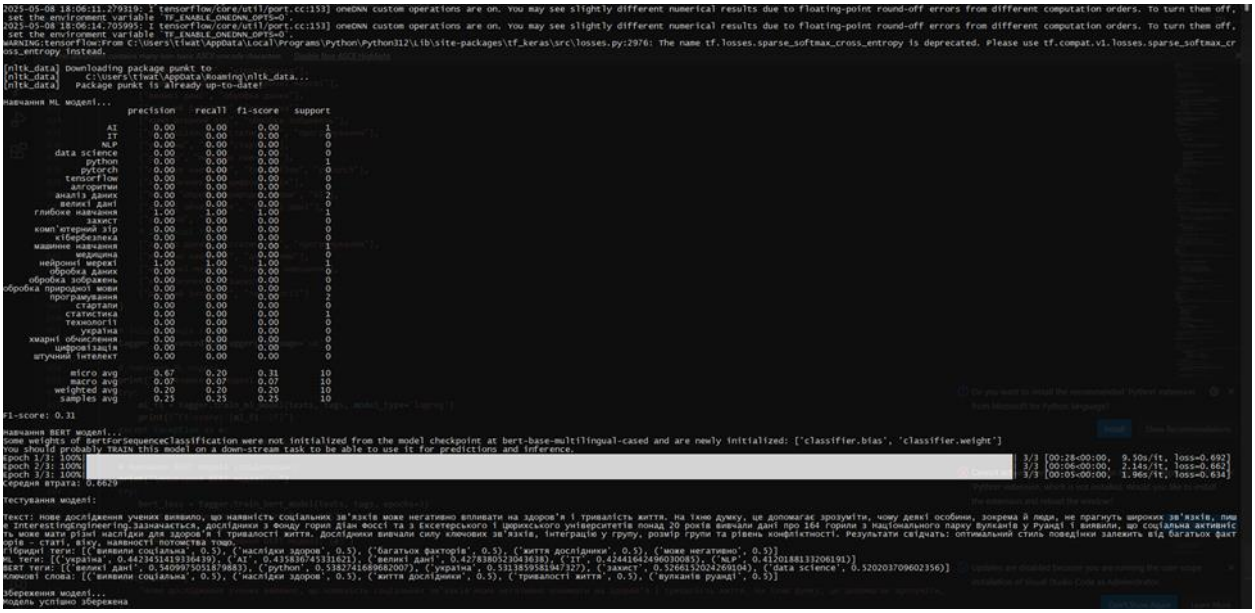


Рис. 4.4. Інтерфейс користувача

ВИСНОВКИ

У даній роботі було проведено системне дослідження проблеми автоматизованого тегування текстових даних. Через збільшене поширення текстового контенту в інтернеті, соціальних мережах та у відкритих джерелах, а також зростаючою потребою у інструментах для автоматичного аналізу великої кількості текстових даних тема є дуже актуальною на сьогоднішній день.

У першому розділі роботи було проведено огляд існуючих методів та рішень для рішення даної проблеми. Було розглянуто різні методи: системи на основі правил, методи машинного навчання та гібридні, наведено їх переваги та недоліки у роботі з текстовими даними. У результаті аналізу було виявлевлено ключову проблему: відсутність моделі, яка могла б якісно та швидко тегувати текстові дані, особливо українською мовою, залишаючись оптимальним вибором незалежно від обсягу даних.

На основі виявлених недоліків у другому розділі було запропоноване рішення даної проблеми, що базується на гібридному підході, який скомбінує сильні сторони методів обраних для розробки. Такий підхід дозволить програмі адаптуватися до обсягу текстових даних та проводити якісний аналіз не поступаючись швидкістю існуючим методам. Також це дозволить підвищити точність роботи програми у порівнянні з існуючими методами у тегуванні складних даних, таких як технічна спеціалізована література, через можливість порівнювати результати аналізу різних методів. Обраними методами були: метод логістичної регресії, метод випадкового лісу, метод KeyBERT та BERT. Кожен з цих методів адаптує вхідні текстові дані для роботи з собою та проводить точний аналіз текстових даних. Для зручності аналізу та порівняння результатів було реалізовано усі методи окремо та окремо було реалізовано гібридний метод. Гнучка архітектура розроблюваної моделі дозволяє легко і ефективно вносити поправки у алгоритми навчання моделі, що дозволяє навчати її на текстових даних різних тем.

У третьому розділі було обгрунтоване використання мови програмування Python для написання методу та розглянуто різні бібліотеки з їх переваги та недоліками для використання у методі. Зокрема було обгрунтовано використання бібліотеки pyTorch для реалізації машинного навчання та різних бібліотек для виявлення ключових слів та семантичних ознак текстових даних. Було описано архітектуру розроблюваного методу та проведено більш глибокий опис кожного модулю.

У четвертому розділі були наведені критерії тестування розробленої системи, зокрема: precision (точність), recall (повнота) та F1-score (середнє гармонічне між точністю та повнотою). Було встановлено, що метод показує точні результати за оптимальну кількість часу у порівнянні з конкурентами. Було наведено час за який модель оброблює різну кількість документів. Також було проведено технічний аналіз та обгрунтовано ресурси який метод вимагає від системи з подальшим наведенням рекомендацій для використання. Таким чином, було підтверджено, що метод впроваджує новітні підходи до рішення проблеми автоматизованого тегування текстових даних та перевершує існуючі підходи до даної проблеми завдяки своїй структурованості, кращому врахування контексту і семантичних ознак, простоті використання та легкості навчання моделі.

Практичне використання моделі було протестовано у реальному середовищі де користувач може запровадити текст для автоматичного тегування і отримати структуровані результати. Це дозволяє інтегрувати систему у більшість існуючих платформ та підприємств. Велика гнучкість та точність моделі дозволяють використовувати її як для дослідження складної наукової літератури, так і для аналізу різного роду повсякденної інформації, наприклад новосних подій.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Наївний баєсів класифікатор [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Наївний_баєсів_класифікатор. – 01.04.201Vision Transformer (ViT) | MDN [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/transformers/model_doc/vit
2. TF-IDF (Term Frequency-Inverse Document Frequency) [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Tf–idf. – 15.05.2020.
3. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/1810.04805. – 11.10.2018.
4. Logistic Regression in Machine Learning [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. – 20.03.2023.
5. Random Forest Classifier [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/modules/ensemble.html#random-forests. – 12.01.2023.
6. KeyBERT: Minimal Keyword Extraction with BERT [Електронний ресурс]. – Режим доступу: https://github.com/MaartenGr/KeyBERT. – 05.02.2022.
7. spaCy: Industrial-Strength Natural Language Processing [Електронний ресурс]. – Режим доступу: https://spacy.io/. – 10.11.2023.
8. NLTK: Natural Language Toolkit [Електронний ресурс]. – Режим доступу: https://www.nltk.org/. – 03.09.2023
9. Multi-label Classification with scikit-learn [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/modules/multiclass.html. – 18.07.2023.
10. Sentence Transformers: Multilingual Sentence Embeddings [Електронний ресурс]. – Режим доступу: https://www.sbert.net/. – 22.06.2023.
11. PyTorch: Deep Learning Framework [Електронний ресурс]. – Режим доступу: https://pytorch.org/. – 14.12.2023.

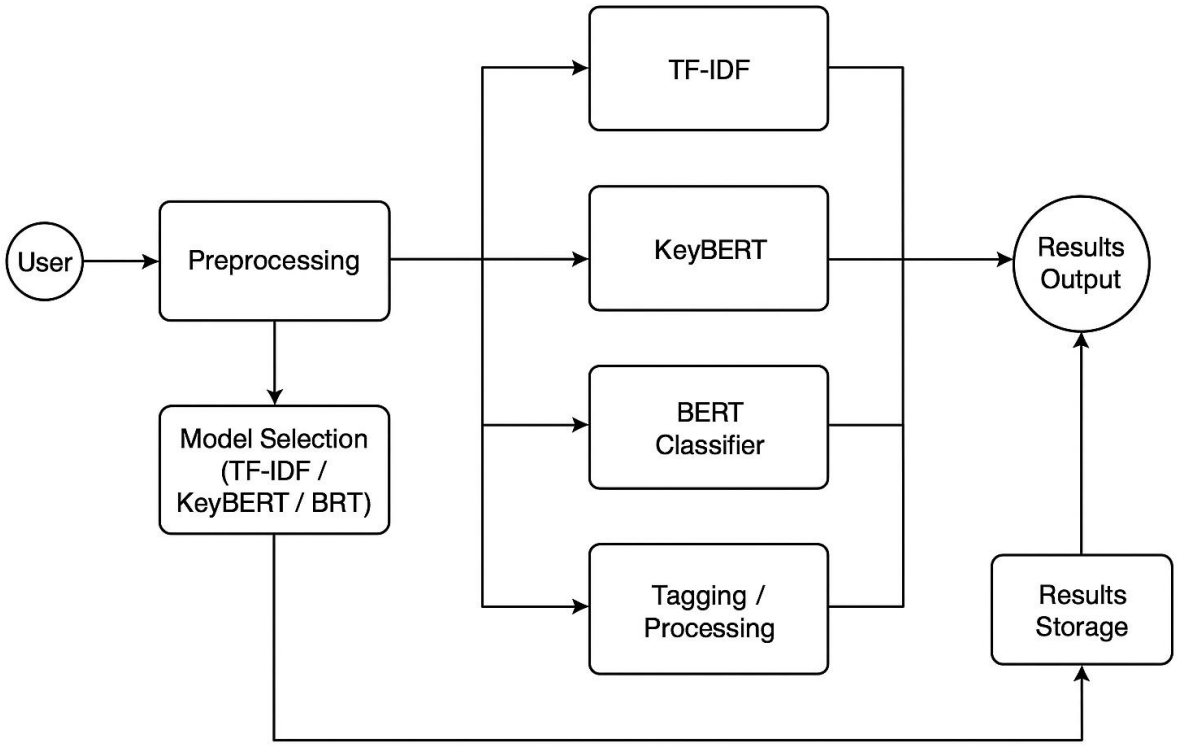
12. Hugging Face Transformers Library [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/transformers/index. – 25.01.2024.
13. Text Classification with BERT in PyTorch [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f. – 05.03.2023.
14. Stop Words in Natural Language Processing [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/. – 08.04.2022.
15. Ukrainian Language Processing with spaCy [Електронний ресурс]. – Режим доступу: https://spacy.io/usage/models#languages. – 17.10.2023.
16. Evaluation Metrics for Multi-label Classification [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/modules/model_evaluation.html#multilabel-ranking-metrics. – 30.05.2023.
17. Keyphrase Extraction: A Survey of the State of the Art [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/1905.05044. – 13.05.2019.
18. The Role of Lemmatization in NLP [Електронний ресурс]. – Режим доступу: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. – 09.11.2022.
19. One-vs-Rest Strategy for Multi-label Classification [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html. – 07.08.2023.
20. Optimizing Hyperparameters in Machine Learning Models [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624. – 21.04.2023.

ДОДАТКИ

Додаток 1

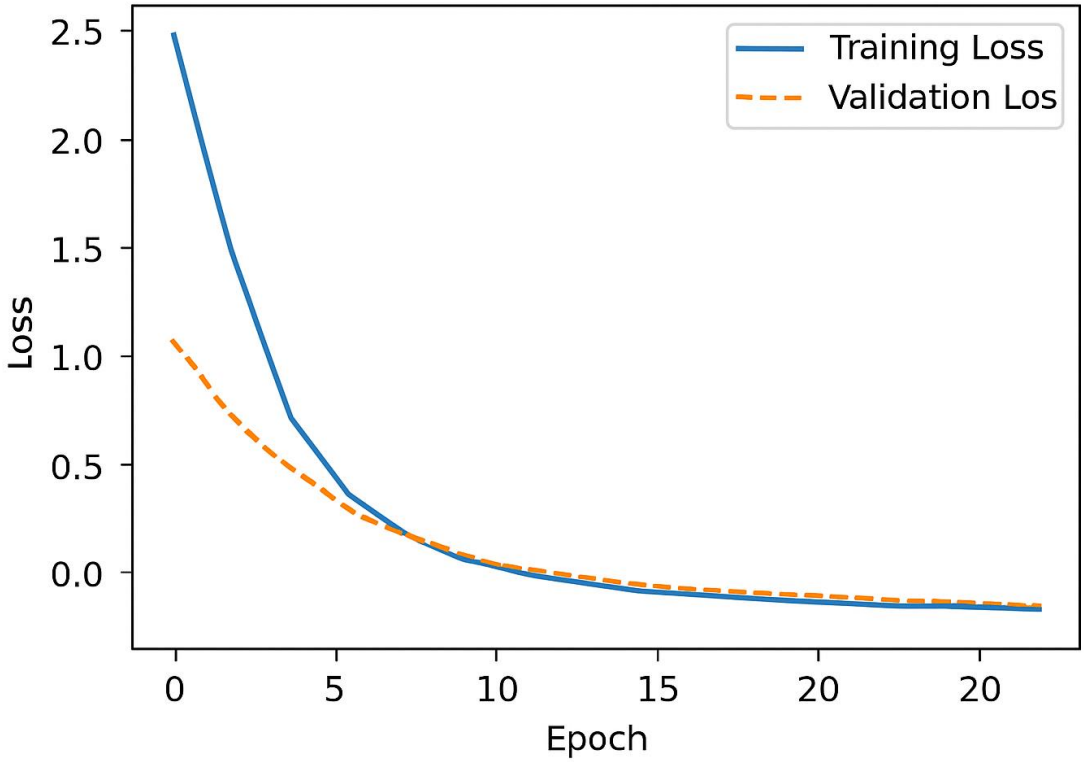
Копії графічних матеріалів

Загальний датафлоу програми



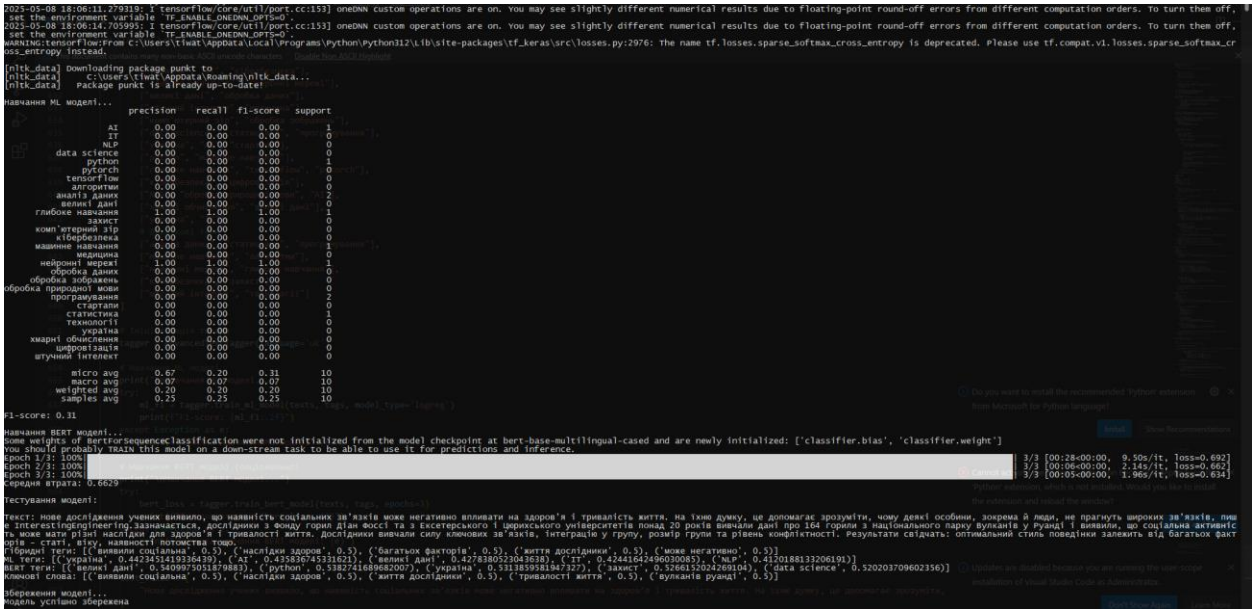
Бабак Артем, КП-31мн

Графік тренування моделі



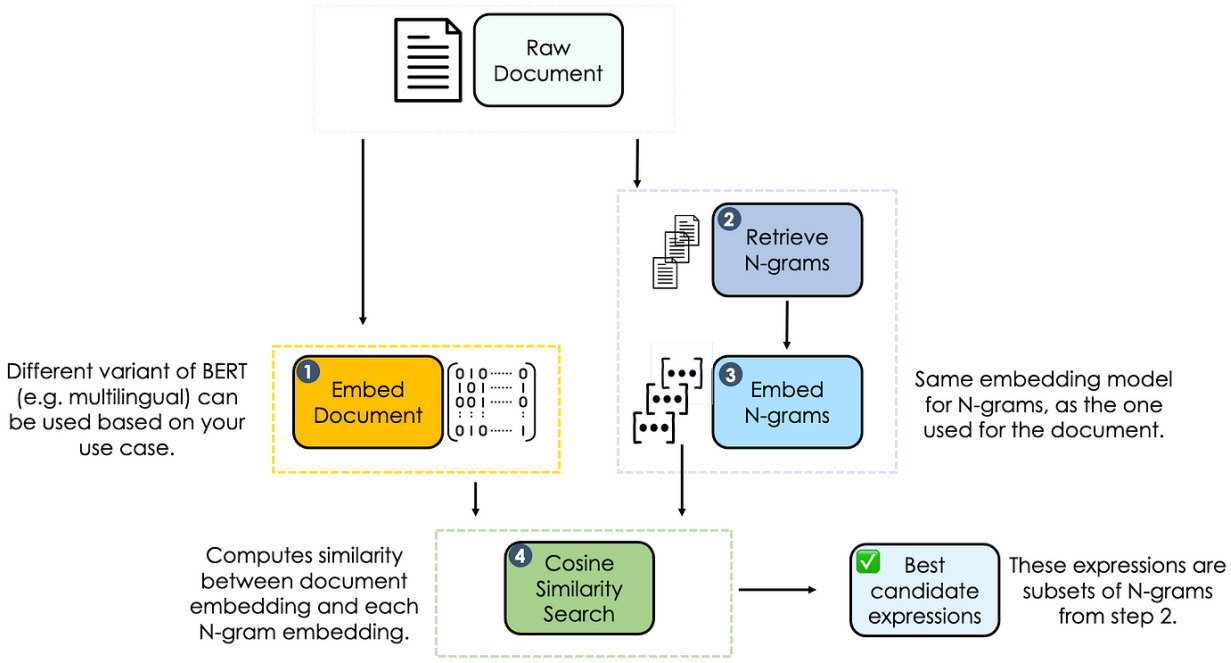
Бабак Артем, КП-31мн

Інтерфейс користувача



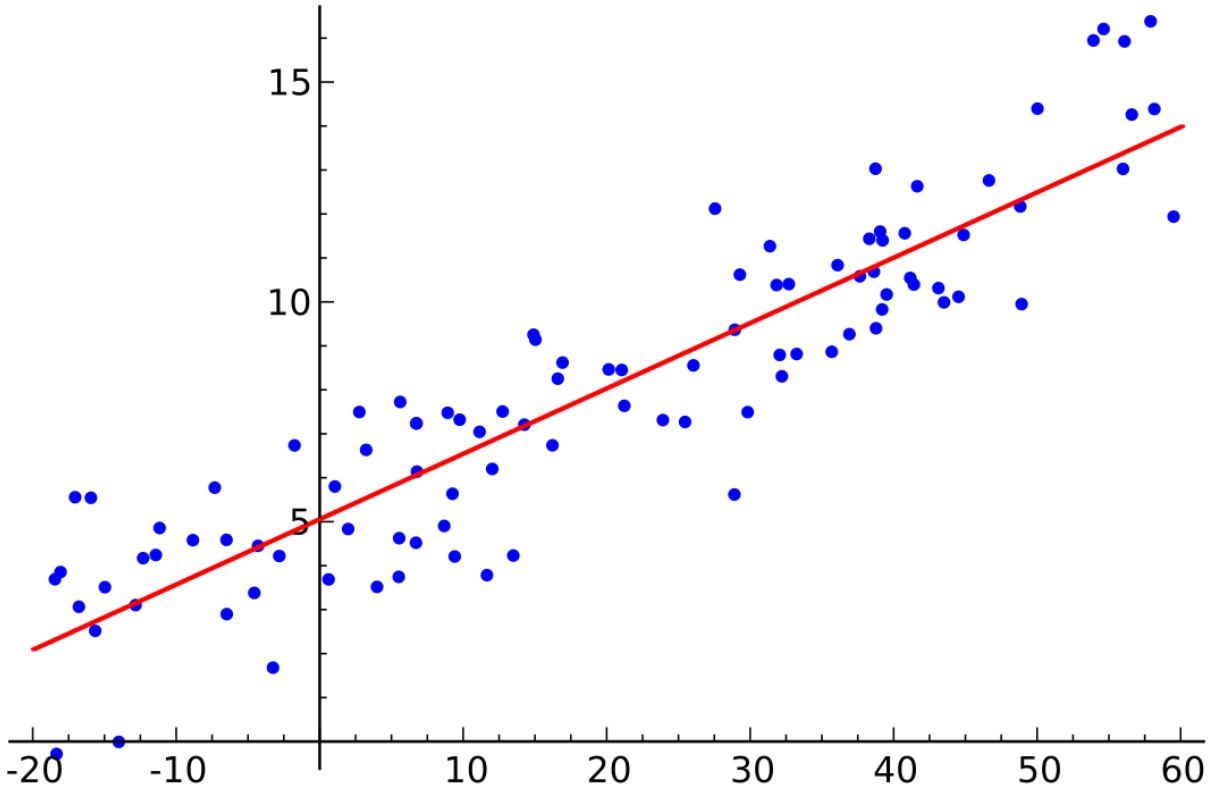
Бабак Артем, КП-31мн

Знаходження ключових слів



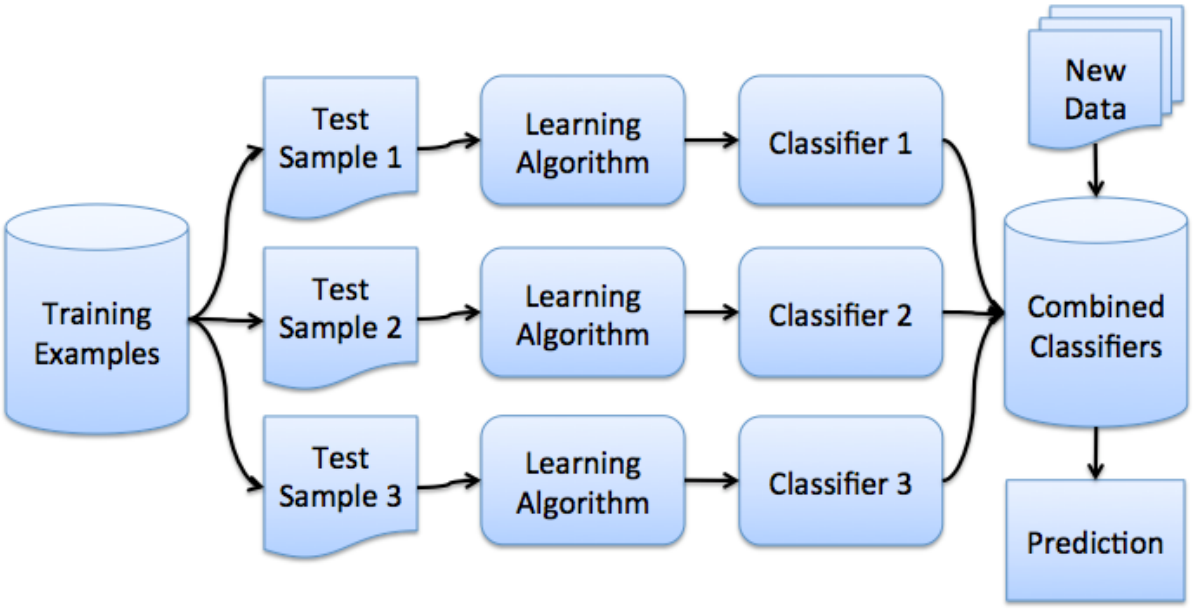
Бабак Артем, КП-31мн

Знаходження ознак методом логістичної регресії



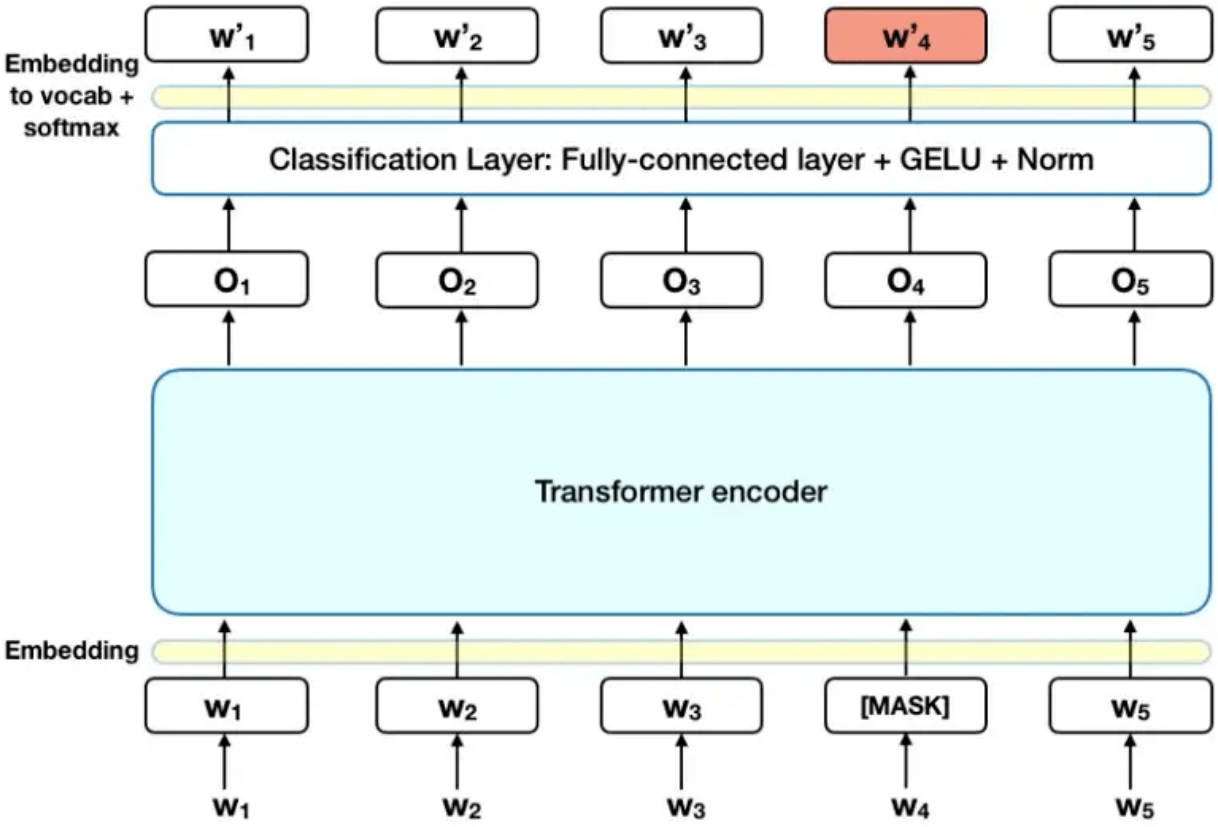
Бабак Артем, КП-31мн

Випадковий ліс



Бабак Артем, КП-31мн

Структура роботи BERT



Бабак Артем, КП-31мн

Додаток 2

Лістинг програми

Лістинг 1. Controller.py

```
import numpy as np
import os
import pickle
import warnings
from collections import Counter
from itertools import groupby
from typing import List, Tuple, Dict, Union, Optional

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
from sklearn.base import BaseEstimator

import nltk
from nltk.tokenize import word_tokenize
import spacy
from tqdm import tqdm

import torch
from torch.utils.data import Dataset, DataLoader
from transformers import (
BertTokenizer,
BertForSequenceClassification,
AdamW,
get_linear_schedule_with_warmup
)
from sentence_transformers import SentenceTransformer
from keybert import KeyBERT

# Налаштування
warnings.filterwarnings('ignore')
nltk.download('punkt')

class AdvancedTextTagger:
def __init__(self, language: str = 'uk', device: str = 'cpu'):
"""Ініціалізація тегера текстів

Args:
language: мова тексту ('uk' або 'en')
device: пристрій для обчислень ('cpu' або 'cuda')
"""
self.language = language
self.device = device
self.vectorizer: Optional[TfidfVectorizer] = None
self.label_binarizer: Optional[MultiLabelBinarizer] = None
self.classifier: Optional[BaseEstimator] = None
self.nlp: Optional[spacy.Language] = None
self.stop_words: set = set()
self.keyword_model: Optional[KeyBERT] = None
self.sentence_model: Optional[SentenceTransformer] = None
self.bert_model: Optional[BertForSequenceClassification] = None
self.bert_tokenizer: Optional[BertTokenizer] = None
self.tag_frequencies: Optional[Counter] = None

self._initialize_nlp_components()
self._initialize_keyword_model()
```

```
      def _initialize_nlp_components(self) -> None:
      """Ініціалізація NLP компонентів з автоматичним завантаженням
моделей"""
      try:
      if self.language == 'uk':
      try:
      self.nlp = spacy.load('uk_core_news_sm')
      except OSError:
      os.system("python -m spacy download uk_core_news_sm")
      self.nlp = spacy.load('uk_core_news_sm')

      # Оптимізований список українських стоп-слів
      self.stop_words = {
      'і', 'в', 'у', 'з', 'на', 'що', 'але', 'та', 'як', 'для', 'до', 'не',
'й', 'ж',
      'про', 'від', 'це', 'тим', 'за', 'а', 'бо', 'то', 'чи', 'вже', 'лише',
'ще'
      }
      else:
      self.nlp = spacy.load('en_core_web_sm')
      nltk.download('stopwords')
      from nltk.corpus import stopwords
      self.stop_words = set(stopwords.words('english'))
      except Exception as e:
      print(f"Помилка ініціалізації NLP: {e}")
      self.stop_words = set()

      def _initialize_keyword_model(self) -> None:
      """Ініціалізація моделі для виділення ключових слів"""
      try:
      self.sentence_model = SentenceTransformer(
      'paraphrase-multilingual-MiniLM-L12-v2',
      device=self.device
      )
      self.keyword_model = KeyBERT(model=self.sentence_model)
      except Exception as e:
      print(f"Помилка ініціалізації KeyBERT: {e}")
      self.keyword_model = None

      def preprocess_text(self, text: str, advanced: bool = True) -> str:
      """Розширена попередня обробка тексту

      Args:
      text: вхідний текст
      advanced: використовувати додаткові методи очистки

      Returns:
      Оброблений текст
      """
      if not isinstance(text, str) or not text.strip():
      return ""

      # Базова очистка
      text = text.lower().strip()

      # Видалення повторюваних символів
      if advanced:
      text = ''.join(''.join(s)[:2] for _, s in groupby(text))

      # Лематизація через spaCy
      try:
      doc = self.nlp(text)
```

```
words = [
token.lemma_ for token in doc
if not token.is_punct and not token.is_space
]

# Видалення стоп-слів та коротких слів
words = [
word for word in words
if word not in self.stop_words and len(word) > 2
]

return ' '.join(words)
except Exception:
return text.lower().strip()

def extract_keywords(
self,
text: str,
top_n: int = 5,
diversity: float = 0.5
) -> List[str]:
"""Виділення ключових слів з тексту

Args:
text: вхідний текст
top_n: кількість ключових слів
diversity: рівень різноманітності

Returns:
Список ключових слів
"""
if not self.keyword_model:
return []

try:
keywords = self.keyword_model.extract_keywords(
text,
keyphrase_ngram_range=(1, 2),
stop_words=list(self.stop_words),
top_n=top_n,
use_mmr=True,
diversity=diversity
)
return [kw[0] for kw in keywords if kw[1] > 0.2]
except Exception as e:
print(f"Помилка виділення ключових слів: {e}")
return []

def train_ml_model(
self,
texts: List[str],
tags: List[List[str]],
model_type: str = 'logreg',
test_size: float = 0.2,
random_state: int = 42
) -> float:
"""Навчання ML моделі з автоматичною оптимізацією

Args:
texts: список текстів
tags: список списків тегів
model_type: тип моделі ('logreg', 'naive_bayes', 'random_forest')
test_size: розмір тестового набору
```

```
        random_state: seed для відтворюваності

        Returns:
        F1-score моделі
        """
        # Перевірка вхідних даних
        if len(texts) != len(tags):
        raise ValueError("Кількість текстів і тегів має бути однаковою")

        if len(texts) < 5:
        raise ValueError("Необхідно мінімум 5 текстів для навчання")

        # Аналіз частот тегів
        self._analyze_tag_frequencies(tags)

        # Обробка текстів
        processed_texts = [self.preprocess_text(text) for text in texts]

        # Векторизація з адаптивними параметрами
        self.vectorizer = TfidfVectorizer(
        max_features=min(5000, len(texts)*10),
        ngram_range=(1, 2),
        stop_words=list(self.stop_words)
        )
        X = self.vectorizer.fit_transform(processed_texts)

        # Бінаризація міток
        self.label_binarizer = MultiLabelBinarizer()
        y = self.label_binarizer.fit_transform(tags)

        # Адаптивний підхід до розділення даних
        if len(texts) < 20:
        test_size = min(0.15, test_size)  # Зменшуємо тестовий набір для малих
даних
        print(f"Попередження: мало даних ({len(texts)}). Зменшено тестовий
набір до {test_size*100}%")

        # Використовуємо просте розділення для уникнення проблем з рідкісними
тегами
        X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=test_size,
        random_state=random_state,
        shuffle=True
        )

        # Вибір моделі з урахуванням розміру даних
        if model_type == 'logreg':
        model = OneVsRestClassifier(LogisticRegression(
        max_iter=1000,
        solver='liblinear',
        class_weight='balanced',
        random_state=random_state
        ))
        elif model_type == 'naive_bayes':
        model = OneVsRestClassifier(MultinomialNB())
        elif model_type == 'random_forest':
        model = OneVsRestClassifier(RandomForestClassifier(
        n_estimators=100,
        class_weight='balanced',
        random_state=random_state
        ))
        else:
```

```
        raise ValueError("Невідомий тип моделі. Оберіть: 'logreg',
'naive_bayes' або 'random_forest'")

        # Навчання моделі
        self.classifier = model
        try:
        self.classifier.fit(X_train, y_train)
        except Exception as e:
        raise RuntimeError(f"Помилка навчання моделі: {e}")

        # Оцінка моделі (якщо тестовий набір не порожній)
        if X_test.shape[0] > 0:
        y_pred = self.classifier.predict(X_test)
        print(classification_report(
        y_test, y_pred,
        target_names=self.label_binarizer.classes_,
        zero_division=0
        ))
        return f1_score(y_test, y_pred, average='micro')
        else:
        print("Тестовий набір порожній. Модель навчена, але оцінки немає.")
        return 0.0

        def _analyze_tag_frequencies(self, tags: List[List[str]]) -> None:
        """Аналіз частот тегів для подальшої оптимізації"""
        all_tags = [tag for sublist in tags for tag in sublist]
        self.tag_frequencies = Counter(all_tags)

        def train_bert_model(
        self,
        texts: List[str],
        tags: List[List[str]],
        epochs: int = 3,
        batch_size: int = 8,
        learning_rate: float = 2e-5
        ) -> float:
        """Навчання BERT моделі для багатолейбкової класифікації

        Args:
        texts: список текстів
        tags: список списків тегів
        epochs: кількість епох
        batch_size: розмір батча
        learning_rate: швидкість навчання

        Returns:
        Середня втрата під час навчання
        """
        if not torch.cuda.is_available() and self.device == 'cuda':
        print("Попередження: CUDA не доступна, використовується CPU")
        self.device = 'cpu'

        # Обробка міток
        self.label_binarizer = MultiLabelBinarizer()
        y = self.label_binarizer.fit_transform(tags)
        self._analyze_tag_frequencies(tags)

        # Ініціалізація BERT
        model_name = 'bert-base-multilingual-cased' if self.language == 'uk'
else 'bert-base-uncased'
        try:
        self.bert_tokenizer = BertTokenizer.from_pretrained(model_name)
        self.bert_model = BertForSequenceClassification.from_pretrained(
```

```
model_name,
num_labels=len(self.label_binarizer.classes_),
problem_type="multi_label_classification"
).to(self.device)
except Exception as e:
raise RuntimeError(f"Помилка завантаження BERT моделі: {e}")

# Підготовка даних
dataset = TextTaggingDataset(texts, y, self.bert_tokenizer)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# Налаштування оптимізатора
optimizer = AdamW(self.bert_model.parameters(), lr=learning_rate)
total_steps = len(dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(
optimizer,
num_warmup_steps=0,
num_training_steps=total_steps
)

# Процес навчання
self.bert_model.train()
losses = []

for epoch in range(epochs):
epoch_losses = []
progress_bar = tqdm(dataloader, desc=f"Epoch {epoch + 1}/{epochs}")

for batch in progress_bar:
optimizer.zero_grad()

inputs = {
'input_ids': batch['input_ids'].to(self.device),
'attention_mask': batch['attention_mask'].to(self.device),
'labels': batch['labels'].float().to(self.device)
}

try:
outputs = self.bert_model(**inputs)
loss = outputs.loss
loss.backward()

torch.nn.utils.clip_grad_norm_(self.bert_model.parameters(), 1.0)
optimizer.step()
scheduler.step()

epoch_losses.append(loss.item())
progress_bar.set_postfix({'loss': np.mean(epoch_losses[-10:])})
except Exception as e:
print(f"Помилка під час навчання: {e}")
continue

losses.extend(epoch_losses)

return np.mean(losses) if losses else 0.0

def predict_tags(
self,
text: str,
method: str = 'hybrid',
threshold: float = 0.3,
top_n: int = 5
) -> List[Tuple[str, float]]:
```

```
      """Передбачення тегів для тексту

      Args:
      text: вхідний текст
      method: метод передбачення ('hybrid', 'ml', 'bert', 'keywords')
      threshold: поріг для ймовірностей
      top_n: кількість тегів

      Returns:
      Список кортежів (тег, ймовірність)
      """
      if not text.strip():
      return []

      if method not in {'hybrid', 'ml', 'bert', 'keywords'}:
      raise ValueError("Невідомий метод. Оберіть: 'hybrid', 'ml', 'bert' або
'keywords'")

      if method == 'hybrid':
      # Комбінуємо всі доступні методи
      ml_tags = self._predict_ml_tags(text, threshold, top_n) if
self.classifier else []
      bert_tags = self._predict_bert_tags(text, threshold, top_n) if
self.bert_model else []
      keyword_tags = self.extract_keywords(text, top_n=top_n*2)  # Більше
ключових слів для гібриду

      # Об'єднання результатів з вагами
      all_tags = (
      [(tag, score * 0.7) for tag, score in ml_tags] +  # Вага для ML
      [(tag, score * 0.8) for tag, score in bert_tags] +  # Вага для BERT
      [(kw, 0.5) for kw in keyword_tags]  # Фіксована вага для ключових слів
      )

      # Агрегація результатів
      tag_scores = {}
      for tag, score in all_tags:
      if tag in tag_scores:
      tag_scores[tag] = max(tag_scores[tag], score)
      else:
      tag_scores[tag] = score

      # Сортування та обмеження кількості
      return sorted(tag_scores.items(), key=lambda x: x[1],
reverse=True)[:top_n]

      elif method == 'ml':
      return self._predict_ml_tags(text, threshold, top_n)
      elif method == 'bert':
      return self._predict_bert_tags(text, threshold, top_n)
      elif method == 'keywords':
      return [(kw, 0.5) for kw in self.extract_keywords(text, top_n=top_n)]

      def _predict_ml_tags(
      self,
      text: str,
      threshold: float,
      top_n: int
      ) -> List[Tuple[str, float]]:
      """Допоміжний метод для передбачення тегів ML моделлю"""
      if self.classifier is None or self.vectorizer is None or
self.label_binarizer is None:
      return []
```

```
      try:
      processed_text = self.preprocess_text(text)
      X = self.vectorizer.transform([processed_text])

      # Обробка різних типів моделей
      if hasattr(self.classifier, 'predict_proba'):
      probs = self.classifier.predict_proba(X)[0]
      elif hasattr(self.classifier, 'decision_function'):
      scores = self.classifier.decision_function(X)[0]
      probs = 1 / (1 + np.exp(-scores))  # Сигмоїда для перетворення у
ймовірності
      else:
      return []

      # Вибір тегів за порогом
      tags_with_scores = [
      (tag, float(probs[i]))
      for i, tag in enumerate(self.label_binarizer.classes_)
      if probs[i] >= threshold
      ]

      return sorted(tags_with_scores, key=lambda x: x[1],
reverse=True)[:top_n]
      except Exception as e:
      print(f"Помилка передбачення ML моделлю: {e}")
      return []

      def _predict_bert_tags(
      self,
      text: str,
      threshold: float,
      top_n: int
      ) -> List[Tuple[str, float]]:
      """Допоміжний метод для передбачення тегів BERT моделлю"""
      if self.bert_model is None or self.bert_tokenizer is None or
self.label_binarizer is None:
      return []

      try:
      encoding = self.bert_tokenizer(
      text,
      max_length=128,
      padding='max_length',
      truncation=True,
      return_tensors='pt'
      ).to(self.device)

      self.bert_model.eval()
      with torch.no_grad():
      outputs = self.bert_model(**encoding)
      probs = torch.sigmoid(outputs.logits).cpu().numpy()[0]

      tags_with_scores = [
      (tag, float(probs[i]))
      for i, tag in enumerate(self.label_binarizer.classes_)
      if probs[i] >= threshold
      ]

      return sorted(tags_with_scores, key=lambda x: x[1],
reverse=True)[:top_n]
      except Exception as e:
      print(f"Помилка передбачення BERT моделлю: {e}")
```

```
        return []

        def save_model(self, path: str = 'text_tagger_model') -> None:
        """Збереження моделі на диск

        Args:
        path: шлях до директорії для збереження
        """
        os.makedirs(path, exist_ok=True)

        if self.classifier is not None and self.vectorizer is not None and
self.label_binarizer is not None:
        try:
        with open(os.path.join(path, 'ml_model.pkl'), 'wb') as f:
        pickle.dump({
        'vectorizer': self.vectorizer,
        'label_binarizer': self.label_binarizer,
        'classifier': self.classifier,
        'tag_frequencies': self.tag_frequencies
        }, f)
        except Exception as e:
        print(f"Помилка збереження ML моделі: {e}")

        if self.bert_model is not None and self.bert_tokenizer is not None:
        try:
        self.bert_model.save_pretrained(os.path.join(path, 'bert_model'))
        self.bert_tokenizer.save_pretrained(os.path.join(path,
'bert_tokenizer'))
        except Exception as e:
        print(f"Помилка збереження BERT моделі: {e}")

        try:
        with open(os.path.join(path, 'config.pkl'), 'wb') as f:
        pickle.dump({
        'language': self.language,
        'device': self.device
        }, f)
        except Exception as e:
        print(f"Помилка збереження конфігурації: {e}")

        def load_model(self, path: str = 'text_tagger_model') -> None:
        """Завантаження моделі з диска

        Args:
        path: шлях до директорії з моделлю
        """
        try:
        if os.path.exists(os.path.join(path, 'ml_model.pkl')):
        with open(os.path.join(path, 'ml_model.pkl'), 'rb') as f:
        data = pickle.load(f)
        self.vectorizer = data['vectorizer']
        self.label_binarizer = data['label_binarizer']
        self.classifier = data['classifier']
        self.tag_frequencies = data.get('tag_frequencies', None)
        except Exception as e:
        print(f"Помилка завантаження ML моделі: {e}")

        try:
        if os.path.exists(os.path.join(path, 'bert_model')):
        self.bert_model = BertForSequenceClassification.from_pretrained(
        os.path.join(path, 'bert_model')
        )
        self.bert_tokenizer = BertTokenizer.from_pretrained(
```

```
os.path.join(path, 'bert_tokenizer')
)
except Exception as e:
print(f"Помилка завантаження BERT моделі: {e}")

try:
if os.path.exists(os.path.join(path, 'config.pkl')):
with open(os.path.join(path, 'config.pkl'), 'rb') as f:
config = pickle.load(f)
self.language = config['language']
self.device = config['device']
except Exception as e:
print(f"Помилка завантаження конфігурації: {e}")

self._initialize_nlp_components()
self._initialize_keyword_model()


class TextTaggingDataset(Dataset):
"""Датасет для BERT моделі"""
def __init__(self, texts: List[str], tags: np.ndarray, tokenizer:
BertTokenizer, max_length: int = 128):
self.texts = texts
self.tags = tags
self.tokenizer = tokenizer
self.max_length = max_length

def __len__(self) -> int:
return len(self.texts)

def __getitem__(self, idx: int) -> Dict[str, torch.Tensor]:
try:
encoding = self.tokenizer(
self.texts[idx],
max_length=self.max_length,
padding='max_length',
truncation=True,
return_tensors='pt'
)
return {
'input_ids': encoding['input_ids'].flatten(),
'attention_mask': encoding['attention_mask'].flatten(),
'labels': torch.FloatTensor(self.tags[idx])
}
except Exception as e:
print(f"Помилка обробки тексту {idx}: {e}")
# Повертаємо пустий тензор у разі помилки
return {
'input_ids': torch.zeros(self.max_length, dtype=torch.long),
'attention_mask': torch.zeros(self.max_length, dtype=torch.long),
'labels': torch.zeros(self.tags.shape[1], dtype=torch.float)
}


def main():
"""Приклад використання з повною обробкою помилок"""
try:
# Приклад даних - рекомендується використовувати більше 20 текстів
texts = [
"Машинне навчання - це підрозділ штучного інтелекту",
"Python є популярною мовою програмування для аналізу даних",
"Україна розвиває IT сектор, особливо кібербезпеку",
"Глибоке навчання використовує нейронні мережі",
```

```
        "Великі дані потребують спеціальних інструментів обробки",
        "Штучний інтелект знаходить застосування в медицині",
        "Комп'ютерний зір дозволяє машинам інтерпретувати зображення",
        "Data Science об'єднує статистику, програмування та предметну
експертизу",
        "Українські стартапи активно розвивають AI технології",
        "Python лідирує серед мов для машинного навчання",
        "TensorFlow та PyTorch - популярні бібліотеки для глибокого навчання",
        "Кібербезпека стає все важливішою в епоху цифровізації",
        "Обробка природної мови (NLP) - ключова технологія AI",
        "Хмарні обчислення полегшують роботу з великими даними",
        "Україна має потужний потенціал у сфері IT",
        # Додаткові приклади для покращення якості
        "Аналіз даних вимагає знання статистики та програмування",
        "Машинне навчання використовує алгоритми для прогнозування",
        "Глибокі нейронні мережі мають багато шарів",
        "Кібербезпека захищає системи від кібератак",
        "Штучний інтелект трансформує сучасні технології"
        ]

        tags = [
        ["машинне навчання", "AI"],
        ["python", "програмування", "аналіз даних"],
        ["україна", "IT", "кібербезпека"],
        ["глибоке навчання", "нейронні мережі"],
        ["великі дані", "обробка даних"],
        ["штучний інтелект", "медицина"],
        ["комп'ютерний зір", "обробка зображень"],
        ["data science", "статистика", "програмування"],
        ["україна", "AI", "стартапи"],
        ["python", "машинне навчання"],
        ["глибоке навчання", "tensorflow", "pytorch"],
        ["кібербезпека", "цифровізація"],
        ["NLP", "обробка природної мови", "AI"],
        ["хмарні обчислення", "великі дані"],
        ["україна", "IT"],
        # Додаткові теги
        ["аналіз даних", "статистика", "програмування"],
        ["машинне навчання", "алгоритми"],
        ["нейронні мережі", "глибоке навчання"],
        ["кібербезпека", "захист"],
        ["штучний інтелект", "технології"]
        ]

        # Ініціалізація тегера
        tagger = AdvancedTextTagger(language='uk')

        # Навчання ML моделі
        print("\nНавчання ML моделі...")
        try:
        ml_f1 = tagger.train_ml_model(texts, tags, model_type='logreg')
        print(f"F1-score: {ml_f1:.2f}")
        except Exception as e:
        print(f"Помилка навчання ML моделі: {e}")

        # Навчання BERT моделі (опціонально)
        print("\nНавчання BERT моделі...")
        try:
        bert_loss = tagger.train_bert_model(texts, tags, epochs=3)
        print(f"Середня втрата: {bert_loss:.4f}")
        except Exception as e:
        print(f"Помилка навчання BERT моделі: {e}")
```

```
    # Тестування
    test_texts = [
    "Нове дослідження учених виявило, що наявність соціальних зв’язків може негативно впливати на здоров’я і тривалість життя. На їхню думку, це допомагає зрозуміти, чому деякі особини, зокрема й люди, не прагнуть широких зв’язків, пише InterestingEngineering.Зазначається, дослідники з Фонду горил Діан Фоссі та з Ексетерського і Цюрихського університетів понад 20 років вивчали дані про 164 горили з Національного парку Вулканів у Руанді і виявили, що соціальна активність може мати різні наслідки для здоров’я і тривалості життя. Дослідники вивчали силу ключових зв’язків, інтеграцію у групу, розмір групи та рівень конфліктності. Результати свідчать: оптимальний стиль поведінки залежить від багатьох факторів - статі, віку, наявності потомства тощо."
    ]

    print("\nТестування моделі:")
    for text in test_texts:
    print(f"\nТекст: {text}")
    print("Гібридні теги:", tagger.predict_tags(text, method='hybrid'))
    if tagger.classifier:
    print("ML теги:", tagger.predict_tags(text, method='ml'))
    if tagger.bert_model:
    print("BERT теги:", tagger.predict_tags(text, method='bert'))
    print("Ключові слова:", tagger.predict_tags(text, method='keywords'))

    # Збереження моделі
    print("\nЗбереження моделі...")
    try:
    tagger.save_model()
    print("Модель успішно збережена")
    except Exception as e:
    print(f"Помилка збереження моделі: {e}")

    except Exception as e:
    print(f"Критична помилка: {e}")


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

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Метод та програмне забезпечення автоматизованого тегування текстових даних

Доповідач: Бабак Артем Андрійович

Науковий керівник: к.ф.-м.н., доц., доцент кафедри ПЗКС Нещадим О.М.

Київ – 2025

ПОСТАНОВКА ЗАДАЧІ

Об’єкт дослідження: процес автоматизованого тегування текстових документів.

Предмет дослідження: Методи та технології, що застосовуються в програмному забезпеченні для автоматизованого тегування текстових документів.

Мета дослідження: покращення процесу автоматичного присвоєння тегів до різних текстових даних.

ОКРЕМІ ЗАВДАННЯ

- Аналіз існуючих рішень для тегування текстових даних.
- Дослідження методів автоматичного тегування текстових даних.
- Реалізація власного методу тегування.
- Реалізація програмного забезпечення.
- Тестування.

АКТУАЛЬНІСТЬ ДОСЛІЖЕННЯ

- Зростання обсягу інформації, зокрема текстових даних.
- Має важливе значення у багатьох сферах наприклад: журналістика, наука, спорт.
- Велика кількість методів є застарілими та не ефективними. Більшість з них вимагають або багато часу, або втрачають у ефективності.

ТЕРМІНОЛОГІЯ

- KNN model with bagging (Named entity recognition) – метрична модель для автоматизованої класифікації об'єктів або регресії. У разі використання моделі для класифікації, об'єкт присвоюється до того класу, який є найбільш поширеним.
- Модель дерева рішень — модель, в якій кожне рішення базується на порівнянні двох чисел за постійний час. Її використовують для встановлення обчислювальної складності сортування та пошуку.

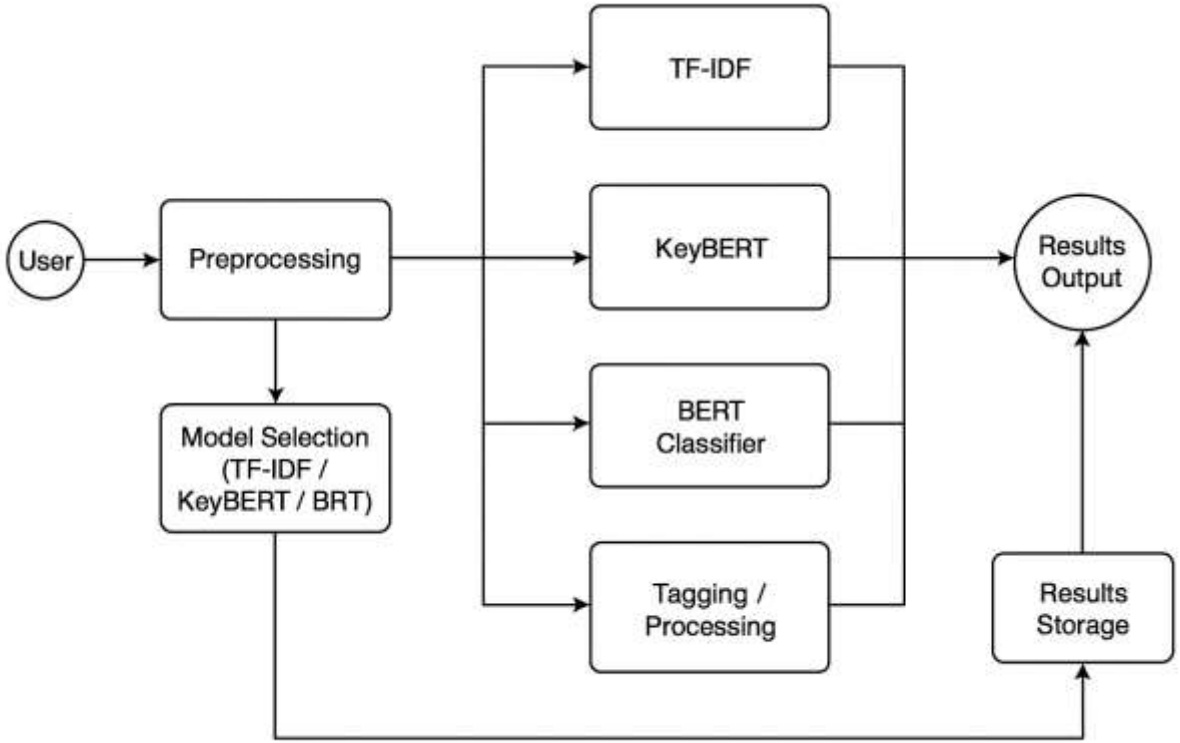
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ

	Фільтрація спаму	Класифікація за категоріями	Виконання в реальному часі
Метод Баєса	3	5	5
Метод опорних векторів	5	2	5
Метод k-найближчих сусідів	3	5	2
Дерева рішень	3	3	4
Випадковий ліс	3	4	4

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

1. Технології штучного інтелекту: методи тренування моделей, методи тегування текстових даних.
2. Технології тренування моделей штучного інтелекту (scikitLearn)
3. Мова програмування (Python, Javascript)

ЗАГАЛЬНИЙ ПОТІК ДАНИХ ПРОГРАМИ



ЗБІР І ПІДГОТОВКА ДАНИХ

- На цьому етапі формується корпус текстів, які будуть використані для навчання та тестування моделі. Джерела можуть бути різними — соціальні мережі, статті, форуми, відгуки користувачів. Важливо, щоб тексти мали достатню варіативність і охоплювали тематику, яка відповідає майбутнім тегам.

ПОПЕРЕДНЯ ОБРОБКА ТЕКСТУ

- Тексти очищуються від шумів: видаляються HTML-теги, спецсимволи, посилання, емодзі. Весь текст переводиться в нижній регістр, виконується токенізація, а для класичних моделей — також лематизація або стемінг. Якщо використовується BERT або подібні трансформери, обробка мінімальна, оскільки ці моделі враховують морфологію та контекст автоматично.

ВЕКТОРИЗАЦІЯ

- Тексти перетворюються у числові вектори, які модель зможе обробити. У класичних підходах це можуть бути TF-IDF або Word2Vec. У більш сучасних рішеннях використовуються контекстуальні трансформери, зокрема BERT, який дозволяє отримати представлення тексту з урахуванням значення кожного слова в контексті. У випадку KeyBERT беруться вектори з BERT і порівнюються з векторами окремих слів для виявлення ключових.

ПОБУДОВА МОДЕЛІ

- Створюється модель багатотегової класифікації. Для цього можна використати класичні алгоритми (наприклад, логістичну регресію з підходом "один проти всіх") або глибокі нейронні мережі. У випадку трансформерів використовується BERT, після якого додається класифікаційний шар із sigmoid-активацією. Кожен тег вважається незалежною бінарною класифікаційною задачею.

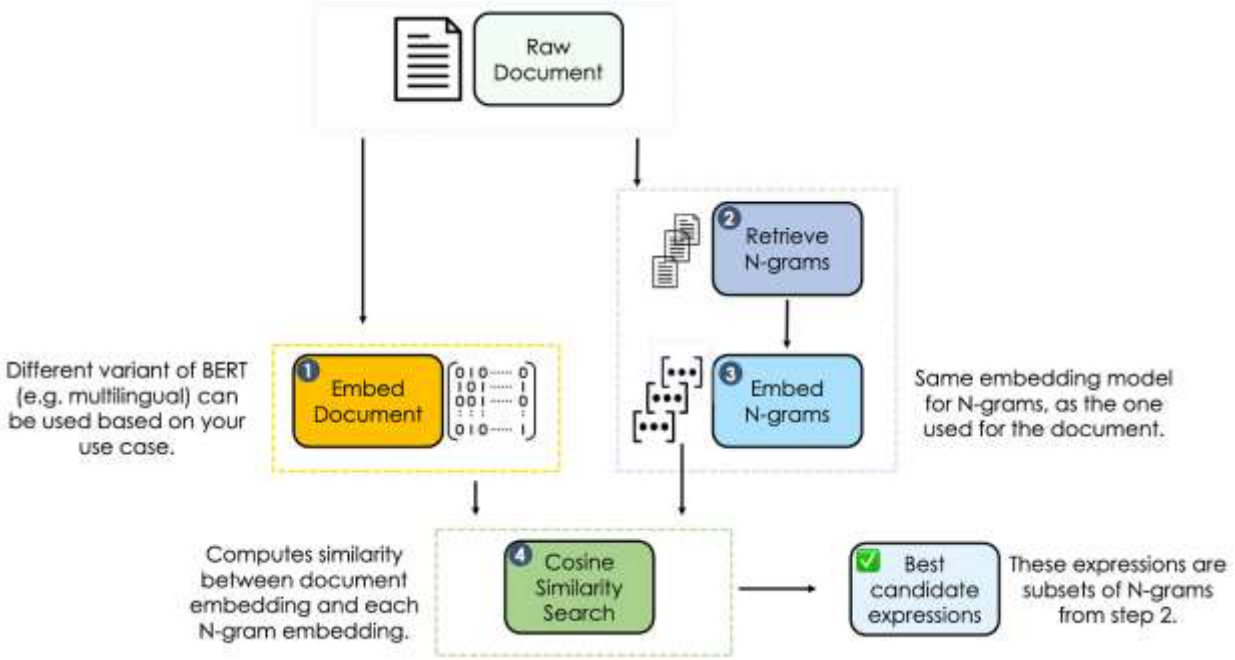
НАВЧАННЯ ТА ВАЛІДАЦІЯ

- Модель навчається на розмічених даних. При цьому частину корпусу залишають для валідації та тестування. Оцінка проводиться за метриками, що підходять для багатотегової класифікації. Також аналізуються випадки, коли модель пропускає важливі теги або навпаки — додає зайві.

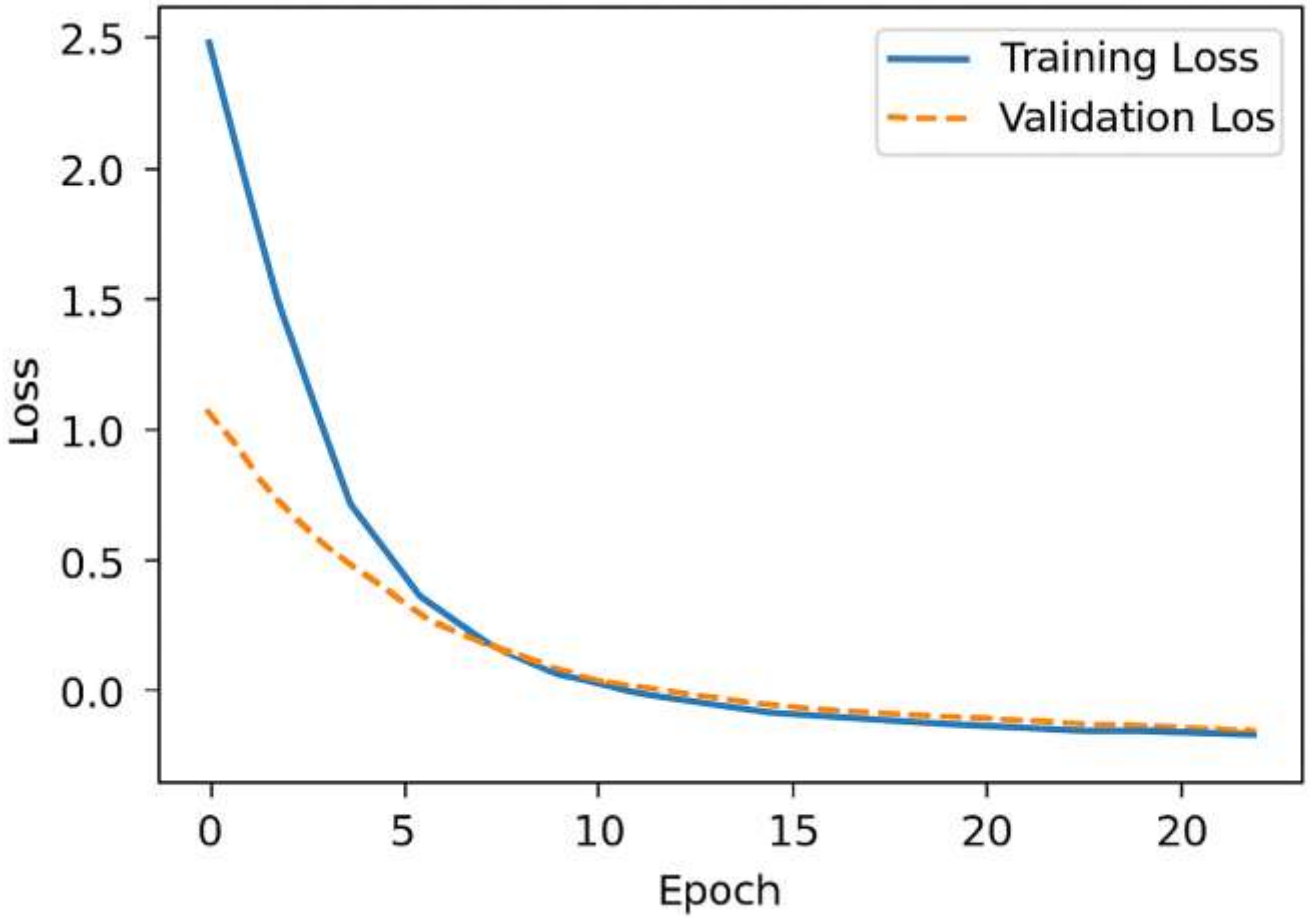
ТЕСТУВАННЯ

- Після навчання модель перевіряється на нових, раніше не бачених текстах. Це дозволяє оцінити її здатність до узагальнення. Також проводиться аналіз помилок і перевірка роботи з рідкісними тегами.

ЗНАХОДЖЕННЯ КЛЮЧОВИХ СЛІВ



ТРЕНУВАННЯ МОДЕЛІ



КРИТЕРІЙ ЕФЕКТИВНОСТІ

- В якості критерія ефективності в даній роботі використовуватимемо міру F1, яка у статистичному аналізі двійкової класифікації є мірою точності визначення категорій, які присутні в тексті. Також були окремо додані precision(точність) та recall(повнота).

ВЛАСНИЙ НАУКОВИЙ ДОРОБОК

1. Модифікація нейромережевого методу тегування текстових даних.
2. Розроблення методу комбінованого методу тегування текстових даних.
3. Проєктування архітектури ПЗ для забезпечення швидкої, точної та безперебійної роботи програми.

НАУКОВО-ІННОВАЦІЙНА НОВИЗНА

Розроблено спосіб автоматизованого тегування текстових даних, що відрізняється від існуючих комбінацією існуючих класичних моделей класифікації текстових даних, таких як BERT, RandomForest і логістична регресія. Запропоновано, ансамбль моделей для автоматизованої категоризації даних який відрізняється від відомих меншою обчислювальною складністю та точнішим результатом роботи з даними, які залежать від часу.

ПРАКТИЧНА ЦІННІСТЬ

Практична цінність отриманих в роботі результатів полягає в тому, що запропоновані методи та засоби дають змогу отримати точні результати під час пошуку та аналізу текстових описів. Розроблені методи і програмне забезпечення для автоматизованого тегування текстових даних, забезпечують вирішення задач пошуку у тексті, сприяють обґрунтованості знаходження результатів проаналізованих текстів.

ПОДАЛЬША РОБОТА

- Збільшення точності та різноманітності класифікації.
- Покращення програмного інтерфейсу для взаємодії з користувачем.

ВИСНОВКИ

1. Проведено аналіз існуючих методів та програмних рішень розпізнавання зображень для визначення геопозицій об’єктів
2. Розроблено алгоритмічно-програмний метод для автоматизованого тегування текстових даних, що продемонстрував точність тегування 89,7% у випадку с політичними новинами , а також приблизно – 82,6%, коли тема була інша з п’яти запропонованих.
3. Проведено аналіз отриманих результатів тегування запропонованим методом.
4. Розроблене програмне забезпечення, що реалізує метод розпізнавання.
5. Запропоновано шляхи вдосконалення розробленого методу.

Дякую за увагу!

Питання?