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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизація аналізу великих обсягів текстової інформації»

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

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

_____ Дмитро БЕНЕДЮК

(підпис)

Виконав: здобувач вищої освіти група ПДМ-62
Дмитро БЕНЕДЮК

Керівник:
к.т.н., доцент Ірина ЩЕРБИНА

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного

забезпечення Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри
Інженерії програмного забезпечення

_____Ірина ЗАМРІЙ
«_____» _____2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бенедюку Дмитру Володимировичу

1. Тема кваліфікаційної роботи: « Автоматизація аналізу великих обсягів текстової інформації»

керівник кваліфікаційної роботи Ірина ЩЕРБИНА, к.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи автоматизації аналізу великих обсягів текстової інформації, порівняльний аналіз методів аналізу великих обсягів текстової інформації для веб-застосунків на ASP.NET.

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

1. Дослідження методів автоматизації аналізу великих обсягів текстової інформації.

2. Моделювання та реалізація тестових сценаріїв, користуючись розробленим методом.
3. Аналіз результативності та продуктивності розробленого методу.

5. Перелік графічного матеріалу: *презентація*
   1. Мета, об'єкта та предмет дослідження.
   2. Актуальність роботи.
   3. Математична модель оцінки продуктивності.
   4. Блок схема роботи методу.
   5. Діаграма класів розробленої моделі.
   6. Результати тестування вибраних метрик.
   7. Результат роботи розробленого методу.
   8. Висновки.
   9. Публікації та апробація роботи.

6. Дата видачі завдання«16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури з аналізу великих обсягів текстової інформації	16.10-26.11.2024	
2	Вивчення матеріалів для аналізу методів обробки текстової інформації	27.10-04.11.2024	
3	Дослідження особливостей роботи ML бібліотек	05.11-14.11.2024	
4	Аналіз впливу методів на швидкість аналізу великих обсягів тексту	15.11-19.11.2024	
5	Моделювання та реалізація тестових сценаріїв для вимірювання продуктивності	20.11-25.12.2024	
6	Застосування та тестування методу аналізу великих обсягів текстової інформації	26.11-01.12.2024	
7	Оформлення роботи: вступ, висновки, реферат	02.12-04.12.2024	
8	Розробка демонстраційних матеріалів	05.12-08.12.2024	
9	Попередній захист роботи	09.12-16.12.2024	

Здобувач вищої освіти _____ Дмитро БЕНЕДЮК
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЩЕРБИНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 76 стор., 1 табл., 16 рис., 29 джерел.

Мета роботи: Оптимізація процесу обробки та аналізу великих обсягів текстових даних за рахунок використання методів машинного навчання, алгоритмів обробки природної мови, та методів векторизації TF-IDF.

Об'єкт дослідження: Процес обробки та аналізу текстових даних у системах автоматизації збору та обробки інформації.

Предмет дослідження: Засоби та технології для автоматизованого аналізу великих обсягів текстової інформації, зокрема методи обчислення TF-IDF, алгоритми машинного навчання та інструменти обробки природної мови.

Використано сучасні методи автоматизації аналізу великих обсягів текстової інформації, включаючи технології машинного навчання (ML) та математичне моделювання, а також інструменти ASP.NET для інтеграції та випробування алгоритмів обробки тексту у реальних умовах.

Проведено аналіз існуючих підходів до автоматизації текстової аналітики (Natural Language Processing, Sentiment Analysis, Text Classification), визначено основні показники ефективності та фактори, що впливають на якість аналізу текстових даних у web-застосунках.

Розроблено метод автоматизованого аналізу текстової інформації, який дозволяє формалізувати результати обробки та отримувати узагальнені метрики ефективності, зокрема точність класифікації, швидкість обробки, використання ресурсів і масштабованість системи.

Проведено експериментальні дослідження для валідації розробленого методу автоматизації аналізу тексту, зіставлено його результати з існуючими підходами та підтверджено практичну придатність запропонованого рішення.

Результати дослідження підтверджують доцільність та перспективність застосування запропонованого методу автоматизації текстової аналітики у web-застосунках на платформі ASP.NET з використанням мови програмування C# та бібліотеки ML. Розроблене рішення продемонструвало очікувані показники ефективності та може бути широко використане в галузях корпоративної інформаційної безпеки, аналітичних платформ та автоматизованих середовищ обробки тексту, сприяючи підвищенню продуктивності та якості аналізу великих обсягів текстової інформації.

У рамках дослідження також опрацьовано можливості адаптації методики до різних типів текстових даних та умов навантаження. Запропонований підхід дозволяє гнучко налаштовувати параметри тестових сценаріїв, що дає змогу враховувати специфіку конкретних проєктів, застосовувати метод у більш широкому діапазоні реальних умов і стимулює подальший розвиток автоматизованих інструментів для аналітики та оптимізації аналізу текстових даних.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ АНАЛІЗУ ТЕКСТУ, ОБРОБКА ВЕЛИКИХ ОБСЯГІВ ДАНИХ, МАШИННЕ НАВЧАННЯ (ML), ОБРОБКА ПРИРОДНОЇ МОВИ (NLP), ASP.NET, C#, СЕМАНТИЧНИЙ АНАЛІЗ, КЛАСИФІКАЦІЯ ТЕКСТУ, КЛАСТЕРИЗАЦІЯ ДАНИХ, ВИТЯГУВАННЯ КЛЮЧОВИХ СЛІВ, ТОЧНІСТЬ І ПОВНОТА АНАЛІЗУ, ШВИДКІСТЬ ОБРОБКИ ДАНИХ, ІНФОРМАЦІЙНИЙ ПОШУК, БІЗНЕС-АНАЛІТИКА, ОПТИМІЗАЦІЯ ПРОЦЕСІВ, ІНТЕЛЕКТУАЛЬНІ ВЕБ-ЗАСТОСУНКИ, МАСШТАБОВАНІСТЬ АЛГОРИТМІВ, АВТОМАТИЗОВАНІ СИСТЕМИ, АНАЛІЗ ТЕКСТОВИХ ДАНИХ, МЕТРИКИ ЕФЕКТИВНОСТІ.

ABSTRACT

The text part of the qualification work for obtaining the master's degree: 76 pages, 1 table, 16 figures, 29 sources.

The purpose of the Optimization of the process of processing and analyzing large volumes of textual data through the use of machine learning methods, natural language processing algorithms, and TF-IDF vectorization techniques.

The object of research the process of processing and analyzing textual data in automated information collection and processing systems.

The subject of research tools and technologies for automated analysis of large volumes of textual information, specifically TF-IDF calculation methods, machine learning algorithms, and natural language processing tools.

Modern methods of automating the analysis of large volumes of textual information have been applied, including machine learning (ML) and natural language processing (NLP) technologies, as well as ASP.NET tools for integrating and testing text analysis algorithms in real-world conditions.

An analysis of existing approaches to automated text analysis has been conducted, specifically methods of classification, clustering, keyword extraction, and building semantic models. Key performance indicators and factors affecting the quality and speed of text data analysis have been identified.

A method for automated text analysis has been developed, allowing for the formalization of data processing and the generation of generalized performance metrics, including accuracy, recall, processing speed, and resource utilization.

Experimental studies have been conducted to validate the proposed method, comparing its results with existing approaches and confirming the practical applicability of the solution for processing large volumes of textual data.

The results of the study demonstrate the feasibility of applying the developed automated text analysis method in ASP.NET-based systems using C# machine learning libraries. The proposed solution has shown high performance and can be widely used in business analytics, information retrieval, automated data processing systems, and intelligent web applications.

The study also explored the possibilities of adapting the methodology to different volumes of textual data and processing conditions. The developed approach allows for the customization of text analysis algorithm parameters according to the specifics of individual projects, ensuring high scalability and flexibility in application. This fosters further development of automated tools for analyzing large volumes of textual information and optimizing data processing in modern information systems.

KEYWORDS: TEXT ANALYSIS AUTOMATION, LARGE DATA PROCESSING, MACHINE LEARNING (ML), NATURAL LANGUAGE PROCESSING (NLP), ASP.NET, C#, SEMANTIC ANALYSIS, TEXT CLASSIFICATION, DATA CLUSTERING, KEYWORD EXTRACTION, ACCURACY AND RECALL, DATA PROCESSING SPEED, INFORMATION RETRIEVAL, BUSINESS ANALYTICS, PROCESS OPTIMIZATION, INTELLIGENT WEB APPLICATIONS, ALGORITHM SCALABILITY, AUTOMATED SYSTEMS, TEXT DATA ANALYSIS, PERFORMANCE METRICS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ML.NET — бібліотека для машинного навчання на платформі .NET.

ASP.NET Core — платформа для розробки веб-додатків на основі .NET.

API — інтерфейс програмування додатків, який забезпечує взаємодію між різними програмними системами.

UI — користувацький інтерфейс (від англ. User Interface), частина системи, через яку користувач взаємодіє з програмою.

DB — база даних, система для зберігання та організації даних.

HTTP — протокол передачі гіпертексту, що використовується для обміну інформацією між сервером і клієнтом.

JSON — формат для зберігання і передачі даних, що часто використовується для обміну інформацією між сервером і клієнтом.

ML — машинне навчання, галузь штучного інтелекту, що зосереджена на алгоритмах для автоматичного навчання на основі даних.

KNN — метод найближчих сусідів (K-Nearest Neighbors) — один з алгоритмів машинного навчання для класифікації та регресії.

SVM — метод опорних векторів (Support Vector Machine) — алгоритм машинного навчання для класифікації та регресії.

NN — нейронні мережі, метод машинного навчання, що моделює структуру людського мозку.

ROC — крива прийняття робіт (Receiver Operating Characteristic) — графік для оцінки якості моделей машинного навчання.

TP — true positive, вірно позитивний результат.

FP — false positive, хибно позитивний результат.

TN — true negative, вірно негативний результат.

FN — false negative, хибно негативний результат.

ВСТУП

У сучасному світі обсяг текстової інформації зростає з неймовірною швидкістю, охоплюючи всі сфери діяльності – від наукових досліджень і бізнес-аналітики до соціальних мереж і державного управління. Обробка та аналіз таких великих обсягів даних вручну стає не лише неефективним, але й практично неможливим завданням. У зв'язку з цим зростає попит на автоматизовані системи, здатні швидко та якісно обробляти текстову інформацію, витягувати з неї корисні знання та генерувати нові інсайти для ухвалення обґрунтованих рішень.

Впровадження методів машинного навчання (ML) та алгоритмів обробки природної мови (NLP) значно розширює можливості аналізу текстових даних, забезпечуючи автоматичне класифікування, виявлення ключових слів, тематичну сегментацію та прогнозування. Особливу роль у цьому процесі відіграють методи векторизації тексту, зокрема TF-IDF (Term Frequency – Inverse Document Frequency), які дозволяють оцінювати значущість термінів у великих текстових масивах.

Платформа ASP.NET та мова програмування C# є потужними інструментами для створення автоматизованих рішень з аналізу текстової інформації, що дозволяє інтегрувати моделі машинного навчання та алгоритми обробки тексту у реальні веб-застосунки та системи збору даних.

Актуальність даної теми зумовлена необхідністю підвищення продуктивності та точності аналізу великих обсягів текстових даних в умовах зростаючої інформаційної насиченості. Автоматизація цих процесів сприяє значному скороченню часу обробки даних, зменшенню людського фактору та підвищенню ефективності роботи систем аналітики та інформаційного пошуку.

Мета роботи: Оптимізація процесу обробки та аналізу великих обсягів текстових даних за рахунок використання методів машинного навчання, алгоритмів обробки природної мови, та методів векторизації TF-IDF.

Об'єкт дослідження: Процес обробки та аналізу текстових даних у системах автоматизації збору та обробки інформації.

Предмет дослідження: Засоби та технології для автоматизованого аналізу великих обсягів текстової інформації, зокрема методи обчислення TF-IDF, алгоритми машинного навчання та інструменти обробки природної мови.

Для досягнення поставленої мети було визначено наступні етапи:

1. Провести огляд існуючих методів обробки та аналізу текстових даних, зокрема методів векторизації, таких як TF-IDF, а також алгоритмів машинного навчання і обробки природної мови. Оцінити їх можливості в автоматизації процесів обробки великих обсягів текстової інформації, підвищенні точності аналізу та швидкості обробки даних.

2. Визначити основні вимоги до систем автоматизованого аналізу текстових даних, зокрема до ефективності векторизації текстів, точності класифікації та зниження шуму в даних. Провести аналіз методів, які дозволяють досягти високих результатів у обробці текстових даних, враховуючи частоту термінів та їх рідкість у колекції документів.

3. Змоделювати обробку текстових даних, та здійснити використання методу TF-IDF

4. Провести тестування створеної системи аналізу текстових даних на прикладі класифікації текстів і виділення ключових слів.

5. Проаналізувати переваги та недоліки методу, визначити його основні обмеження та можливості для подальшого вдосконалення в реальних системах обробки даних.

Таким чином, результати роботи спрямовані на розробку інструменту, який дозволить аналізувати великі обсяги текстової інформації.

1. АНАЛІЗ ТЕОРЕТИЧНИХ ОСНОВ ТА ТЕХНОЛОГІЙ

1.1 Основи роботи з ASP.NET для веб застосунків

ASP.NET – це сучасна, потужна платформа для розробки динамічних веб-застосунків, створена компанією Microsoft. Вона є частиною екосистеми .NET і забезпечує ефективні засоби для створення масштабованих, високопродуктивних і безпечних веб-рішень різної складності – від простих сайтів до складних корпоративних порталів та сервісів. ASP.NET дозволяє створювати інтерактивні веб-застосунки з багатим функціоналом, забезпечуючи при цьому простоту підтримки, розширюваність і гнучкість системи. Завдяки інтеграції з мовою програмування C#, платформою .NET та середовищем розробки Visual Studio, ASP.NET пропонує інструменти для швидкої та ефективної роботи над веб-застосунками будь-якого масштабу[1].

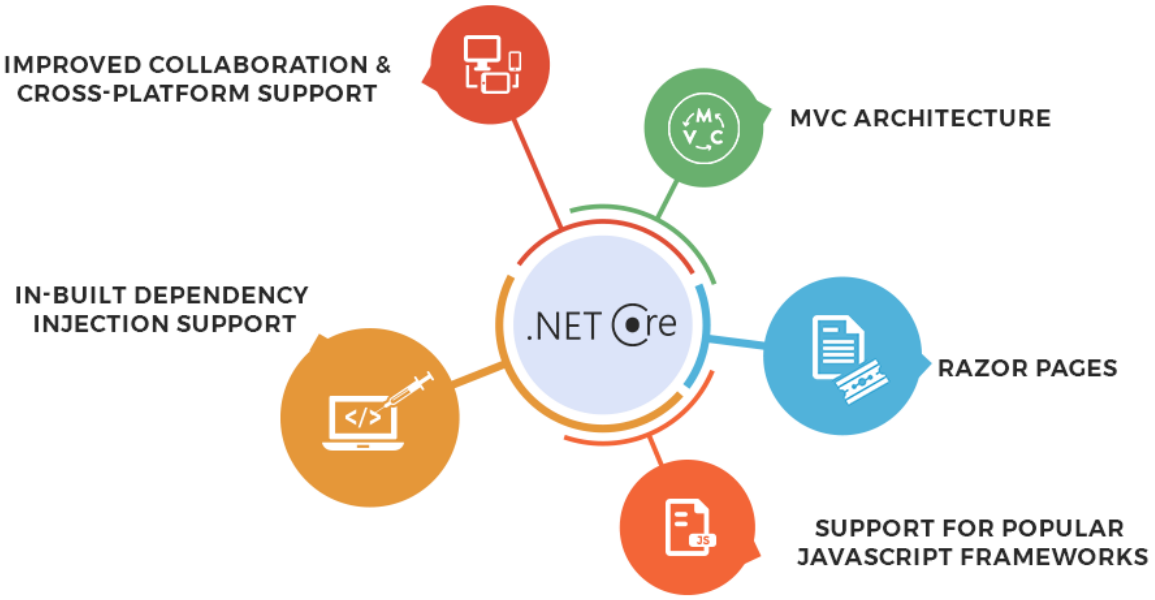


Рис 1.1 Переваги використання фреймворку ASP.NET Core

З основних особливостей .NET можна виділити наступні:

- Єдина екосистема: ASP.NET є невід'ємною частиною потужної та багатофункціональної екосистеми .NET, що дає змогу розробникам використовувати єдиний стек технологій для створення різноманітних типів застосунків. Це забезпечує високу узгодженість, інтеграцію та оптимізацію процесу розробки, оскільки всі компоненти екосистеми побудовані на єдиній платформі з загальними бібліотеками, інструментами та стандартами. В результаті, розробники можуть працювати в межах єдиної екосистеми, що істотно спрощує взаємодію між різними типами застосунків і сприяє швидшому освоєнню нових технологій. Усі компоненти .NET (ASP.NET, Xamarin, WPF, UWP, Azure) використовують спільні бібліотеки, засоби розробки та середовище виконання, зокрема Common Language Runtime (CLR), яке відповідає за виконання коду, збірку сміття та взаємодію з апаратним забезпеченням. Це означає, що розробники можуть використовувати однакові мовні конструкції та концепції, незалежно від того, чи працюють вони з веб-застосунками, десктопними програмами чи мобільними застосунками.

Завдяки спільним бібліотекам (наприклад, System.Linq, System.IO, System.Net) та компонентам, такі технології, як ASP.NET, можуть безперешкодно взаємодіяти з іншими частинами екосистеми, як, наприклад, з серверними або клієнтськими частинами застосунків, що значно знижує витрати на інтеграцію та обслуговування різних частин програми[2].

Через єдину екосистему також можна зробити висновок що ключовою перевагою у ASP.NET є кросплатформенність. Розробники можуть створювати веб-застосунки, які працюватимуть не тільки на Windows, але й на Linux та macOS. Це відкриває нові можливості для застосунків, які можуть працювати в різних середовищах без необхідності переписувати код для кожної операційної системи.

Використання спільного середовища .NET Core дозволяє запускати застосунки на різних платформах, зберігаючи високу продуктивність та зручність розробки, що, у свою чергу, забезпечує максимальну ефективність ресурсів та

швидкість виконання. Це також дає змогу розробникам реалізувати мультиплатформенні застосунки, що є важливим для багатьох сучасних проектів.

.NET екосистема підтримує створення не тільки веб-застосунків (через ASP.NET), але й мобільних застосунків (через Xamarin), десктопних програм (через WPF, Windows Forms), а також сервісів, орієнтованих на хмарні технології (через Azure). Це дозволяє розробникам використовувати одну мову програмування – C# – для розробки на будь-якій платформі та для різних типів застосунків.

Оскільки .NET платформа є стандартизованою, розробники мають доступ до чітких та загальноприйнятих стандартів розробки, що включають архітектурні патерни, такі як Model-View-Controller (MVC), Dependency Injection (DI), Unit of Work та інші. Ці патерни та практики дозволяють будувати масштабовані та стійкі до змін веб-застосунки, що є важливим при роботі з великими обсягами даних або створенні складних бізнес-логік.

Використання стандартів та загальних практик не тільки полегшує процес розробки, але й забезпечує зручність у тестуванні, налагодженні та підтримці готових рішень.

Один з головних принципів .NET – спрощення розробки. Завдяки використанню єдиного середовища розробки – Visual Studio – програмісти можуть зручно працювати з різними типами проектів у межах однієї IDE. Інтеграція з такими інструментами, як NuGet, Azure DevOps, GitHub, дає змогу швидко налаштовувати середовище для створення, тестування та розгортання застосунків.

Крім того, завдяки наявності великої кількості готових бібліотек та компонентів, розробники можуть використовувати попередньо створені рішення для інтеграції з іншими системами (наприклад, зовнішні бази даних, сторонні API тощо), що значно скорочує час розробки та знижує вартість проектів.

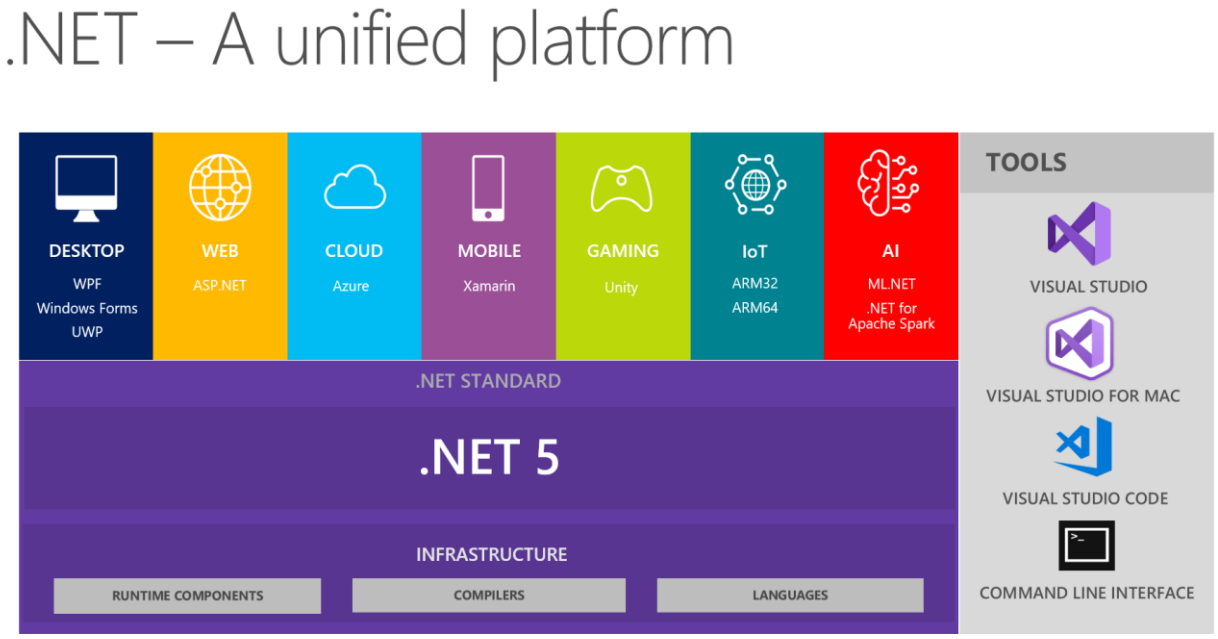


Рис. 1.2 Екосистема .NET

- Висока продуктивність: ASP.NET має кілька важливих особливостей, що дозволяють забезпечити високу продуктивність навіть у випадку значного навантаження. Окрім компіляції коду, яка значно пришвидшує виконання, платформа активно використовує механізми кешування на різних рівнях. Це включає кешування сторінок, кешування результатів запитів до бази даних і збереження даних у пам'яті для зменшення часу доступу. Кешування дозволяє мінімізувати необхідність повторного виконання однакових операцій, що суттєво знижує навантаження на сервер і зменшує витрати часу на обробку запитів. Завдяки цьому, навіть при значних обсягах трафіку веб-застосунки можуть забезпечити високу швидкість реагування і стабільну роботу без зниження продуктивності. Ще одним важливим фактором є здатність ASP.NET оптимізувати використання системних ресурсів, таких як пам'ять і процесор, завдяки ефективному управлінню ресурсами під час виконання програм. Вбудовані механізми автоматичного управління пам'яттю, зокрема збірка сміття, дозволяють платформі ефективно управляти обсягами даних, що зберігаються в пам'яті, і зменшувати ймовірність виникнення проблем, пов'язаних з надмірним споживанням пам'яті. Крім того, можливість налаштування параметрів, таких як кількість потоків і рівень

паралелізму, дає змогу гнучко адаптувати продуктивність системи до різних умов навантаження.

ASP.NET також підтримує асинхронне виконання запитів, що дозволяє значно підвищити ефективність обробки запитів, особливо в умовах великого навантаження або при обробці складних обчислень. Асинхронність дає змогу не блокувати основний потік виконання програми, що дозволяє зберігати високу пропускну здатність і швидкість відповіді навіть при обробці складних обчислень. Асинхронність дає змогу не блокувати основний потік виконання програми, що дозволяє зберігати високу пропускну здатність і швидкість відповіді навіть при значному навантаженні на сервер.

- Маштабованість: пропонує відмінні можливості для масштабування веб-застосунків, що дозволяє розробникам адаптувати їх під зростаючі вимоги користувачів і збільшення обсягів даних. Масштабованість є однією з ключових переваг платформи, адже вона підтримує як вертикальне, так і горизонтальне масштабування. Вертикальне масштабування передбачає підвищення потужності існуючого сервера, наприклад, за рахунок додавання ресурсів, таких як пам'ять або процесорна потужність. Горизонтальне масштабування, в свою чергу, дозволяє додавати нові сервери для розподілу навантаження та обробки більшої кількості запитів. Платформа забезпечує підтримку горизонтального масштабування через інтеграцію з хмарними сервісами, такими як Microsoft Azure, що дозволяє без проблем розширювати інфраструктуру залежно від вимог проекту. Це робить ASP.NET ідеальним вибором для веб-застосунків, які потребують адаптації до змінних умов, наприклад, у випадку зростання трафіку або необхідності обробки великих обсягів даних.
- Безпека є важливою складовою веб-застосунків, і ASP.NET забезпечує вбудовані засоби для захисту застосунків від різноманітних загроз. Платформа надає механізми аутентифікації та авторизації, які допомагають забезпечити доступ лише авторизованим користувачам. Крім того, ASP.NET містить вбудовані інструменти для захисту від таких атак, як міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF) та SQL-ін'єкції, що є одними з найпоширеніших типів

атак на веб-застосунки. Ці механізми допомагають значно знизити ймовірність виникнення уразливостей і забезпечують належний рівень безпеки застосунків на платформі ASP.NET. Вбудовані засоби захисту дають змогу розробникам зосередитися на бізнес-логіці застосунку, не турбуючись про наявність основних заходів безпеки.

- Гнучка архітектура : ASP.NET також пропонує гнучку архітектуру, яка підтримує різні підходи до розробки застосунків, дозволяючи обирати найбільш підходящий стиль архітектури залежно від вимог проекту. Розробники можуть обирати між монолітною архітектурою, мікросервісами, serverless-підходами або іншими архітектурними стилями, що дозволяє створювати масштабовані, зручні для обслуговування та ефективні застосунки. Це дозволяє адаптувати проект до специфічних потреб, зберігаючи при цьому високу продуктивність, зручність розширення та спрощене управління кодовою базою. Гнучкість архітектурних рішень дозволяє розробникам вибирати ті стратегії, які найкраще відповідають вимогам їхнього бізнесу і технологічного оточення[3].

1.2 Робота з текстовими даними в C#

У мові програмування C# робота з текстовими даними є одним із важливих аспектів, що охоплює широкий спектр операцій, таких як маніпулювання рядками, обробка текстових файлів, використання регулярних виразів, а також застосування механізмів для роботи з кодуваннями. Ці можливості забезпечують високу ефективність при обробці великих обсягів текстової інформації, що є особливо важливим у контексті сучасних веб-застосунків і систем автоматизації.

Одним із основних типів даних для зберігання тексту в C# є клас String, який являє собою незмінний об'єкт. Це означає, що будь-яка операція зі зміни рядка, така як додавання, видалення або заміна символів, призводить до створення нового екземпляра рядка. Така особливість реалізації класу String забезпечує безпеку при використанні в багатозадачних середовищах, однак може бути менш ефективною при великих обсягах даних або при частих операціях зі зміною

тексту. У таких випадках доцільно використовувати клас StringBuilder, який дозволяє здійснювати зміни в тексті без створення нових об'єктів, що значно підвищує ефективність роботи з великими текстовими даними, зокрема при численних операціях конкатенації або вставки.

Для виконання складніших маніпуляцій з текстовою інформацією C# пропонує потужний інструмент у вигляді регулярних виразів, які реалізовані через клас Regex, що знаходиться в просторі імен System.Text.RegularExpressions. Регулярні вирази дають змогу здійснювати складний пошук та заміну в тексті за заданими патернами, що дозволяє ефективно обробляти текстові дані, а також виконувати валідацію введених даних, наприклад, перевірку електронних адрес або телефонних номерів. Вони також є незамінними при необхідності витягування певних даних з тексту, що особливо актуально при роботі з великими наборами інформації.

C# також надає ефективні засоби для роботи з текстовими файлами, зокрема через класи StreamReader та StreamWriter. Клас StreamReader дозволяє здійснювати послідовне читання текстових файлів, що є важливим для роботи з великими даними, оскільки він підтримує різноманітні формати кодування. Це дозволяє правильно обробляти текст, що зберігається в різних кодуваннях, таких як UTF-8, UTF-16, ASCII, або Unicode. Клас StreamWriter, у свою чергу, використовується для запису текстової інформації в файли з підтримкою різних кодувань, що забезпечує коректне збереження даних і зручність їх подальшого використання.

Одним із ключових аспектів при роботі з текстовими даними є підтримка різних типів кодувань. C# дозволяє налаштовувати кодування при зчитуванні та запису тексту, що є необхідним при роботі з мультимовними системами або при обробці текстів, що містять спеціальні символи. Це дає змогу забезпечити коректну обробку даних незалежно від їх вихідного кодування, що має велике значення для багатомовних веб-застосунків або систем, що працюють з міжнародними даними.

Додатково, C# надає можливість використання LINQ (Language Integrated Query) для обробки колекцій текстових даних. LINQ дозволяє застосовувати декларативний підхід до обробки масивів або списків, що значно спрощує код і покращує його читабельність. LINQ дає змогу ефективно фільтрувати, сортувати, групувати та здійснювати інші операції на колекціях, що містять текстові дані. Це особливо корисно при необхідності швидкого пошуку в колекціях або при виконанні складних перетворень тексту в реальному часі[4].

В умовах обробки великих обсягів текстових даних або в контексті створення розвинутих систем обробки природної мови (NLP), C# пропонує інтеграцію з різноманітними бібліотеками та інструментами для аналізу тексту. Це дозволяє автоматизувати процеси обробки, наприклад, класифікації тексту, аналізу настроїв, витягування сутностей або побудови семантичних моделей. Завдяки цьому C# стає потужним інструментом для побудови систем, здатних ефективно працювати з великими масивами текстової інформації та виконувати складні завдання, пов’язані з її аналізом.

1.3 Використання бібліотек ML.NET для аналізу тексту

Бібліотека ML.NET є потужним інструментом для розробки та реалізації моделей машинного навчання в середовищі C#. Вона надає великий набір можливостей для роботи з даними, включаючи обробку текстової інформації, що є важливим компонентом при побудові інтелектуальних систем. Використання ML.NET для аналізу тексту дозволяє ефективно вирішувати завдання класифікації текстів, кластеризації, видобутку сутностей, а також побудови рекомендаційних систем або аналізу настроїв.

Одним з основних інструментів для роботи з текстом в ML.NET є модель "TextFeaturizingEstimator", яка виконує перетворення текстових даних у векторні ознаки. Цей процес є необхідним для того, щоб текстові дані можна було використовувати в моделях машинного навчання. Одним із найбільш популярних методів перетворення є векторизація тексту за допомогою TF-IDF (Term

Frequency-Inverse Document Frequency), що дозволяє оцінити важливість кожного слова в документі на основі його частоти появи в тексті та рідкості серед всіх документів у наборі даних[5].

ML.NET надає вбудовані механізми для виконання попередньої обробки тексту, включаючи видалення стоп-слів, нормалізацію, лематизацію і стемінг. Це дозволяє значно покращити якість навчання моделей машинного навчання та зменшити кількість непотрібної інформації, що може негативно вплинути на ефективність алгоритмів.

Для задач класифікації тексту, ML.NET пропонує кілька алгоритмів машинного навчання, кожен з яких має свої переваги та застосування в залежності від типу текстових даних та конкретної задачі:

- **Логістична регресія:** Логістична регресія є одним з найпопулярніших алгоритмів для задачі класифікації, зокрема для двокласової класифікації. Вона використовує функцію активації (сигмоїдну функцію), щоб перевести лінійні комбінації ознак у ймовірність того, що об'єкт належить до певного класу. У контексті аналізу тексту, логістична регресія зазвичай застосовується для задач, де необхідно віднести текст до двох категорій, таких як позитивний або негативний відгук, спам або не спам. Оскільки алгоритм є лінійним, він добре працює на текстах, де зв'язок між ознаками (наприклад, словами) та класом є лінійним або може бути наближений лінійною моделлю[6].
- **Наївний баєсів класифікатор:** Наївний баєсів класифікатор ґрунтується на байєсівській теоремі й припускає незалежність між ознаками (словами). Цей алгоритм є дуже ефективним для великих наборів текстових даних і часто використовується для задач класифікації тексту, таких як фільтрація спаму, автоматична категоризація новин, класифікація електронних листів тощо. Оскільки кожне слово у вхідному документі розглядається як окреме, незалежне ознака, цей метод є досить простим і швидким, але може не працювати так ефективно, якщо залежності між словами в документі суттєві. Тим не менше, наївний баєсів класифікатор може показувати дуже хороші результати, особливо при великих обсягах текстових даних.

- **Дерева рішень:** Дерева рішень – це ще один популярний алгоритм, який використовується для класифікації тексту. Він працює шляхом побудови дерева, де кожна внутрішня вершина відповідає певному порогу для ознаки (наприклад, частота появи слова), а кінцева вершина (листи дерева) представляють класи. Алгоритм розбиває простір ознак на підмножини, які вказують на ймовірний клас для кожного тексту. Дерева рішень є інтуїтивно зрозумілими і дозволяють легко інтерпретувати, чому було прийнято певне рішення для класифікації. Проте вони можуть бути схильні до перенавчання, особливо при глибоких деревах. Для запобігання перенавчанню часто використовуються методи, такі як випадкові ліси (Random Forests), де створюється набір дерев, що поліпшує точність моделі.
- **Алгоритм опорних векторів (SVM):** Алгоритм опорних векторів є одним з потужних методів для задач класифікації тексту, особливо коли потрібно розділити дані, які не є лінійно відокремленими. SVM знаходить гіперплощину, яка максимізує відстань між класами в просторі ознак. Для текстових даних SVM часто застосовується для складних задач класифікації, де текст може містити множину складних взаємозв'язків. Оскільки SVM є потужним інструментом для виявлення структур у даних, він може працювати ефективно навіть в умовах високої варіативності в контексті текстових документів.

Алгоритми кластеризації тексту також є важливими інструментами для автоматизації організації великих обсягів даних, де не завжди є чітке розмежування між класами, і метою є виявлення груп або структур в даних без попереднього маркування класів[7].

- Алгоритм k-середніх (k-means): Алгоритм k-середніх є одним з найбільш популярних методів кластеризації, який розбиває набір текстових документів на k груп, де кожна група є кластером документів, схожих за змістом. Алгоритм працює за принципом мінімізації відстані між центроїдами кластерів і всіма точками в групі. Для аналізу текстів цей метод вимагає попереднього перетворення текстових даних у векторні ознаки, наприклад, за допомогою методів векторизації, таких як TF-IDF. Оскільки алгоритм вимагає вказівки кількості кластерів заздалегідь, важливо правильно вибрати значення k, щоб

отримати найбільш значущі групи (1.1).

$$J(c,\mu) = i = 1\sum nk = 1\sum K1(ci = k) \parallel xi - \mu k \parallel^2 \quad (1.1)$$

де J(c,μ) — функція варіації кластерів, яка відображає якість кластеризації; чим менше значення, тим краще обрані центроїди μk,

сі — кластер, до якого належить точка хі,

μk — центр (центроїд) кластера k,

хі — точка даних із вхідного набору,

1(сі=k) — індикаторна функція, яка дорівнює 1, якщо точка хіх_іхі належить кластеру kkk, і 0 в іншому випадку,

‖хі−μk‖ — квадрат Евклідової відстані між точкою хі та центроїдом μk.

- Ієрархічна кластеризація: Ієрархічна кластеризація є ще одним методом для групування текстів на основі схожості. Цей метод будує ієрархічну структуру, де на кожному етапі зливаються або діляться класи, утворюючи деревоподібну структуру (дендрограму). Ієрархічна кластеризація не вимагає попереднього визначення кількості кластерів, оскільки вона будується поступово, що дає можливість досліднику приймати рішення про оптимальну кількість груп після аналізу дендрограми. Це дуже зручний підхід для аналізу текстових даних, де кількість категорій може бути невизначеною на початковому етапі дослідження.

ML.NET надає різноманітні моделі для обробки природної мови (NLP), що дозволяють вирішувати широке коло завдань, починаючи від базового аналізу текстів до складних завдань, пов'язаних із глибоким навчанням і нейронними мережами.

Це включає:

- Аналіз настроїв (Sentiment Analysis): Одним із найпоширеніших завдань у NLP є аналіз емоційного забарвлення тексту, що дозволяє визначати, чи є текст позитивним, негативним або нейтральним. Моделі для аналізу настроїв в ML.NET використовують різні методи машинного навчання для класифікації відгуків, постів у соціальних мережах, електронних листів або інших текстів за

емоційним тоном. Це дає змогу автоматично визначати настрої користувачів, що дуже корисно в таких сферах, як обслуговування клієнтів, маркетинг або соціальні мережі[8].

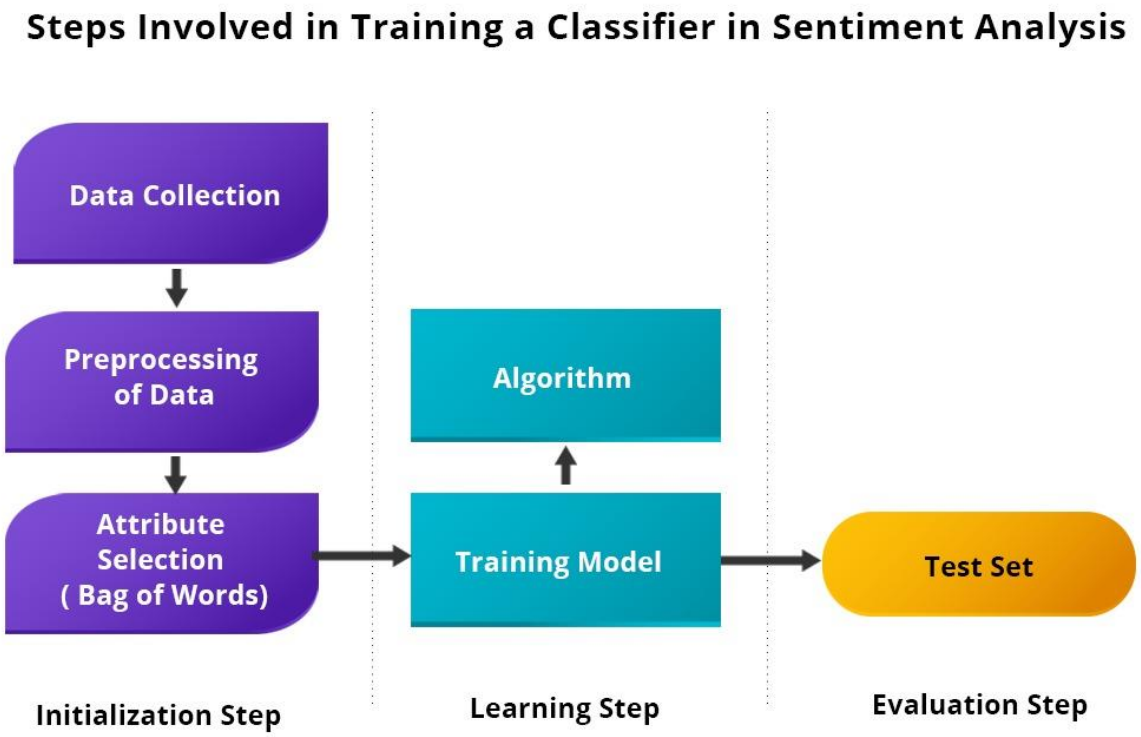


Рис. 1.3 Кроки тренування моделі для аналізу настроїв

- Виявлення сутностей (Named Entity Recognition, NER): Виявлення сутностей є важливим завданням для автоматичної обробки тексту, що полягає у виявленні конкретних імен, дат, локацій, організацій та інших сутностей, що можуть бути значущими в контексті тексту. Наприклад, за допомогою NER можна виділяти імена компаній у статтях, дати подій у новинах або місця в оголошеннях. ML.NET надає алгоритми для таких завдань, що дозволяє ефективно виділяти сутності з великих обсягів текстових даних для подальшого аналізу або класифікації[9].

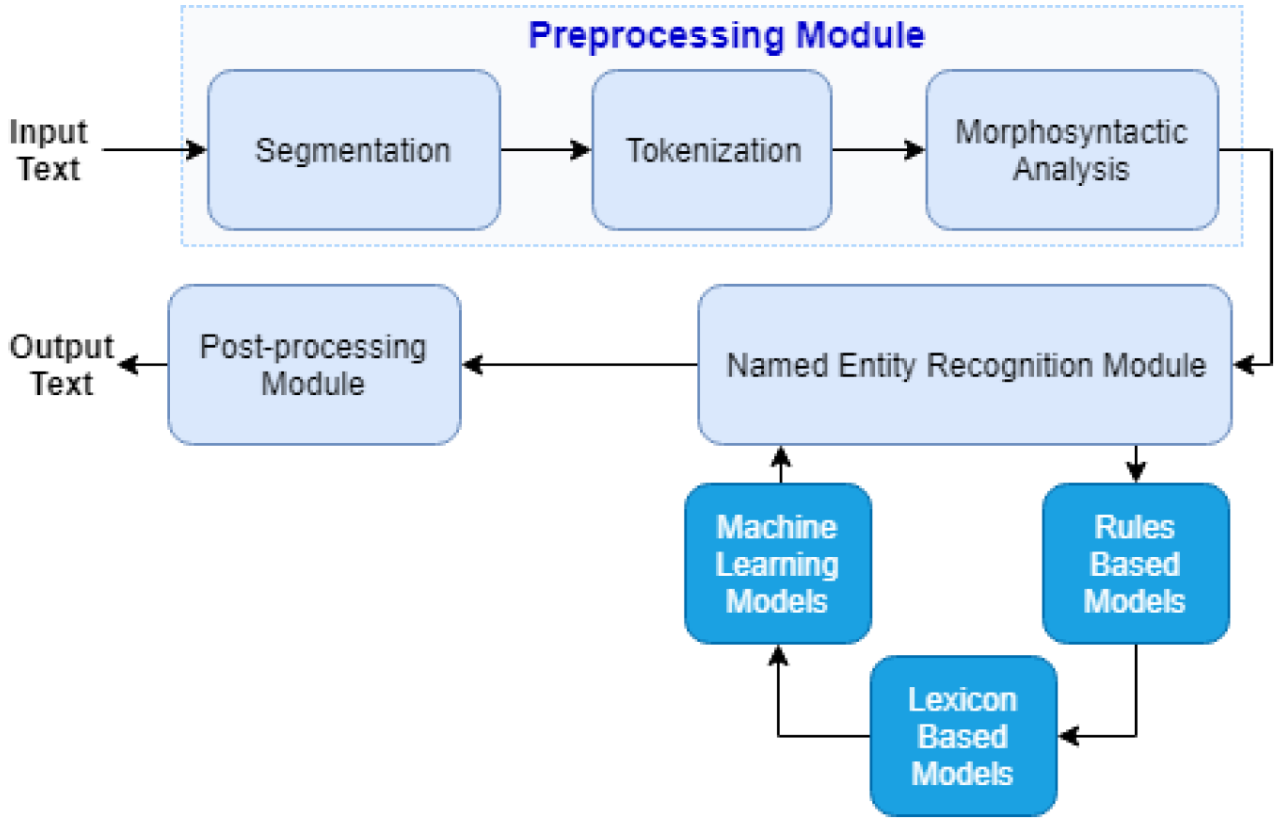


Рис. 1.4 Алгоритм виявлення сутностей

- Витягування характеристик (Feature Extraction): Витягування характеристик з тексту включає в себе процес перетворення тексту в числові вектори, що можуть бути використані для машинного навчання. Це є основою для подальшого аналізу або класифікації тексту. У ML.NET для цього використовуються різні техніки, зокрема методи векторизації на основі термів, такі як TF-IDF (Term Frequency-Inverse Document Frequency), які дозволяють виділити важливі слова та фрази з тексту. В результаті, система отримує числові представлення текстових даних, які можуть бути використані для подальшого аналізу.

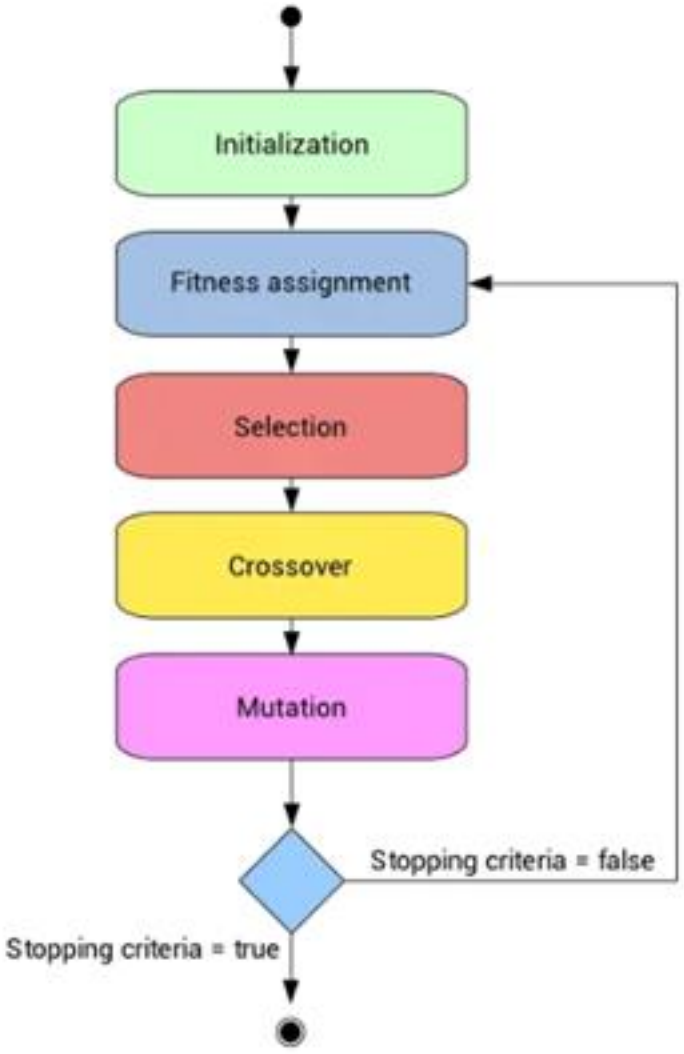


Рис. 1.5 Алгоритм витягування характеристик

- Глибоке навчання (Deep Learning): ML.NET також підтримує використання нейронних мереж, що дозволяє вирішувати більш складні завдання, такі як аналіз контексту, переклад тексту, генерація тексту, а також синтез мови. Моделі глибокого навчання здатні виявляти складні залежності в даних, що дозволяє їм краще розуміти контекст тексту і виконувати більш точні прогнози. Наприклад, нейронні мережі можуть використовуватися для автоматичного перекладу між мовами або для генерації тексту, що відповідає заданому контексту.
- Моделі для обробки великих обсягів даних: Для обробки великих обсягів текстових даних ML.NET забезпечує можливості масштабування та оптимізації моделей, що дозволяє працювати з великими наборами текстів,

наприклад, з великими базами даних новинних статей, відгуків користувачів чи документів. Це забезпечує високу продуктивність при обробці великих обсягів текстової інформації, що є важливим для сучасних систем аналізу даних, особливо в контексті бізнес-аналітики або соціальних медіа[10].

1.4 Проблеми продуктивності при обробці великих обсягів тексту

Обробка великих обсягів текстової інформації є складною задачею, що включає безліч викликів, які можуть значно вплинути на загальну продуктивність системи. Текстові дані часто є неструктурованими, що ускладнює їх обробку, класифікацію та аналіз. Зазначені проблеми можуть проявлятися на різних етапах процесу обробки тексту і включають як технічні, так і концептуальні складнощі. Серед основних проблем, з якими стикаються розробники при роботі з великими обсягами тексту, можна виділити кілька ключових аспектів.

- Часова складність алгоритмів є однією з основних проблем при обробці великих обсягів текстових даних. Багато з алгоритмів, що використовуються для аналізу тексту, мають високу часову складність, що суттєво впливає на загальну ефективність системи, особливо коли йдеться про великі набори даних. Одним з найбільш ресурсоємних етапів у цьому процесі є векторизація тексту, зокрема за допомогою методу TF-IDF (Term Frequency-Inverse Document Frequency). Цей процес потребує обчислення частоти термінів у документах і їх розподілу по всьому корпусу тексту, що призводить до створення великих матриць і відповідно до значних витрат обчислювальних ресурсів. Часова складність таких операцій пропорційна кількості слів і документів, що при обробці великих обсягів інформації суттєво збільшує час виконання і потребує великої кількості пам'яті. Особливу увагу слід приділити складності алгоритмів машинного навчання, які використовуються для класифікації тексту. Алгоритми, такі як логістична регресія, дерева рішень або методи опорних векторів (SVM), потребують значних обчислювальних ресурсів на етапі тренування моделей. Це зумовлено необхідністю виконання численних операцій оптимізації, порівняння розподілів ймовірностей і

розв'язання складних математичних задач. Якщо обсяг навчальних даних значний, це призводить до збільшення часу тренування моделі, що робить алгоритми менш ефективними для обробки великих даних в реальному часі. Ще однією важливою проблемою є часова складність при виконанні операцій, пов'язаних з аналізом змісту тексту, таких як виділення сутностей або виявлення емоційного забарвлення. Алгоритми, що здійснюють синтаксичний і семантичний аналіз, повинні виконувати кілька етапів обробки тексту, включаючи токенізацію, частиномовний аналіз, парсинг та синтаксичний аналіз. Кожен з цих етапів вимагає значних обчислювальних ресурсів, особливо при роботі з великими обсягами даних, оскільки необхідно обробляти кожен елемент тексту окремо. Це ускладнює масштабування систем і може значно знизити їх ефективність при обробці великих обсягів інформації. Крім того, проблеми з часом виконання виникають при обробці текстових даних у реальному часі. Враховуючи швидкий потік інформації, система повинна забезпечувати безперервний процес зчитування, аналізу та видачі результатів з мінімальними затримками. Це вимагає не лише оптимізації алгоритмів, а й належної організації архітектури програмного забезпечення, що включає ефективне збереження даних, кешування результатів і розподілену обробку. Оскільки потоки даних можуть бути великими або варіюватися по обсягу, це накладає додаткові вимоги на швидкість виконання кожної операції. З метою підвищення продуктивності обробки великих обсягів тексту необхідно застосовувати різноманітні стратегії оптимізації. Одним із таких підходів є використання більш ефективних алгоритмів векторизації, таких як зменшення розмірності або використання спарсених матриць, що дозволяють зменшити складність обчислень. Для підвищення швидкості тренування моделей можуть бути застосовані методи паралельних обчислень або розподілені обчислювальні ресурси, що дозволяє значно зменшити час обробки при збереженні точності результатів. Важливою стратегією є також використання спеціалізованих хмарних рішень для обробки великих обсягів даних, що дозволяє збільшити продуктивність за рахунок гнучкої масштабованості ресурсів[11].

- Масштабованість обчислень є однією з ключових вимог при розробці систем для обробки великих обсягів текстових даних, особливо коли йдеться про реальний час або коли дані приходять із значною швидкістю. Масштабованість включає в себе здатність системи адаптувати свої обчислювальні потужності під збільшення обсягів даних, забезпечуючи ефективне їх оброблення без значних втрат продуктивності. Важливо зазначити, що масштабування обчислювальних ресурсів можна здійснювати кількома способами, включаючи горизонтальне та вертикальне масштабування, і кожен з цих підходів має свої переваги та виклики.
- Горизонтальне масштабування, або масштабування "по ширині", полягає в додаванні нових обчислювальних одиниць, таких як сервери або процесорні потоки, до системи, щоб збільшити її загальну потужність. При горизонтальному масштабуванні система розподіляє навантаження між кількома вузлами, що дозволяє обробляти великі обсяги текстових даних за допомогою паралельної обробки. Цей підхід є особливо корисним для систем, які повинні працювати з великими наборами текстів або які мають вимогу до високої швидкості обробки, наприклад, в обробці потокових даних чи в ситуаціях з великими базами даних. Забезпечення горизонтального масштабування може бути складним і вимагати додаткових зусиль для організації та підтримки розподіленої обчислювальної інфраструктури. Це включає в себе завдання по ефективному розподілу даних між різними вузлами, узгодженню роботи кількох серверів і синхронізації результатів обробки. Якщо масштабування не здійснюється правильно, це може призвести до зниження ефективності системи, зокрема, через проблеми з управлінням розподіленими даними, синхронізацією або збільшенням часу на комунікацію між вузлами[12].
- Вертикальне масштабування, або масштабування "по висоті", передбачає збільшення потужності окремих обчислювальних одиниць, таких як сервери або окремі комп'ютери, шляхом додавання процесорних ядер, оперативної пам'яті або потужних графічних процесорів (GPU). Це дозволяє зменшити потребу в складних розподілених системах, оскільки всі обчислення здійснюються на одному сервері, що спрощує архітектуру та забезпечує високу продуктивність для

деяких видів обробки даних.

Вертикальне масштабування має обмеження, пов'язані з фізичними можливостями серверів. Збільшення потужності сервера може бути обмежене його апаратними характеристиками або витратами на енергоспоживання. Тому, для ефективного вертикального масштабування необхідно використовувати апаратні засоби, які здатні підтримувати високі навантаження без зниження продуктивності. Важливим аспектом масштабованості є забезпечення ефективного управління пам'яттю та обчислювальними ресурсами, оскільки великі обсяги текстових даних можуть потребувати значних обсягів пам'яті для зберігання проміжних результатів та виконання складних операцій. Наприклад, при векторизації текстів із використанням методів, таких як TF-IDF або Word2Vec, необхідно працювати з великими матрицями, що вимагають великих обсягів пам'яті. У таких випадках, важливо оптимізувати використання пам'яті, застосовуючи алгоритми, що дозволяють працювати з даними, не завантажуючи їх у всю пам'ять одночасно, або використовуючи техніки стискання даних. Для масштабованих архітектур в реальному часі також необхідно враховувати механізми обробки потокових даних, які дозволяють системам обробляти великі обсяги даних, що надходять безперервно, як це часто трапляється в середовищах з постійним потоком текстової інформації, наприклад, в соціальних мережах чи новинних агентствах. Обробка таких даних вимагає гнучкої інфраструктури, яка може адаптуватися до змінних навантажень і забезпечувати високу швидкість реагування на вхідні запити. Застосування хмарних технологій є ще одним способом забезпечення масштабованості. Використання хмарних сервісів, таких як Microsoft Azure або Amazon Web Services (AWS), дає можливість автоматично збільшувати обчислювальні потужності в залежності від навантаження, що дозволяє значно знизити витрати на інфраструктуру та підвищити ефективність обробки великих даних. Хмарні сервіси також забезпечують можливість використовувати спеціалізовані ресурси, такі як потужні графічні процесори (GPU) для обробки великих обсягів текстових даних у задачах машинного навчання або глибокого навчання.

1.5 Сучасні тренди автоматизації текстової аналітики

Сучасні тренди в автоматизації текстової аналітики свідчать про швидкий розвиток технологій, що дозволяють значно покращити ефективність і масштабованість обробки великих обсягів текстових даних. Враховуючи великий попит на автоматизацію процесів аналізу тексту в різних галузях, від бізнес-аналітики до наукових досліджень, з'являються нові підходи, що використовують передові технології та методи:

- Глибоке навчання (Deep Learning) : Одним із найбільш значних та впливових трендів у сучасному аналізі текстових даних є застосування методів глибокого навчання для обробки природної мови (NLP). Глибокі нейронні мережі, включаючи рекурентні нейронні мережі (RNN), довгі короткочасні пам'яті (LSTM), а також трансформери, такі як BERT і GPT, стали основою нових підходів до вирішення складних задач у сфері обробки тексту. Ці моделі значно покращують точність та ефективність аналізу текстів, дозволяючи комп'ютерам «розуміти» контекст і значення слів у їхніх взаємозв'язках, а не тільки на основі окремих термінів чи фраз.Рекурентні нейронні мережі (RNN) здатні обробляти послідовні дані, такі як текст, де порядок слів має важливе значення. Їхня здатність зберігати інформацію про попередні елементи послідовності дає можливість моделі враховувати контекст в процесі аналізу тексту. Проте, класичні RNN мають обмеження, пов'язані з труднощами у збереженні довготривалої залежності між елементами послідовності. Для подолання цих обмежень були розроблені довгі короткочасні пам'яті (LSTM), які мають здатність зберігати інформацію про більш віддалені частини тексту і ефективно використовувати її для точнішого прогнозування або класифікації. BERT і GPT, представляють собою новий підхід до обробки текстових даних. Вони використовують механізм уваги (attention), який дозволяє моделі фокусуватися на найбільш значущих частинах тексту під час його обробки, незалежно від їхнього місця у послідовності. Це дозволяє значно покращити точність, оскільки модель може приділяти увагу не лише найближчим

словам, а й контексту на всіх рівнях тексту. В результаті такі моделі забезпечують значно глибше розуміння змісту тексту та дозволяють виконувати складні завдання, що вимагають аналізу великих обсягів текстових даних. Цей метод став стандартом у розв'язанні таких задач, як генерація тексту, автоматичне резюмування, переклад мов та виявлення сутностей (named entity recognition).

Наприклад, у задачах оцінки настроїв, моделі глибокого навчання здатні не лише визначити, чи є текст позитивним чи негативним, але й виявляти нюанси емоцій, такі як радість, гнів чи сум, з більшою точністю. Водночас, для завдань, пов'язаних з автоматичним перекладом або резюмуванням, ці моделі дозволяють досягати результатів, які раніше були недосяжними за допомогою традиційних методів, таких як правила на основі фрагментів чи статистичні методи[13].

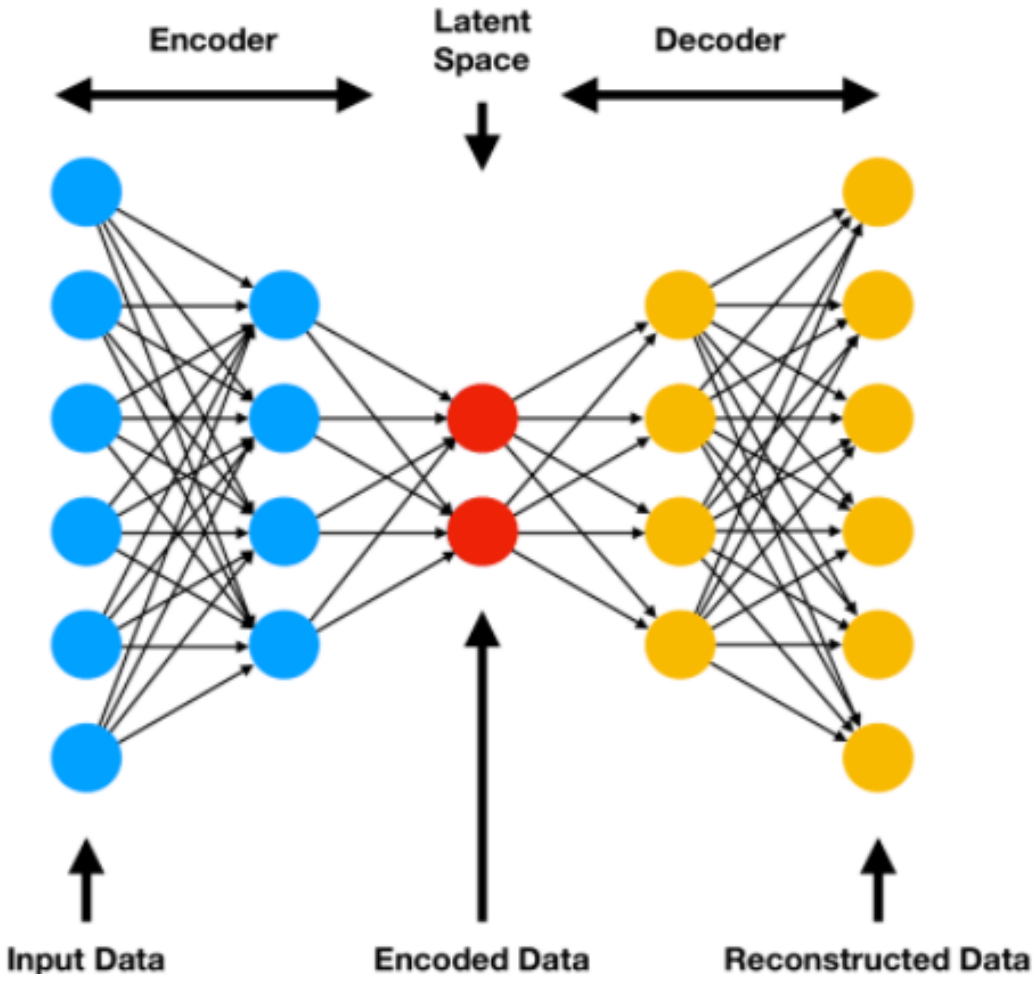


Рис. 1.6 Алгоритм глибокого навчання

- Автоматизація виявлення сутностей (Entity Recognition) є одним з основних завдань в галузі обробки природної мови (NLP), що передбачає автоматичне визначення та класифікацію важливих елементів інформації в текстах. Сутності можуть включати різноманітні категорії, такі як імена осіб, організацій, локацій, дати, числові значення та інші типи, що мають специфічне значення в контексті оброблюваного тексту. Оскільки текстова інформація в більшості випадків є неструктурованою, виявлення сутностей є важливим етапом для перетворення цієї інформації у формат, зручний для подальшого аналізу та обробки. Досягнення значного прогресу в цій області стало можливим завдяки розвитку методів машинного навчання, зокрема глибоких нейронних мереж. Технології, такі як рекурентні нейронні мережі (RNN) і трансформери, значно покращили якість автоматичного виявлення сутностей, дозволяючи системам обробляти текст з високою точністю та враховувати контекст на рівні окремих слів і фраз.

Раніше існуючі методи, такі як правило-орієнтовані системи або статистичні моделі, не могли забезпечити таку ж гнучкість і ефективність при обробці текстів з великими обсягами або складними мовними конструкціями. Завдяки сучасним підходам, автоматичне виявлення сутностей дозволяє не лише здійснювати їх ідентифікацію, але й класифікувати їх за різними категоріями з високим рівнем точності.

Класифікація сутностей забезпечує можливість не лише визначити, що є сутністю, але й зрозуміти її роль або тип у контексті конкретного тексту. Наприклад, імена можуть бути класифіковані як «особи» чи «організації», а дати можуть бути позначені як «події» або «терміни». Цей процес класифікації сутностей має велике значення для організації та структурування текстових даних.

Після виявлення і класифікації сутностей, інформація може бути ефективно витягнута для подальшого аналізу, зберігання або використання в різноманітних додатках, таких як пошукові системи, рекомендаційні системи, системи аналізу настроїв або навіть в правових і медичних дослідженнях.

Більш того, з огляду на великий обсяг даних, що генеруються в різних сферах діяльності, автоматизація виявлення сутностей забезпечує значну економію часу і ресурсів, підвищуючи загальну ефективність системи обробки тексту.

Системи автоматичного виявлення сутностей також можуть бути використані в реальному часі, що робить їх невід'ємною частиною таких сфер, як фінансові послуги, соціальні мережі, медіа та інші.

Крім того, завдяки глибокому навчанню, ці системи стають дедалі більш адаптивними, здатними працювати з текстами різних жанрів, стилів і мов, що дає їм гнучкість і універсальність в застосуванні.Аналіз тем та кластеризація тексту (Topic Modeling) Технології автоматичного виділення тем із тексту (наприклад, Latent Dirichlet Allocation, LDA) дозволяють розподіляти тексти на різні категорії або групи, що стосуються певних тем. Це особливо корисно для обробки великих обсягів документів, таких як наукові статті, новини або публікації в соціальних мережах.

Використання цих методів дозволяє автоматично класифікувати текстову інформацію без необхідності ручного визначення категорій, що значно спрощує обробку великих масивів даних і допомагає виявляти приховані тренди та кореляції[14].

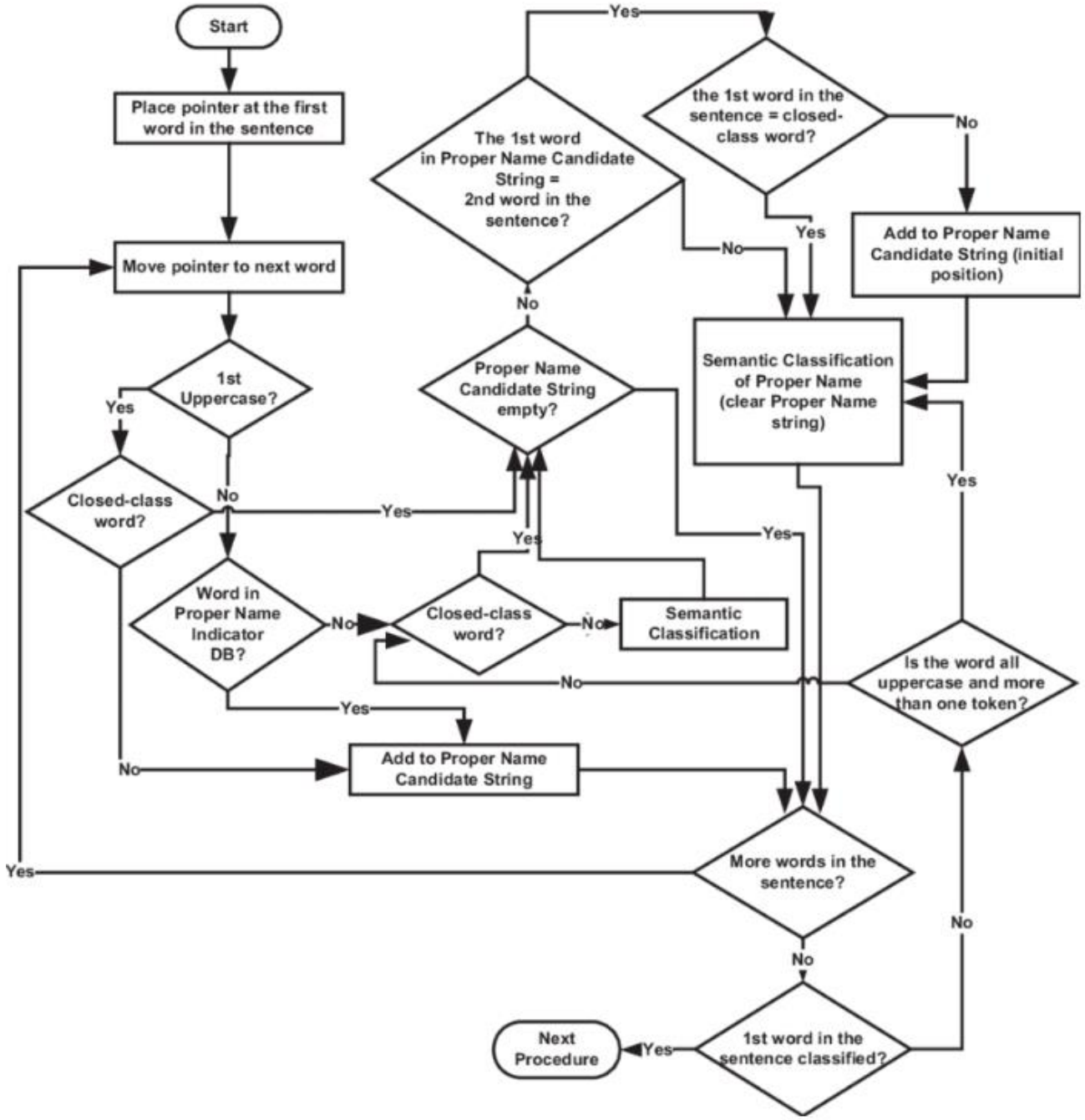


Рис 1.7 Алгоритм автоматизації виявлення сутностей

- Автоматизація витягання ключових слів та фраз (Keyword Extraction) є важливим етапом обробки текстової інформації, спрямованим на виділення термінів або фраз, які несуть найбільшу семантичну вагу і є суттєвими для розуміння змісту тексту. У традиційних методах обробки тексту ця задача часто виконувалась на основі простих статистичних підходів, однак з розвитком машинного навчання та глибоких нейронних мереж стало можливим досягти

значного прогресу в точності та ефективності витягання важливих компонентів тексту. Автоматизація цього процесу відіграє важливу роль у ряді застосувань, таких як класифікація документів, побудова рекомендованих систем, аналіз настроїв та оптимізація пошукових систем.

Одним з основних методів для автоматичного витягання ключових слів є використання статистичних підходів, таких як метод TF-IDF (Term Frequency-Inverse Document Frequency), який оцінює важливість терміна в контексті всього корпусу текстів, враховуючи як частоту появи терміна в конкретному документі, так і його рідкість у всьому наборі документів. Це дозволяє виділяти терміни, що характеризують зміст тексту та є специфічними для кожного документу, що є корисним при побудові індексів для пошукових систем або для класифікації. Проте статистичні методи, такі як TF-IDF, мають певні обмеження, зокрема, вони не завжди здатні врахувати контекст і семантичні зв’язки між термінами в документі.

Для подолання цих обмежень застосовуються більш складні підходи, зокрема використання моделей, заснованих на нейронних мережах, таких як Word2Vec, GloVe або трансформери. Ці моделі мають здатність захоплювати більш складні зв'язки між словами та їх контекстуальними значеннями, що дозволяє не лише виділяти окремі ключові слова, але й виявляти важливі фрази або концепти, які можуть бути неоднозначно виражені в тексті.

Особливо важливою є здатність новітніх моделей глибокого навчання, таких як трансформери (BERT, GPT тощо), здійснювати витягання ключових слів з урахуванням контексту, в якому ці терміни з’являються. Це дозволяє системам автоматичного витягання ключових слів ефективно працювати з більш складними структурами тексту, такими як іронія, метафори або неоднозначні висловлювання, що раніше було важко досягнути за допомогою традиційних статистичних методів[15].

Витягання ключових слів та фраз також включає задачі, пов’язані з ідентифікацією семантичних зв’язків між термінами в межах одного документу або між документами. Такі задачі можуть бути вирішені за допомогою різноманітних підходів машинного навчання, включаючи навчання на великих текстових корпусах для формування контекстуальних представлень слів, що дає можливість визначити найбільш важливі фрази і словосполучення.

Автоматичне резюмування текстів Одним із важливих напрямків є автоматичне резюмування великих обсягів тексту, яке дозволяє зменшити час на обробку документів, надаючи користувачам лише найбільш суттєву інформацію. Використання методів глибокого навчання дозволяє генерувати змістовні анотації для великих текстів, що є важливим для новинних сайтів, наукових публікацій і юридичних документів. Це дозволяє значно підвищити ефективність роботи з великими наборами текстів[16].

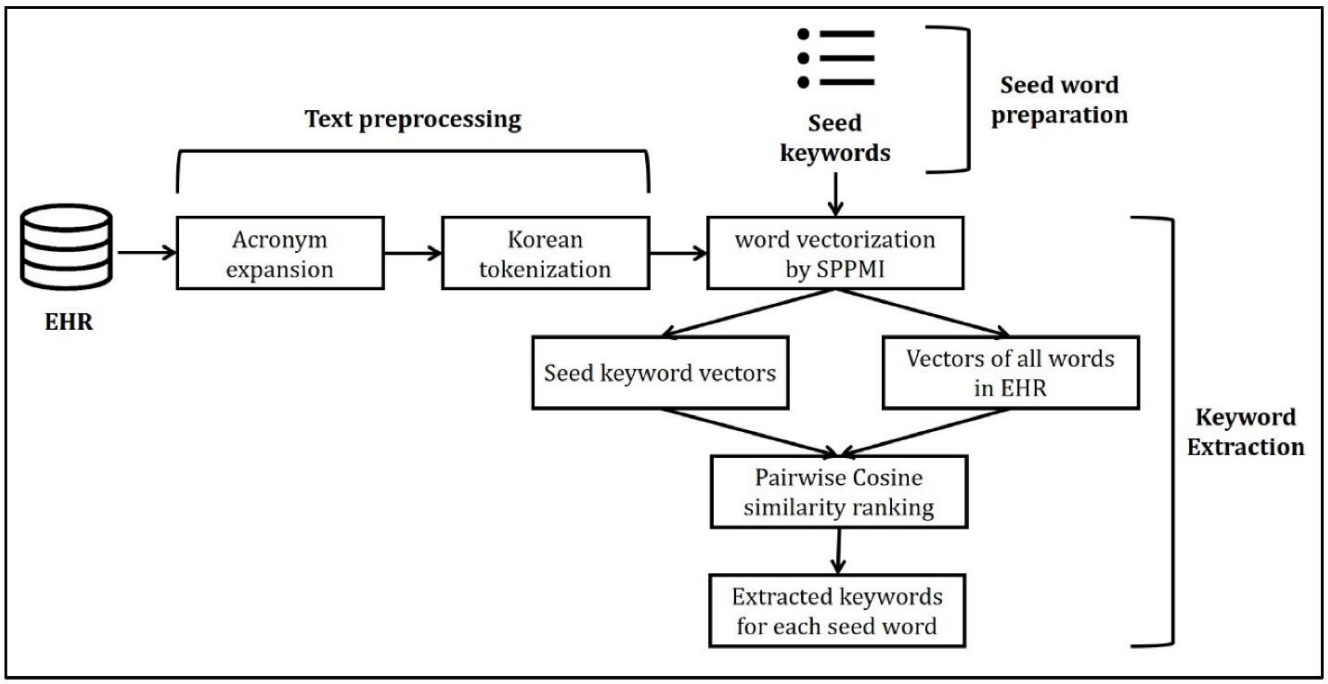


Рис 1.8 Алгоритм витягання ключових слів та фраз

2. АНАЛІЗ МЕТОДІВ АВТОМАТИЗАЦІЇ АНАЛІЗУ ТЕКСТУ

2.1 Метрики для оцінки ефективності текстового аналізу

Оцінка ефективності алгоритмів текстового аналізу є важливою складовою процесу розробки та впровадження систем обробки природної мови. Для цього використовуються різноманітні метрики, які дозволяють кількісно виміряти точність, повноту, швидкість та інші показники результативності застосовуваних моделей. Вибір відповідних метрик залежить від типу задачі, що розв'язується, та від специфіки даних. Найбільш широко використовуваними метриками для оцінки ефективності текстового аналізу є точність, повнота, F1-міра, точність на рівні класифікації та матриця плутанини[17].

Точність (accuracy) є однією з найбільш широко використовуваних метрик для оцінки ефективності моделей класифікації. Вона визначається як відношення кількості правильно класифікованих елементів до загальної кількості елементів у наборі даних(2.1).

$$\text{Accuracy} = \frac{\text{TP+TN}}{TP+TN+FP+FN} \tag{2.1}$$

де:

- **TP (True Positives)** — кількість позитивних елементів, які були правильно класифіковані як позитивні;
- **TN (True Negatives)** — кількість негативних елементів, які були правильно класифіковані як негативні;
- **FP (False Positives)** — кількість негативних елементів, які були неправильно класифіковані як позитивні;
- **FN (False Negatives)** — кількість позитивних елементів, які були неправильно класифіковані як негативні.

Точність є корисною метрикою, оскільки вона дає загальне уявлення про те, як добре модель працює в цілому. Вона дає відношення правильно передбачених результатів до всіх спроб. Якщо класи в наборі даних збалансовані, тобто кількість елементів у кожному класі приблизно однакова, то точність може бути досить інформативною і представляти реальний рівень продуктивності моделі. Проте точність стає менш корисною та навіть може вводити в оману, коли дані мають значну диспропорцію в розподілі класів. У таких випадках модель може демонструвати високу точність, навіть якщо вона ігнорує рідкісні класи, що може бути критично важливим у певних ситуаціях.

Наприклад, у задачі фільтрації спаму модель може класифікувати більшість електронних листів як «не спам», навіть якщо вона пропускає деякі важливі випадки спаму. У цьому випадку точність буде високою, оскільки більшість листів дійсно є не спамом, однак модель не виконує свої функції коректно. Тому точність може бути не найкращим вибором для задач, де класи мають значну асиметрію або де потрібно точно класифікувати елементи рідкісних класів. В таких ситуаціях зазвичай використовують додаткові метрики, такі як повнота , точність та F1-міра, які дозволяють краще оцінити здатність моделі досягати бажаних результатів по кожному класу окремо.

Точність (Precision) є важливою метрикою, що дозволяє виміряти якість класифікації моделі в контексті позитивних класів. Вона показує, скільки з елементів, які були класифіковані як належні до певного класу, дійсно належать до цього класу.

Формально точність можна обчислити за формулою 2.2:

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2.2}$$

де:

TP (True Positives) — кількість елементів, які правильно були класифіковані як позитивні.

FP (False Positives) — кількість елементів, які були неправильно класифіковані як позитивні, хоча насправді вони належать до негативного класу.

Точність є критично важливою в тих задачах, де важливо уникнути хибних спрацьовувань або помилок, що призводять до визнання негативних елементів як позитивних. Одним із таких прикладів є фільтрація спаму, де неправильно класифікований електронний лист як спам (хибнопозитивний результат) може призвести до втрати важливої інформації. Важливо відзначити, що точність має велике значення в контексті завдань, де потрібно мінімізувати витрати на хибнопозитивні передбачення, оскільки кожне таке передбачення може бути витратним чи некорисним. У контексті аналізу тексту точність допомагає у визначенні того, наскільки коректно модель ідентифікує важливі ключові слова, сутності чи фільтрує спам.

F1-міра є метрикою, що поєднує в собі дві ключові характеристики ефективності класифікації — точність та повноту (recall). Вона є гарним варіантом для оцінки моделей, де необхідно досягти балансу між мінімізацією хибнопозитивних і хибнонегативних результатів. F1-міра особливо корисна в задачах, де одне значення (точність або повнота) не може бути пріоритетним, а потрібно досягти гармонії між ними (2.3).

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{2.3}$$

де:

Precision — точність, яка показує, яка частина позитивних передбачень є правильними.

Recall — повнота, що вимірює, яку частину всіх дійсних позитивних елементів модель змогла виявити.

F1-міра стає надзвичайно корисною, коли є необхідність не лише зберігати точність у класифікації, але й забезпечити високу здатність до виявлення всіх важливих елементів або класів, навіть якщо це може призвести до деяких помилок. Наприклад, у завданнях виявлення фальшивих новин модель, яка має високу точність, але низьку повноту, може пропускати важливі випадки, тоді як F1-міра дозволяє зберегти баланс між цими двома характеристиками.

F1-міра є вкрай корисною в задачах, де одна із помилок (хибнопозитивна чи хибнонегативна) може мати серйозні наслідки. Наприклад, у випадках автоматичної класифікації медичних записів, де важливо виявляти усі можливі захворювання (повнота), але при цьому також важливо мінімізувати помилкові діагнози (точність). В такому контексті F1-міра дозволяє оптимізувати модель, враховуючи як виявлення важливих елементів, так і уникання хибних класифікацій.

Матриця плутанини є важливим інструментом для оцінки ефективності алгоритмів класифікації, оскільки дозволяє наочно побачити, як модель працює з різними класами. Вона особливо корисна при аналізі двокласових класифікаційних задач, хоча може бути застосована і до багатокласових задач.

Матриця плутанини містить чотири основні величини, які вказують на правильні та неправильні передбачення моделі, а саме:

True Positives (TP) — це кількість елементів, які модель правильно класифікувала як позитивні. Іншими словами, це кількість позитивних випадків, які були справедливо віднесені до позитивного класу. Наприклад, в задачі виявлення спаму це будуть всі електронні листи, які справді є спамом і були правильно класифіковані моделлю як спам.

True Negatives (TN) — це кількість елементів, які модель правильно класифікувала як негативні. Це негативні випадки, які були правильно віднесені до негативного класу. У контексті фільтрації спаму, це всі листи, які не є спамом і були правильно віднесені до класу "не спам"[18].

False Positives (FP) — це кількість елементів, які модель неправильно класифікувала як позитивні, хоча насправді вони належать до негативного класу. Це хибні позитивні результати, які можуть бути небажаними, оскільки вони показують, що модель визнала щось позитивним, хоча це насправді негативне. У задачі фільтрації спаму це будуть листи, які не є спамом, але модель помилково їх класифікувала як спам (хибнопозитивні).

False Negatives (FN) — це кількість елементів, які модель неправильно класифікувала як негативні, хоча вони належать до позитивного класу. Це хибні

негативні результати, коли модель не змогла виявити важливі позитивні елементи. В контексті спаму це будуть листи, які є спамом, але модель помилково класифікувала їх як не спам (хибнонегативні)[19].

Матриця плутанини дозволяє глибше зрозуміти, як модель справляється з різними категоріями та яких типів помилок вона найбільше припускається. Вона є основою для багатьох метрик, таких як точність (Accuracy), точність (Precision), повнота (Recall), F1-міра, а також для побудови ROC-кривих та обчислення AUC-значень.

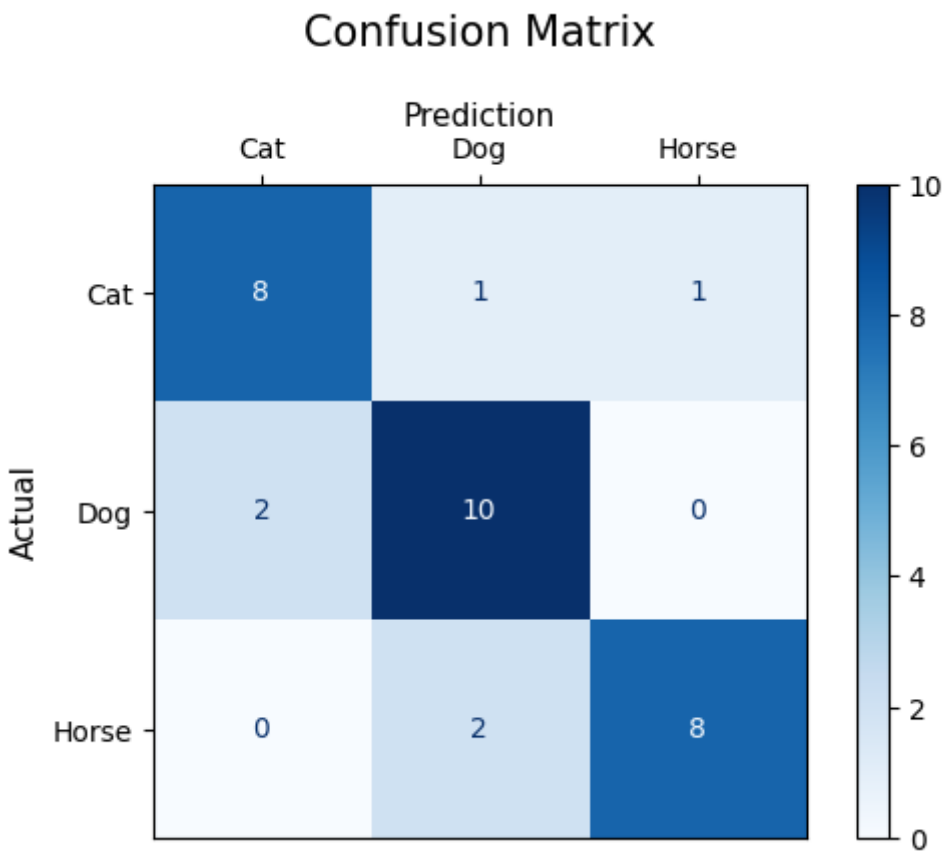


Рис 2.1 Приклад матриці плутанини

Індекс Жаккара (Jaccard Index) є метрикою, яка вимірює подібність між двома наборами об'єктів. У контексті текстового аналізу цей індекс використовується для оцінки схожості між двома текстами або документами на основі їхніх термінів (наприклад, слів або фраз). Індекс Жаккара обчислюється як відношення кількості спільних елементів двох наборів до кількості елементів, які

зустрічаються хоча б в одному з наборів. (2.4)

$$\text{Jaccard} = \frac{|A \cap B|}{|A \cup B|} \tag{2.4}$$

де:

A та B — це два набори термінів, які представляють два документи або тексти;

A∩B — це перетин двох наборів (терміни, які зустрічаються в обох текстах);

A∪B— це об'єднання двох наборів (усі терміни, що зустрічаються хоча б в одному з текстів).

Цей індекс варіюється від 0 до 1, де значення 1 вказує на те, що тексти ідентичні (усі терміни збігаються), а значення 0 свідчить про повну відсутність спільних термінів між двома текстами. Індекс Жаккара є корисним у задачах класифікації текстів, пошуку схожих документів, а також у виявленні дублюючих або схожих даних в великих текстових корпусах[20].

Log-Loss (також відома як логарифмічна втрата або крос-ентропія) є метрикою, яка використовується для оцінки моделей, що передбачають ймовірності класів замість конкретних категорій. Ця метрика вимірює, наскільки добре модель передбачає ймовірності класів для кожного елемента, порівнюючи ці ймовірності з фактичними мітками класів. Log-Loss обчислюється на основі логарифмічної функції, яка штрафує модель за більшу помилку у випадку високих ймовірностей для невірних класів. Чим більше ймовірність, яку модель призначає неправильному класу, тим більше значення Log-Loss, що вказує на погану якість передбачення. У випадку правильно передбаченого класу, значення Log-Loss наближається до нуля(2.5).

$$\text{LogLoss} = -\text{N1i} = 1\sum\text{Nc} = 1\sum\text{Cyi}, \text{clog}(\text{pi}, \text{c}) \tag{2.5}$$

де:

- N — кількість елементів (тобто кількість зразків у тестовому наборі),
- C — кількість класів,

- yi,cy — це фактичний клас для елемента iii, де yi,c=1, якщо елемент належить класу c, і yi,c = 0 в іншому випадку,
- pi,c— це ймовірність, яку модель приписує класу c для елемента iii.

Log-Loss вимірює середнє значення логарифмічних помилок для всіх класів і зразків. Чим менше значення Log-Loss, тим краща модель, оскільки це вказує на меншу відстань між ймовірностями, передбаченими моделлю, і фактичними класами. Log-Loss є особливо корисною в задачах багатокласової класифікації, де потрібно оцінити точність ймовірнісних прогнозів для кожного з класів[21].

Log-Loss є критично важливою метрикою в задачах, де модель не лише передбачає категорії, але й генерує ймовірності для кожної категорії. Це дозволяє використовувати Log-Loss для задач, в яких потрібно не тільки точно класифікувати елементи, але й прогнозувати ймовірність їх належності до певних класів. Наприклад, у задачах прогнозування, де точність класифікації важлива, однак також необхідно знати, наскільки впевнена модель у своїх передбаченнях, Log-Loss може надати додаткову інформацію про якість моделі.

У задачах, де модель може надавати ймовірнісні оцінки для кожного класу, Log-Loss дозволяє точно оцінити ефективність моделі в усіх категоріях, що дає змогу здійснити точну налаштування гіперпараметрів і обрати модель, яка найкраще відповідає вимогам задачі.

2.2 Огляд методів та інструментів для текстового аналізу

Текстовий аналіз, або обробка природної мови (NLP), є однією з найважливіших галузей сучасної науки та технологій, оскільки він дозволяє автоматично обробляти та інтерпретувати текстову інформацію, яка складає більшість даних у цифровому вигляді. Технології текстового аналізу активно використовуються у таких сферах, як машинне навчання, інформаційний пошук, автоматизація бізнес-процесів, медична діагностика, фінансовий аналіз, а також у багатьох інших галузях, що мають справу з великими обсягами текстових даних.

Завдяки швидкому розвитку методів машинного навчання та глибокого навчання, текстовий аналіз перетворився на потужний інструмент для вирішення широкого спектру завдань. Ці методи дозволяють ефективно обробляти тексти будь-якої складності, автоматизуючи процеси, які раніше потребували ручної праці та великих часових витрат.

Основні методи текстового аналізу включають в себе кілька ключових етапів: підготовка текстових даних, векторизація тексту, класифікація, кластеризація, а також витягнення сутностей та аналіз настроїв. Кожен з цих етапів вимагає використання специфічних інструментів і технік, що дозволяють отримати потрібні результати на основі текстової інформації[22].

Векторизація тексту дозволяє перетворити текстову інформацію у числову форму, що дає змогу використовувати її в різноманітних алгоритмах машинного навчання та інших методах аналізу. Існують різні підходи до векторизації, кожен з яких має свої особливості та підходить для вирішення конкретних завдань.

Метод TF-IDF є класичним підходом до векторизації тексту, який використовується для оцінки важливості термінів у конкретному документі або в колекції документів. Цей метод заснований на двох основних складових: Term Frequency (TF) та Inverse Document Frequency (IDF)[23].

Term Frequency (TF) вимірює частоту виникнення терміну в конкретному документі. Це значення може бути простою кількістю входжень терміна в текст або нормованою величиною, що враховує загальну кількість слів у документі. TF дозволяє оцінити, наскільки часто термін зустрічається у вибраному документі, що дає змогу визначити, які слова є найбільш значущими для даного тексту.

Однак такий підхід може призвести до зміщень у випадку документів різного обсягу. Наприклад, слово може частіше зустрічатися у великому тексті просто через його більший розмір. Тому зазвичай використовують нормовану форму TF, яка враховує загальну кількість слів у документі:

Inverse Document Frequency (IDF) вимірює, наскільки рідко термін зустрічається в усьому корпусі документів. Ідея полягає в тому, що слова, які зустрічаються рідше, є більш важливими для розрізнення текстів, тоді як слова, що

зустрічаються часто (наприклад, загальні слова типу "і", "в", "на"), мають меншу інформаційну цінність(2.6).

$$\mathrm{TF-IDF}(t, d) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t) \quad (2.6)$$

де:

t — термін,

d — документ.

У результаті, чим вищий результат TF-IDF для терміна, тим важливіший цей термін для конкретного документа в контексті всього корпусу текстів. Метод TF-IDF широко використовується для таких завдань, як класифікація текстів, фільтрація спаму, пошукові системи, де потрібно визначити найбільш значущі слова для конкретного документа серед великого обсягу текстових даних.

TF часто використовується для визначення ключових слів у тексті, де терміни з вищою частотою розглядаються як потенційно більш значущі.

У тематичному аналізі TF допомагає виявити основні ідеї чи теми документа.

У пошукових системах TF визначає релевантність документа для запиту, підкреслюючи тексти, які частіше містять пошукові слова.

У кластеризації та класифікації текстів TF використовується для створення векторних представлень документів, що дозволяє застосовувати алгоритми машинного навчання.

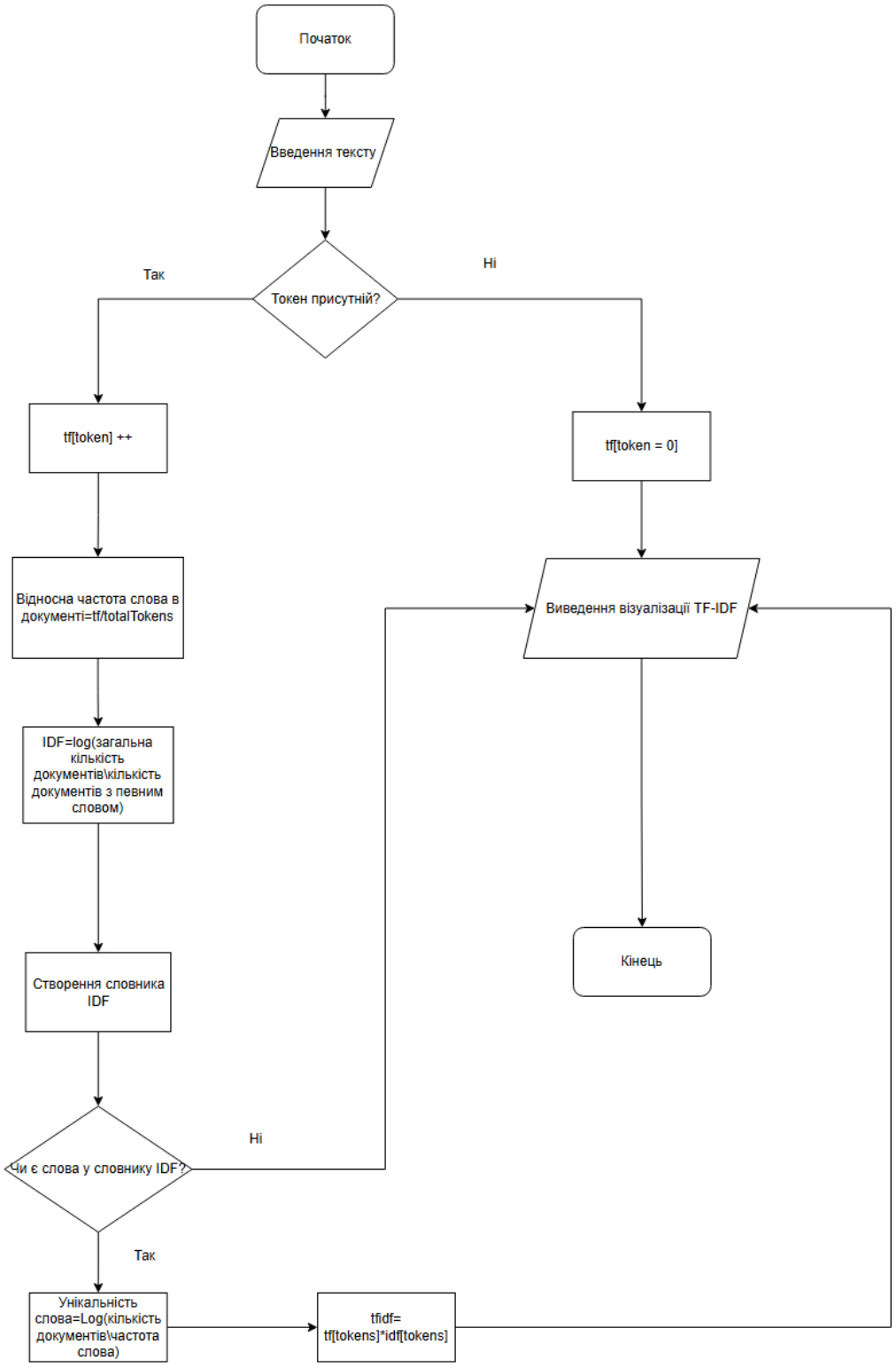


Рис 2.2 Алгоритм TF-IDF

Word2Vec є однією з найбільш популярних моделей для створення векторних представлень слів, заснованих на контексті, у якому ці слова зустрічаються. Вона належить до класу моделей, що використовують методи глибокого навчання для

трансформації тексту в числові вектори, що дозволяє більш ефективно працювати з текстовими даними в завданнях машинного навчання та обробки природної мови. Однією з основних характеристик Word2Vec є здатність відображати семантичну схожість між словами: чим більше схожі контексти, в яких зустрічаються два слова, тим схожіші їхні векторні представлення. У Word2Vec є дві основні архітектури: Skip-Gram та Continuous Bag of Words (CBOW), кожна з яких має свої специфікації та підходи до навчання моделі.

Архітектура Skip-Gram орієнтована на те, щоб для кожного вхідного слова передбачити навколишні слова в контексті. Це означає, що модель отримує одне слово як вхід і намагається передбачити найближчі слова, які з'являються до та після цього слова в тексті. Основний принцип полягає в тому, що чим більше контекстуальних слів модель може передбачити для заданого слова, тим точніше вона може навчити його представлення. Алгоритм Skip-Gram працює таким чином, що вхідним елементом є певне слово, а модель намагається знайти ймовірності для кожного з навколишніх слів у заданому контексті. Для цього вхідне слово перетворюється на вектор, який потім порівнюється з іншими словами в корпусі, що допомагає моделі встановити ймовірність появи кожного з цих слів у конкретному контексті. Основною перевагою Skip-Gram є те, що цей метод добре працює з великими корпусами текстів і здатний ефективно навчати представлення для рідкісних слів. Оскільки модель фокусується на передбаченні навколишніх слів, вона має можливість глибше захоплювати контекстуальні залежності, що дозволяє зберігати багатше значення навіть для слів, що зустрічаються рідше в тексті. Це робить Skip-Gram особливо корисним у випадках, коли важливо отримати точні представлення для слів, які не є частими, або для аналізу складних контекстів. Цей підходить для задач, де необхідно добре розуміти, як слово використовується в різних контекстах. Наприклад, він може бути корисним у тих випадках, коли модель має виявляти приховані зв'язки між словами, навіть якщо вони не зустрічаються часто, що може бути важливим для таких завдань, як пошук синонімів, семантичний аналіз тексту або навіть класифікація текстів на основі їх змісту.

Модель Continuous Bag of Words (CBOW) використовує контекст для передбачення поточного слова. Вона працює шляхом аналізу кількох слів навколо цільового слова, тобто контексту, і на основі цього контексту намагається передбачити ймовірність поточного слова. Алгоритм CBOW приймає на вхід векторні представлення слів з контексту і використовує їх для оцінки ймовірності поточного слова. В результаті цього процесу модель створює середнє представлення контекстуальних слів, яке згодом використовується для прогнозування цільового слова. Перевагою CBOW є її ефективність з обчислювальної точки зору. Модель працює швидше, оскільки в ній контекст слів агрегується в одне середнє представлення, що дозволяє значно зменшити кількість обчислень порівняно з більш складними підходами. Це робить CBOW ідеальним вибором для роботи з великими наборами текстових даних, де важлива швидкість обробки. Цей підхід ефективно працює з частими словами і здатен швидко генерувати векторні представлення, він може не бути таким точним у вирішенні складних семантичних задач. Це пов'язано з тим, що CBOW використовує середнє значення контексту, що може спрощувати виявлення тонких семантичних зв'язків між словами, особливо якщо контекст складається з різнорідних термінів.

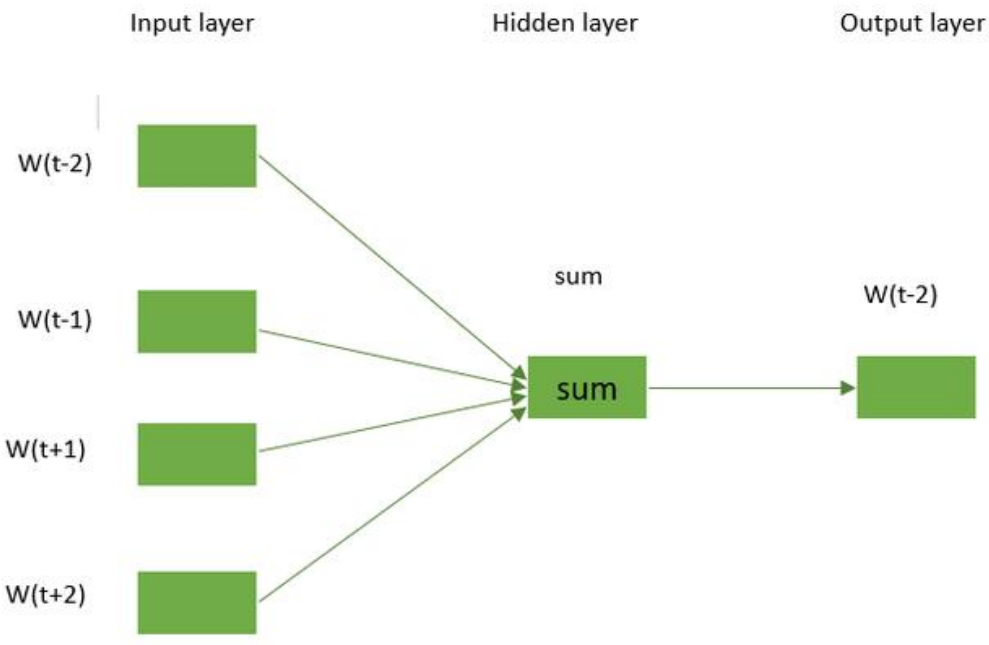


Рис 2.3 Модель CBOW

Для порівняння методів представимо таблицю 2.1:

Таблиця 2.1

Порівняльна характеристика методів

Характеристика	TF-IDF	Word2Vec	GloVe
Простота реалізації	Проста реалізація ефективний для основних задач	Складніша реалізація, потребує глибокого навчання	Вимагає складної факторизації матриць
Контекстуальність	Не враховує контекстуальні залежності між словами	Враховує контекстуальні залежності між словами	Враховує як локальні, так і глобальні контексти
Обчислювальні вимоги	Легкий з точки зору обчислень	Потребує великих обсягів даних та ресурсів	Вимагає великих обчислювальних ресурсів для факторизації
Врахування синонімії/антонімію	Не враховує синонімію чи антонімію	Враховує семантичну схожість між словами	Може враховувати схожість і взаємозв'язки між словами
Застосування	Добре підходить для простих задач класифікації та пошукових систе	Краще для складних завдань, що вимагають глибокого розуміння контексту	Підходить для задач, де важливі глобальні взаємозв'язки між словами
Залежність від даних	Незалежний від великого обсягу даних	Потребує великого обсягу даних для навчання	Потребує великих корпусів текстів для факторизації
Обробка синонімів	Не обробляє синоніми	Обробляє синоніми через схожість векторів	Обробляє синоніми через схожість векторів
Обмеження	Не обробляє контекст або синонімію	Чутливий до шуму в даних	Високі обчислювальні витрати при факторизації

2.2.1 **Використання ML.NET для класифікації тексту**

ML.NET є потужною бібліотекою для машинного навчання, розробленою Microsoft, що дозволяє інтегрувати можливості машинного навчання в .NET додатки. Вона підтримує широкий спектр задач, зокрема класифікацію тексту, що є важливим для аналізу великих обсягів текстових даних.

Для класифікації тексту в ML.NET використовується вбудовані інструменти для обробки текстових даних, які дозволяють перетворювати текст у числові ознаки, необхідні для навчання моделей. Одним із основних етапів є векторизація тексту, що здійснюється через методи, такі як TF-IDF або TextFeaturizingEstimator. Цей процес перетворює текст в числовий формат, що робить його доступним для машинного навчання.

ML.NET надає широкі можливості для класифікації тексту за допомогою різноманітних алгоритмів, які можна адаптувати до конкретних потреб і типів даних. Оскільки текстові дані часто мають велику кількість варіацій і складних семантичних зв'язків, важливо використовувати методи, які дозволяють досягти високої точності та гнучкості при класифікації.

Алгоритми для класифікації тексту одним із основних алгоритмів для класифікації тексту в ML.NET є логістична регресія, яка є одним з найпоширеніших методів для бінарної класифікації. Вона застосовується для задач, де необхідно передбачити, чи належить певний текст до одного з двох класів. Логістична регресія працює на основі ймовірностей і здатна ефективно класифікувати текстові дані, розподіляючи їх на два класи.

Окрім логістичної регресії, ML.NET також підтримує методи, засновані на випадкових лісах. Випадковий ліс є ансамблевим методом машинного навчання, який поєднує кілька дерев рішень для досягнення кращої точності класифікації. Цей метод особливо корисний для складних задач, де дані можуть мати різноманітні взаємозв'язки і залежності. Він підходить для багатокласових задач, де необхідно класифікувати текст на декілька категорій одночасно.

Завдяки використанню таких алгоритмів ML.NET може працювати як з простими бінарними задачами, так і з більш складними багатокласовими задачами,

де кількість класів може бути значною.

Оцінка ефективності моделі: після тренування моделей важливо оцінити їхню ефективність, використовуючи різноманітні метрики. Однією з таких метрик є точність (accuracy), яка вказує на загальний відсоток правильно класифікованих елементів. Однак точність може бути не зовсім коректною, якщо класи в наборі даних сильно дисбалансовані. У таких випадках більше значення мають інші метрики, як-от повнота (recall) і точність (precision), які дозволяють оцінити, як добре модель визначає позитивні та негативні класи, не помилково зараховуючи одні до інших.

Особливо корисною є F1-міра, яка поєднує точність і повноту в одній метриці. Вона дає чітке уявлення про те, як модель працює в балансі між цими двома аспектами, що особливо важливо в задачах, де критично важливо уникнути як хибнопозитивних, так і хибнонегативних результатів.

Прогнозування на нових даних: після того як модель навчена, її можна використовувати для прогнозування на нових текстових даних. ML.NET дозволяє безперешкодно застосовувати вже навчену модель для класифікації текстів, які не входили в навчальний набір. Це забезпечує можливість автоматизації процесів, таких як фільтрація контенту або автоматичний аналіз текстів у реальному часі. Наприклад, модель може класифікувати нові відгуки користувачів, новини або коментарі в соціальних мережах, швидко визначаючи їхню тематику або емоційну забарвленість.

Один з ключових аспектів, який робить ML.NET потужним інструментом для класифікації тексту, — це можливість інтегрувати модель безпосередньо в .NET додатки. Це дає змогу розробникам застосовувати модель в реальних умовах, автоматизуючи аналіз тексту в широкому спектрі додатків, від корпоративних систем до мобільних додатків та веб-сервісів.

2.2.2 Робота з Sentiment Analysis в ML.NET

Процес класифікації текстів, спрямований на визначення емоційного забарвлення або настрою, що виражений у текстових даних. Це один із найбільш

поширених випадків текстового аналізу, що використовується для оцінки позитивного, негативного або нейтрального настрою тексту. Наприклад, в аналізі відгуків користувачів, пошукових запитів або публікацій в соціальних мережах, метою є класифікація текстів на категорії "позитивний", "негативний" чи "нейтральний".

ML.NET надає зручний інтерфейс для реалізації таких задач, дозволяючи створювати моделі для аналізу настроїв, використовуючи стандартні алгоритми машинного навчання.

Для початку необхідно підготувати набір даних, який містить текстову інформацію та відповідні мітки настрою (наприклад, "позитивний", "негативний", "нейтральний"). У ML.NET для роботи з такими даними використовуються класичні об'єкти IDataView, що є універсальним типом даних для обробки інформації.

```
public class SentimentData
{
    Ссылок: 0
    public string Text { get; set; }
    Ссылок: 0
    public bool Label { get; set; }
}
```

Рис 2.4 Приклад стурктури даних для роботи з ML.NET

Тексти необхідно перетворити в числові векторні представлення, щоб вони могли бути оброблені алгоритмами машинного навчання. Для цього ML.NET пропонує кілька методів векторизації, таких як TF-IDF або Bag of Words. Векторизація є ключовим етапом для подальшого навчання моделі.

У ML.NET векторизація тексту здійснюється за допомогою трансформерів, таких як TextFeaturizingEstimator, що дозволяє перетворити текстові дані у векторні представлення.

ML.NET дозволяє використовувати різноманітні алгоритми для навчання моделей класифікації тексту. Для задачі аналізу настроїв, один з найбільш часто використовуваних алгоритмів — це логістична регресія. Алгоритм працює з

бінарними мітками ("позитивний" або "негативний"), що є типовим для таких задач. Для багатокласових завдань також можна використовувати інші моделі, наприклад, випадковий ліс (Random Forest) або методи підтримки векторних машин.

Для оцінки якості моделі в ML.NET використовуються різноманітні метрики, такі як точність (accuracy)**,** повнота (recall)**,** точність (precision) та F1-міра. Залежно від специфіки задачі, можна вибирати найбільш релевантні метрики для оцінки продуктивності моделі.

Після того як модель була навчена, її можна використовувати для прогнозування настроїв нових текстів. За допомогою методу Predict, можна отримати клас прогнозу для кожного новому тексту.

Однією з сильних сторін ML.NET є його здатність інтегруватися з реальними додатками. Після навчання моделі, її можна впровадити в систему для автоматичного аналізу відгуків, публікацій або коментарів в реальному часі. Це дозволяє швидко отримувати результат класифікації і використовувати його для прийняття рішень або подальшого аналізу.

2.2.3 Тестування моделей за допомогою BenchmarkDotNet

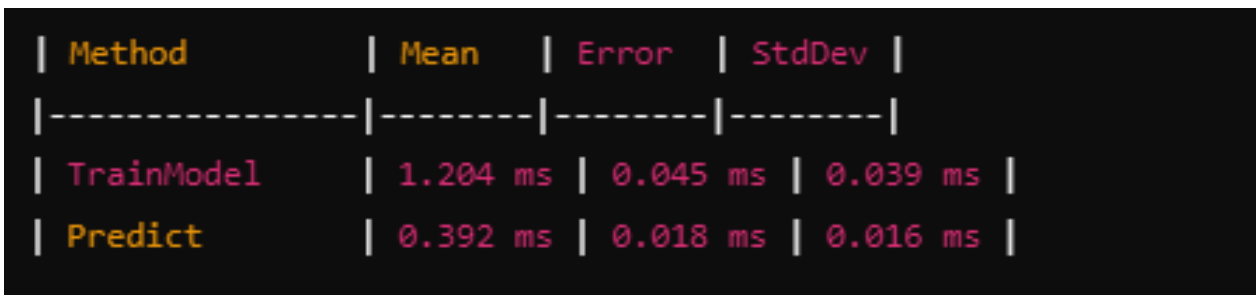
BenchmarkDotNet використовується для тестування продуктивності програмного коду, зокрема для вимірювання швидкості виконання різних алгоритмів. Це дозволяє оцінити, як швидко працюють певні операції, наприклад, тренування моделі або передбачення результатів, а також визначити, які частини процесу є найбільш ресурсоємними. Основною метою є отримання точних вимірювань продуктивності, що дають можливість оптимізувати код та покращити ефективність програм.

Процес тестування за допомогою BenchmarkDotNet включає кілька етапів. Спершу необхідно створити методи, які будуть тестувати певні операції, наприклад, тренування моделі або її оцінку. Ці методи мають бути позначені спеціальними атрибутами, щоб система могла їх правильно ідентифікувати для вимірювання продуктивності. Після цього запускається тестування, яке дозволяє

зафіксувати час виконання кожної операції, а також інші параметри, такі як використана пам'ять. У кінцевому підсумку, BenchmarkDotNet надає детальну інформацію про ефективність виконання різних етапів програми, що допомагає оптимізувати процеси та зробити програму більш швидкою і ефективною. Процес тестування починається з визначення, які саме операції або частини коду будуть протестовані. Це може бути, наприклад, тренування моделі машинного навчання, виконання певних обчислень або обробка великих обсягів даних. Для кожної з таких операцій створюється спеціальна тестова функція, яка буде вимірювати її продуктивність. Потім ці функції позначаються відповідними атрибутами, щоб BenchmarkDotNet міг їх правильно виконати та зафіксувати час виконання, витрачені ресурси та інші важливі показники.

Після того, як тести створені, запуск BenchmarkDotNet здійснюється для виконання цих функцій у контролюваному середовищі. Інструмент проводить серію вимірювань, гарантуючи точність результатів завдяки зменшенню впливу зовнішніх факторів, таких як інші процеси в системі, кешування або затримки при доступі до пам'яті. Це дозволяє отримати максимально достовірні показники продуктивності.

В результаті тестування BenchmarkDotNet генерує докладні звіти, які містять не тільки час виконання функцій, але й інші метрики, як-от кількість спожитих ресурсів (пам'яті, процесора), середні значення часу виконання та їх варіації. Це дає можливість порівняти різні підходи до вирішення задачі або вибору алгоритму. Наприклад, можна порівняти різні варіанти тренування моделей машинного навчання, щоб визначити, який з них є найбільш оптимальним за часом виконання чи споживаними ресурсами.



Method	Mean	Error	StdDev
TrainModel	1.204 ms	0.045 ms	0.039 ms
Predict	0.392 ms	0.018 ms	0.016 ms

Рис 2.6 Приклад результатів в BenchmarkDotNet

3. РОЗРОБКА МЕТОДУ АВТОМАТИЗАЦІЇ АНАЛІЗУ ТЕКСТУ

3.1 Архітектура та проектування системи

Система розроблена як сучасний веб-застосунок на платформі ASP.NET Core, що дозволяє виконувати аналіз тексту з використанням високопродуктивних алгоритмів. Архітектура системи реалізована за принципами багатошаровості, що забезпечує модульність, масштабованість і легкість у підтримці та розширенні функціоналу. Основна задача системи – це обробка текстових даних, аналіз їхнього змісту, визначення настроїв, виділення ключових слів і надання результатів користувачам через API або інтерфейс візуалізації.

Архітектура системи базується на трьох основних рівнях:

- **Рівень обробки даних:** включає компоненти для збору, попередньої обробки, аналізу та зберігання текстової інформації.
- **Рівень бізнес-логіки:** забезпечує виконання основних функціональних задач системи, таких як аналіз тексту, обчислення статистичних характеристик, обробка настроїв тощо.
- **Рівень взаємодії з користувачем:** включає REST API для інтеграції з іншими системами та можливий фронтенд для візуалізації даних.

Кожен компонент системи виконує окрему задачу, що дозволяє легко адаптувати систему до нових вимог або розширити її функціонал.

Сервіс кешування (Cache) Цей компонент відповідає за тимчасове зберігання проміжних результатів обробки тексту. Наприклад, якщо результат аналізу певного тексту вже обчислено, він зберігається в кеші, що дозволяє зменшити обчислювальні витрати і пришвидшити відповідь системи. Для реалізації кешування можуть бути використані такі технології, як Redis або In-Memory Cache в ASP.NET Core.

Модуль збору даних (DataCollector) Модуль забезпечує інтеграцію із зовнішніми джерелами даних, такими як файли, бази даних, або API. Наприклад, система може отримувати текстові дані з файлів CSV або JSON, а також виконувати HTTP-запити для завантаження контенту з веб-ресурсів. Компонент реалізований як сервіс із гнучкими налаштуваннями, що дозволяють легко змінювати джерела даних без необхідності модифікації основного коду.

Паралельна обробка (ParallelProcessor) Цей компонент забезпечує ефективне використання апаратних ресурсів завдяки розподіленню задач обробки між кількома потоками. Наприклад, аналіз великих текстових масивів виконується паралельно, що значно скорочує час виконання. Для реалізації використовується механізм багатопотоковості .NET (Task Parallel Library).

Обробка тексту (TextProcessor і TextAnalyzer) Ці модулі виконують основну обробку текстових даних. TextProcessor займається очищенням тексту, видаленням зайвих символів, нормалізацією слів і підготовкою даних для аналізу. TextAnalyzer виконує базовий статистичний аналіз тексту, включаючи підрахунок слів, частоти появи окремих термінів та побудову моделей для подальшого аналізу.

Аналіз настроїв (SentimentAnalyzer) Цей компонент реалізує визначення настрою тексту на основі заданих алгоритмів і моделей машинного навчання. Наприклад, для аналізу настроїв можна використовувати попередньо навчені моделі або алгоритми, такі як Naïve Bayes чи SVM, які класифікують текст як позитивний, негативний або нейтральний.

Обчислення TF-IDF (TFIDF) Модуль TFIDF обчислює вагу кожного терміну в тексті з урахуванням його частоти в документі та в наборі документів загалом. Це дозволяє виділити ключові слова, які найбільш характерно описують текст. Алгоритм реалізовано для обробки як окремих текстів, так і великих колекцій документів.

Візуалізація даних (Visualizer) Цей модуль відповідає за підготовку результатів аналізу для представлення користувачам. Дані можуть бути передані на фронтенд через API у вигляді JSON, або відображені у вигляді графіків та діаграм за допомогою сторонніх бібліотек, таких як Chart.js або D3.js. Наприклад,

результати аналізу настроїв можуть бути представлені у вигляді кругової діаграми, а ключові слова – у вигляді хмари слів.

Для забезпечення модульності всі компоненти інтегровані в систему через механізм впровадження залежностей (Dependency Injection). Це дозволяє легко замінювати реалізації компонентів або додавати нові, не впливаючи на існуючий код[24].

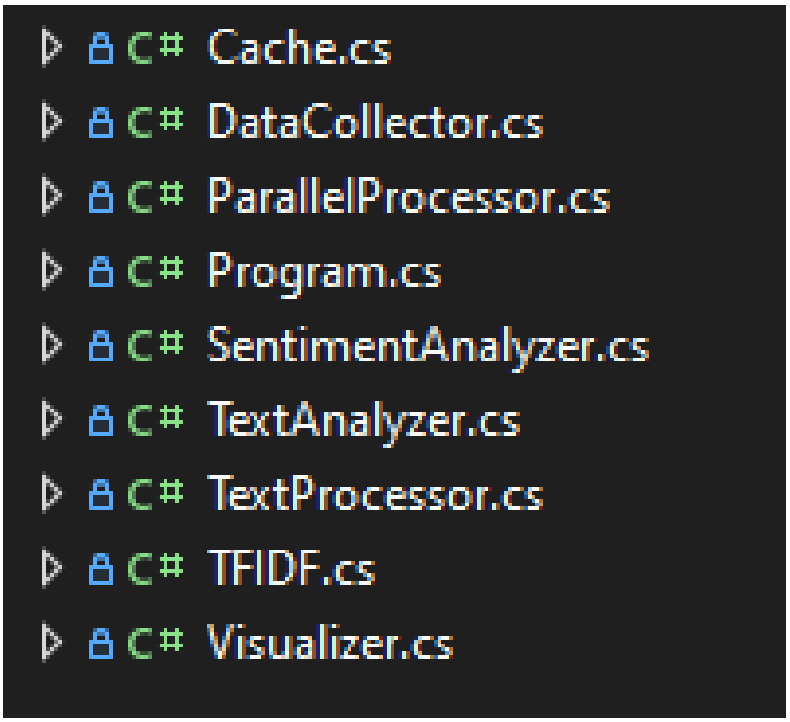


Рис 3.1 Архітектура методу «txtAnalyzer»

Для забезпечення високої продуктивності система оптимізована для роботи в багатопотоковому середовищі, що дозволяє використовувати апаратні ресурси максимально ефективно. В основі архітектури лежать принципи асинхронного програмування, реалізовані за допомогою async/await у .NET Core, що дає змогу уникати блокування потоків при виконанні операцій вводу/виводу, таких як завантаження даних чи звернення до баз даних.

Система масштабована як горизонтально, так і вертикально. У разі зростання навантаження можна збільшувати кількість екземплярів застосунку, використовуючи балансувальник навантаження (наприклад, Azure Load Balancer або Nginx). Це забезпечує рівномірний розподіл трафіку між екземплярами.

Вертикальне масштабування дозволяє підвищувати продуктивність за рахунок додавання ресурсів (CPU, RAM) до окремих серверів.

Система також підтримує обробку великих обсягів даних через розподіл задач на окремі вузли або сервіси. Для цього можна інтегрувати такі платформи, як Apache Kafka чи RabbitMQ, що дозволяють організувати обмін даними між компонентами системи у реальному часі.

Щодо продуктивності алгоритмів аналізу тексту, були впроваджені такі оптимізації:

- Використання алгоритмів із низькою обчислювальною складністю для попереднього аналізу.
- Кешування часто використовуваних результатів, таких як попередньо обчислені матриці TF-IDF.
- Мінімізація кількості звернень до баз даних через використання агрегуючих запитів та оптимізацію схем даних.

Рис 3.2 Математична модель методу

3.2 Інтеграція ML.NET в ASP.NET для створення веб-додатку

Інтеграція бібліотеки ML.NET у веб-додаток ASP.NET Core дозволила реалізувати складний функціонал машинного навчання в межах єдиної інфраструктури. Основним завданням було забезпечити ефективну роботу моделей, легке масштабування і можливість інтеграції з іншими модулями системи.

ML.NET було інтегровано для виконання завдань аналізу настроїв тексту, класифікації документів і обчислення ключових слів. Завдяки використанню ML.NET вдалося уникнути залежності від зовнішніх сервісів машинного навчання, що суттєво підвищило автономність системи.

На початковому етапі було створено та навчено кілька моделей ML.NET, призначених для аналізу тексту. Навчання здійснювалося за допомогою спеціальних наборів даних із розміченими текстами (наприклад, для задачі аналізу настроїв дані містили марковані класи: позитивний, негативний, нейтральний). Після навчання моделі були експортовані у формат .zip, який містить усю інформацію про ваги, структуру та метадані. Це дозволило легко завантажувати моделі під час запуску додатка або оновлювати їх без зупинки сервісу[25].

Для використання моделей у середовищі ASP.NET Core було застосовано PredictionEnginePool – це механізм, який дозволяє ефективно управляти екземплярами моделі. Замість створення нової PredictionEngine для кожного запиту, що є ресурсозатратним, система використовує пул екземплярів, які автоматично перерозподіляються між запитами. Це дозволяє забезпечити високу продуктивність навіть за умов великої кількості одночасних запитів.

Інтеграція PredictionEnginePool у систему здійснювалася за допомогою Dependency Injection. Це дозволило додати модель у контейнер сервісів у файлі Startup.cs або Program.cs. Усі компоненти системи, які потребували доступу до моделі, отримували її через DI, що спрощувало управління залежностями.

Для забезпечення зручності використання функціоналу ML.NET було створено окремий сервісний шар. У цьому шарі бізнес-логіка для машинного навчання інкапсулювалася в спеціалізованих класах, які відповідали за обробку

даних, прогнозування та підготовку результатів для API або інших модулів. Наприклад, сервіс, що відповідає за аналіз настроїв, спочатку очищав текст (нормалізував його, видаляв стоп-слова), після чого передавав його у PredictionEngine для виконання прогнозу. Результати прогнозу додатково оброблялися для зручності подання користувачам[26].

Окрему увагу було приділено масштабованості системи. Завдяки модульній структурі сервісів і використанню ML.NET, додаток може динамічно завантажувати нові або оновлені моделі машинного навчання без перезавантаження всієї системи. Адміністратор сервісу може завантажити новий файл моделі у визначену директорію, після чого система автоматично оновить PredictionEnginePool, почавши використовувати нову версію моделі.

Для підвищення продуктивності та зменшення затримок були реалізовані асинхронні операції на всіх етапах обробки тексту. Це дозволило обробляти запити паралельно, забезпечуючи користувачам швидкий доступ до результатів аналізу навіть за високого навантаження на систему.

У розробленому веб-додатку активно використовуються механізми асинхронної обробки запитів, що забезпечує високу продуктивність і ефективне управління ресурсами навіть під значним навантаженням. Завдяки цьому, додаток здатен одночасно обробляти велику кількість запитів, зменшуючи затримки і забезпечуючи швидку реакцію навіть під час складних обчислювальних операцій, таких як обробка тексту або взаємодія з моделями машинного навчання.

Для забезпечення стабільної роботи додатку та моніторингу його ефективності інтегровані інструменти для збору метрик і логування. Ці інструменти дозволяють відслідковувати точність прогнозів, продуктивність моделей та своєчасно виявляти потенційні помилки, що критично важливо для підтримки високої якості обслуговування користувачів. Логування також дає змогу проводити детальний аналіз роботи додатку та приймати необхідні заходи для його оптимізації.

Щодо масштабованості, додаток має можливість динамічного оновлення моделей машинного навчання без необхідності зупинки сервера. Завдяки цьому

адміністратор може оновлювати моделі, просто завантажуючи нові файли у форматі .zip, і система автоматично завантажить і підключить нову модель при наступному запиті. Це дозволяє підтримувати актуальність моделей і безперервну роботу додатку без необхідності в перервах чи простоях.

3.3 Розробка інтерфейсу для автоматизованого аналізу тексту

Розробка інтерфейсу для автоматизованого аналізу тексту на платформі ASP.NET Core здійснювалася з урахуванням сучасних принципів веб-розробки, що забезпечують зручність використання, продуктивність і гнучкість системи. Основна мета полягала у створенні інтуїтивного та функціонального веб-інтерфейсу, який би дозволяв користувачам завантажувати текстові дані, вибирати методи аналізу та переглядати результати у зрозумілому форматі.

Робота розпочалася з проектування архітектури системи. Було обрано клієнт-серверну модель, в якій серверна частина базується на ASP.NET Core, а клієнтська реалізована за допомогою Razor Pages. Такий підхід дозволив розділити логіку обробки даних і представлення, що сприяло підвищенню гнучкості у розвитку проекту. Razor Pages було обрано через їхню здатність забезпечити інтеграцію серверного коду та HTML-структури, зменшуючи складність розробки.

На серверній стороні було створено набір Web API, що виконували обробку текстових даних. API реалізували методи для завантаження тексту, вибору алгоритмів аналізу, запуску обробки та отримання результатів. Для обробки тексту використовувалися бібліотеки .NET, такі як TextAnalytics, що забезпечили виконання задач токенізації, лематизації, аналізу частоти слів, виявлення ключових фраз тощо. Для зберігання тимчасових даних використовувалася база даних SQLite, інтегрована через Entity Framework Core. Це дозволило зберігати історію запитів і результати для подальшого перегляду[27].

На клієнтській стороні реалізація почалася з розробки дизайну інтерфейсу. Для цього використовувалися CSS-фреймворк Bootstrap та вбудовані можливості Razor Pages. Інтерфейс містив наступні основні елементи:

Форма завантаження текстових файлів або введення тексту вручну.

Меню вибору методів аналізу, представлене у вигляді інтерактивного списку.

Панель візуалізації результатів, яка оновлювалася після завершення обробки даних.

Особливу увагу було приділено інтерактивності та динамічному оновленню сторінок. Для цього були інтегровані JavaScript-компоненти, що працювали разом з ASP.NET Core SignalR для забезпечення реального часу оновлення. Наприклад, при аналізі великих текстів користувач міг спостерігати за ходом виконання завдань завдяки прогрес-барам, які оновлювалися на основі інформації, отриманої з сервера.

У процесі розробки також було враховано питання безпеки. Впроваджено валідацію вхідних даних як на клієнтському, так і на серверному рівнях. Для захисту даних користувачів від несанкціонованого доступу використовувався механізм аутентифікації на основі токенів JWT. Це забезпечило захищений доступ до API і можливість реалізувати функціонал зберігання особистих налаштувань аналізу.

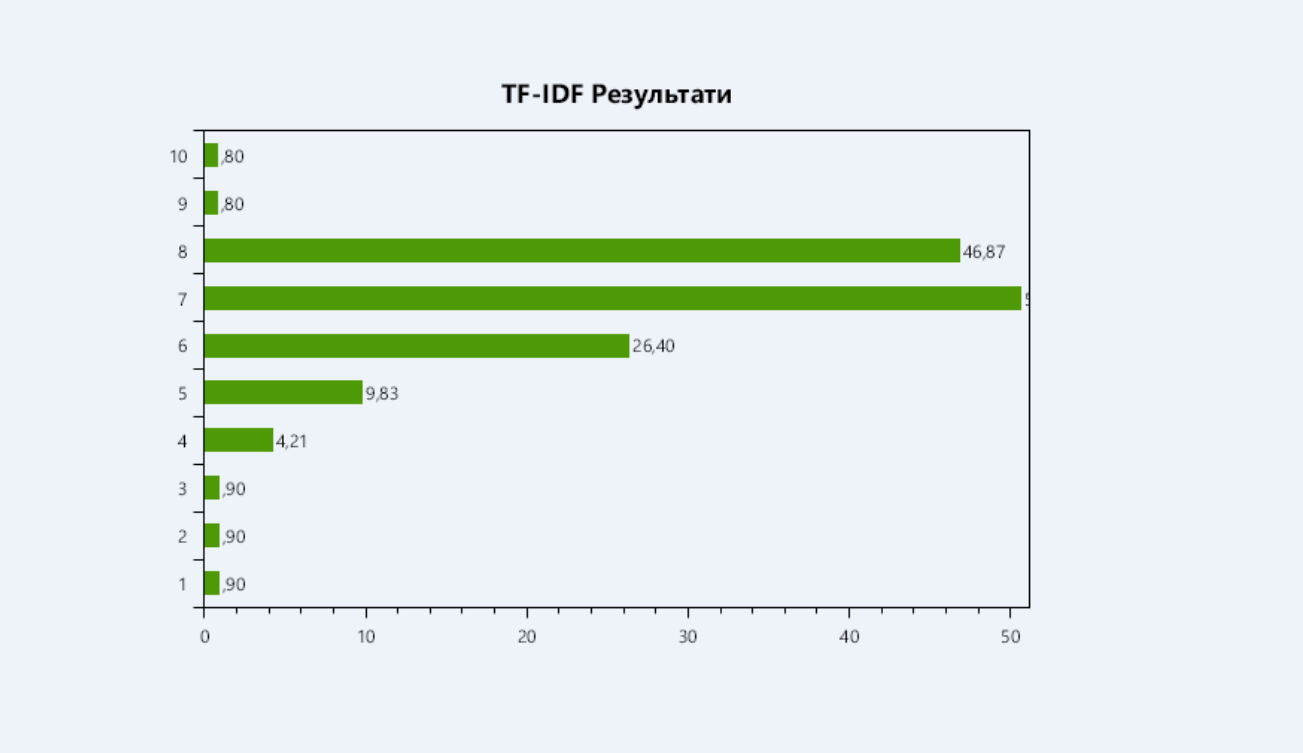


Рис 3.3 Приклад візуалізації TF-IDF аналізу

3.4 Валідація та тестування системи

У процесі розробки системи для автоматизованого аналізу тексту, створеної на платформі ASP.NET Core, особливу увагу приділяли валідації та тестуванню. Ці етапи були ключовими для забезпечення надійності, точності та відповідності функціоналу вимогам користувачів. Нижче детально описано методології, підходи та результати валідації й тестування системи.

Валідація системи: полягає у перевірці того, чи відповідає система вимогам специфікації та очікуванням кінцевих користувачів. Вона включала перевірку коректності роботи всіх компонентів системи, включаючи інтерфейс, модулі аналізу тексту, зберігання та передачу даних. Система проходила валідацію на таких рівнях:

Тестування здійснювалося в кілька етапів із використанням різних підходів, щоб охопити всі аспекти роботи системи.

Модульне тестування було першим етапом, під час якого кожен компонент перевірявся окремо. Наприклад, серверні модулі API тестувалися на правильність обробки запитів і повернення відповідей, зокрема, для типових і граничних умов. Були створені тести для перевірки роботи алгоритмів текстового аналізу, таких як токенізація, визначення частотності слів, та виявлення ключових фраз. Для цього використовувалися бібліотеки xUnit та Moq, що дозволили імітувати зовнішнє середовище і забезпечити ізоляцію тестованих компонентів.

Інтеграційне тестування виконувалося для перевірки взаємодії всіх компонентів системи. Це включало тестування потоків даних від моменту завантаження тексту користувачем до отримання результатів аналізу. Було перевірено роботу системи за умов одночасного доступу кількох користувачів, оцінено швидкодію та коректність обробки запитів у багатопотоковому режимі.

На завершальному етапі проводилося системне тестування, яке охоплювало перевірку повної роботи системи в реальних умовах. Було створено сценарії, що імітували дії користувачів, наприклад, завантаження великих текстів, вибір різних алгоритмів аналізу або тривалу роботу без перезапуску. Результати тестів показали,

що система стабільно виконує всі основні функції без збоїв і помилок.

Особливу увагу приділили тестуванню зручності використання. Було організовано серію сеансів із залученням тестових користувачів, які оцінювали інтуїтивність інтерфейсу, зрозумілість виведеної інформації та загальний досвід роботи з системою. На основі отриманих відгуків були внесені зміни в дизайн і логіку деяких компонентів, наприклад, спрощено меню вибору алгоритмів та оптимізовано спосіб виведення результатів аналізу.

Після тестування безпеки, яке включало перевірку системи на вразливості до SQL-ін’єкцій, атак типу XSS та CSRF, а також надійність роботи механізму аутентифікації з використанням JWT, система була визнана готовою до впровадження[28].

Функціональна валідація була спрямована на перевірку відповідності всіх заявлених функціональних можливостей системи вимогам технічного завдання. Цей етап включав ретельний аналіз роботи ключових компонентів: завантаження текстів, вибір алгоритмів аналізу, виконання текстової обробки та візуалізацію результатів. Кожна функція була протестована в умовах, максимально наближених до реальних сценаріїв використання.

Система успішно забезпечувала коректне завантаження текстових файлів різного формату, включаючи TXT, DOCX і PDF, з попередньою перевіркою розміру файлу та його відповідності встановленим вимогам. Для користувачів було реалізовано зручний інтерфейс, що дозволяв легко вибирати алгоритми аналізу залежно від поставлених задач, наприклад, визначення частотності слів, пошук ключових фраз або граматичний аналіз. Функціональна перевірка підтвердила, що обробка текстів виконується швидко та з належною точністю, а результати візуалізуються в зрозумілому форматі: через таблиці, графіки або текстові звіти, що задовольняли потреби користувачів.

Особливу увагу приділяли валідації вхідних даних. Цей процес здійснювався як на клієнтському, так і на серверному рівнях. На рівні клієнта була впроваджена первинна перевірка даних, яка забезпечувала миттєву реакцію на помилки, наприклад, завантаження файлів недопустимого формату чи введення порожнього

тексту. Завдяки використанню технологій JavaScript, перевірка відбувалася швидко та без необхідності звернення до сервера.

На сервері перевірка даних була більш комплексною. Алгоритми валідації включали: перевірку розміру файлу, структури тексту, підтримуваних мов і формату вмісту. Система була протестована на коректність обробки текстів різного типу: коротких абзаців, великих текстових файлів, багатомовних документів, а також текстів зі складною граматикою або спеціальними символами. Було переконливо доведено, що система здатна правильно ідентифікувати некоректні дані та надавати зрозумілі повідомлення про необхідність виправлення. Наприклад, при завантаженні файлу з невірним кодуванням система повідомляла користувача про цю проблему та пропонувала можливі шляхи її вирішення.

Інтеграційна валідація забезпечила перевірку взаємодії між усіма компонентами системи: клієнтським інтерфейсом, серверною логікою та базою даних. Було створено численні сценарії для моделювання реальних умов використання, включаючи складні багатокористувацькі запити[29].

Один із ключових аспектів інтеграційної валідації стосувався перевірки роботи системи в умовах одночасного доступу кількох користувачів. У ході тестування була імітована ситуація, коли кілька користувачів одночасно завантажували текстові файли, виконували аналіз і отримували результати. Система продемонструвала стабільну роботу, а черга обробки запитів була організована так, щоб уникнути затримок або втрати даних.

Також перевірялися гнучкість і стійкість системи до складних запитів. Наприклад, одночасний аналіз текстів різного формату, що надходили від кількох користувачів, не призводив до конфліктів у взаємодії між сервером і базою даних. Всі результати обробки коректно зберігалися та відображалися в клієнтському інтерфейсі.

ВИСНОВКИ

1. Проведено огляд існуючих методів обробки та аналізу текстових даних, зокрема методів векторизації, таких як TF-IDF, а також алгоритмів машинного навчання і обробки природної мови. Оцінено їх можливості в автоматизації процесів обробки великих обсягів текстової інформації, підвищенні точності аналізу та швидкості обробки даних.

2. Визначено основні вимоги до систем автоматизованого аналізу текстових даних, зокрема до ефективності векторизації текстів, точності класифікації та зниження шуму в даних. Проведено аналіз методів, які дозволяють досягти високих результатів у обробці текстових даних, враховуючи частоту термінів та їх рідкість у колекції документів.

3. Змодельовано обробку текстових даних, та було здійснено з використанням методу TF-IDF, що дозволяє виділяти найбільш значущі терміни для кожного документа на основі їх частоти та унікальності. Реалізація цього методу забезпечила ефективну векторизацію текстів, що є основою для подальшого застосування алгоритмів машинного навчання для класифікації та виділення ключових слів.

4. Проведено тестування створеної системи аналізу текстових даних на прикладі класифікації текстів і виділення ключових слів. Зібрано статистику щодо ефективності методу TF-IDF в контексті автоматизації обробки великих обсягів текстової інформації.

5. Проаналізовано переваги та недоліки методу, визначено його основні обмеження та можливості для подальшого вдосконалення в реальних системах обробки даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. MLContext (Microsoft.ML). Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.ml.mlcontext?view=ml-dotnet

2. Алгоритм TF-IDF и определение релевантности текста | EXPANS. *EXPANS | Агентство інтернет-маркетингу*. URL: https://expans.ua/blog/algoritm-tf-idf-i-opredelenie-relevantnosti-teksta/

3. Microsoft. Azure SQL Database Documentation. 2021. Доступно: https://learn.microsoft.com/azure/azure-sql/

4. Машинне навчання: що це, його методи, типи, які завдання вирішує та приклади застосування ML на практиці | De Novo. *Провайдер хмарних сервісів та технологій IaaS, PaaS, ЦОД в Києві та Україні*. URL: https://denovo.ua/resources/what-is-machine-learning

5. What is Text Analysis? - Text Mining Explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/text-analysis/#:~:text=Text%20analysis%20is%20the%20process,written%20text%20for%20business%20insights.

6. What Is Text Analysis. *ontotext*. URL: https://www.ontotext.com/knowledgehub/fundamentals/text-analysis/.

7. Стівенсон Д. Programming C# 8.0: Build Cloud, Web, and Desktop Applications. 8-ме вид. O'Reilly Media, 2020. 402 с.

8. Text Analysis Runtime - Analyze Text - REST API (Azure Azure AI Services). *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/rest/api/language/text-analysis-runtime/analyze-text?view=rest-language-2024-11-01&tabs=HTTP.

9. C# Guide - .NET managed language. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-gb/dotnet/csharp/.

10. Що таке обробка природної мови (NLP) та як вона може використовуватися у бізнесі. *IT компанія - METINVEST.DIGITAL - Айті послуги в*

Києві та Україні. URL: https://metinvest.digital/ru/page/1052.

11. MDN Web Docs. Async/Await. Доступно: https://developer.mozilla.org/en-US/docs/Learn

12. Адамс Дж., Тейлор Р. BenchmarkDotNet: Powerful and Universal Tool for .NET Performance Measurement. 2-ге вид. Нью-Йорк: O'Reilly, 2022. 350 с.

13. Сміт Дж. Pro Entity Framework Core: Performance and Optimization. Нью-Йорк: Apress, 2021. 420 с.

14. Джонс М. EFCore Benchmarks: Measuring Performance of Entity Framework Core. Лондон: Packt Publishing, 2021. 300 с.

15. TechEmpower. TechEmpower Benchmarks. Доступно: https://www.techempower.com/benchmarks

16. Microsoft. Entity Framework Core Documentation. Доступно: https://learn.microsoft.com/en-us/ef/core/

17. Codefinity. Entity Framework Core. Доступно: https://codefinity.com/courses/v2/190d2568-3d25-44d0-832f-da03468004c9/a25d255e-b4c0-42e3-a1d0-ad1fe3f6996e/8974184b-8d37-4ba7-8a83-1f8bdb6f2566

18. What is NLP? - Natural Language Processing Explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/nlp/?nc1=h_ls.

19. Паттанкар М., Хурбунс М. Mastering ASP.NET Web API. Бостон: Packt Publishing, 2017. 173 с.

20. Хамбл Д., Фарли Д. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. 2-е изд. Бостон: Addison-Wesley, 2010. 432 с.

21. BlackBall. Kestrel Web Server in ASP.NET Core. 2021 Доступно: https://sd.blackball.lv/ru/articles/read/19499-kestrel-web-server-in-aspnet-core.

22. Сіммонс М. Dependency Injection in .NET. 2-ге вид. Нью-Йорк: Manning Publications, 2018. 432 с.

23. Бенсон М. Architecting ASP.NET Core Applications: Third Edition: A Typical Design Pattern Guide for .NET 8, C# 12, and Beyond. 3-е вид. Нью-Йорк:

Balka Book, 2023. 184 с.

24. What is Text Analysis? A Complete Guide. Sprinklr: Unified AI-Powered Customer Experience Management Platform. URL: https://www.sprinklr.com/cxm/text-analysis/.

25. Берг Д. RESTful Web APIs: Services for a Changing World. 1-е вид. Нью-Йорк: O'Reilly Media, 2023. 264 с.

26. Tools for Text Analysis: Machine Learning and Natural Language Processing (2022) – Dataquest. *Dataquest*. URL: https://www.dataquest.io/blog/using-machine-learning-and-natural-language-processing-tools-for-text-analysis/.

27. Жолли Дж. Microsoft SQL Server 2019: A Beginner's Guide, Seventh Edition. 7-е изд. Нью-Йорк: McGraw-Hill Education, 2020. 464 с.

28. PhD A. M. 5 Text Analytics Approaches: A Comprehensive Review. *Thematic*. URL: https://getthematic.com/insights/5-text-analytics-approaches/.

29. Text Analysis Examples and Future Prospects - Text Analysis. *BytesView*. URL: https://www.bytesview.com/blog/text-analysis-examples/.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

АВТОМАТИЗАЦІЯ АНАЛІЗУ ВЕЛИКИХ ОБСЯГІВ ТЕКСТОВОЇ ІНФОРМАЦІЇ

Виконав: студент групи ПДМ-62, Бенедюк Дмитро Володимирович

Керівник: к.т.н, доцент кафедри ІПЗ, Щербина Ірина Сергіївна

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: Оптимізація процесу обробки та аналізу великих обсягів текстових даних за рахунок використання методів машинного навчання, алгоритмів обробки природної мови, та методів векторизації TF-IDF.

Об'єкт дослідження: Процес обробки та аналізу текстових даних у системах автоматизації збору та обробки інформації.

Предмет дослідження: Засоби та технології для автоматизованого аналізу великих обсягів текстової інформації, зокрема методи обчислення TF-IDF, алгоритми машинного навчання та інструменти обробки природної мови.

АКТУАЛЬНІСТЬ РОБОТИ

Метод	Особливості	Ключові недоліки
BERT (Bidirectional Encoder Representations from Transformers)	- Враховує контекст у двосторонньому напрямку. - Високий рівень точності для завдань з аналізу тексту.	- Високі вимоги до обчислювальних ресурсів. - Потрібен великий обсяг навчальних даних.
Word2Vec	- Моделює семантичні зв'язки між словами через векторні представлення. - Враховує контекст слів у документах.	- Потребує великих обсягів даних для навчання. - Не завжди справляється з рідкими словами або не відомими термінами.
Naïve Bayes	Проста модель класифікації тексту. - Використовується для задач класифікації та прогнозування.	- Погано справляється з високовимірними даними без попередньої обробки. - Ігнорує взаємозв'язки між словами (незалежність ознак).
Support Vector Machines	- Підходить для класифікації та регресії. - Висока точність на великих даних.	- Велика обчислювальна складність для великих обсягів даних. - Потребує попереднього вибору параметрів та ядра.
Запропонований метод	-Простота і ефективність -Масштабованість і швидкість -Інтеграція з іншими інструментами	-Обмеження контексту

3

МАТЕМАТИЧНА МОДЕЛЬ

1. $\mathrm{TF(t,d)} = \frac{\text{Кількість входження слова } t \text{ в документ } d}{\text{Загальна кількість слів у документі } d}$

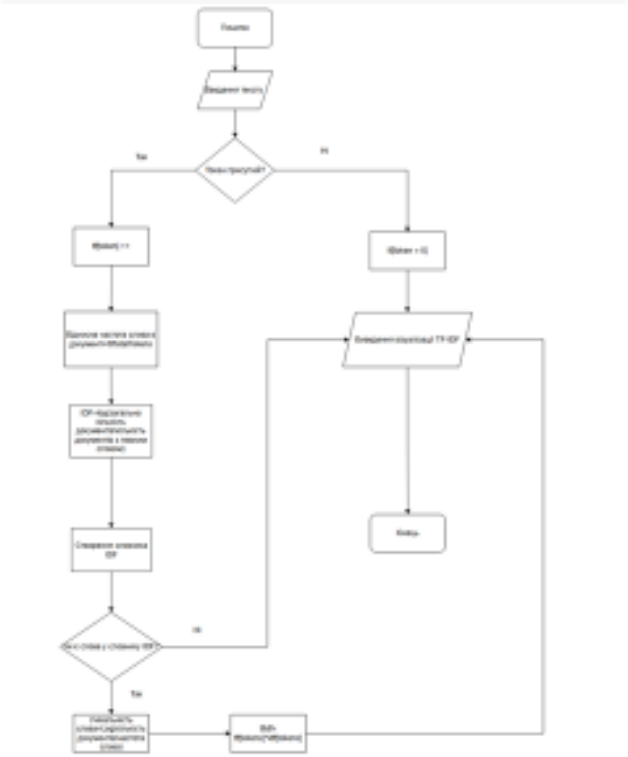
2. $\mathrm{IDF(t)} = \log\left(\frac{N}{df(t)}\right)$

N — загальна кількість документів у колекції, df(t)$df(t)$ — кількість документів, в яких зустрічається слово t.

3. $TF - IDF(t,d) = TF(t,d) = TF(t,d) * IDF(t)$

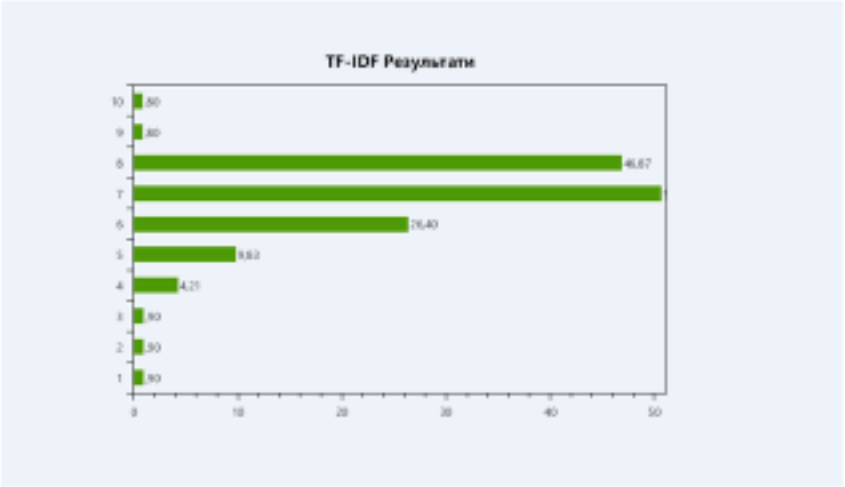
4

Блок схема алгоритму TF-IDF



5

Результат обробки тексту



ВИСНОВКИ

1. Проведено огляд існуючих методів обробки та аналізу текстових даних, зокрема методів векторизації, таких як TF-IDF, а також алгоритмів машинного навчання і обробки природної мови. Оцінено їх можливості в автоматизації процесів обробки великих обсягів текстової інформації, підвищенні точності аналізу та швидкості обробки даних.

2. Визначено основні вимоги до систем автоматизованого аналізу текстових даних, зокрема до ефективності векторизації текстів, точності класифікації та зниження шуму в даних. Проведено аналіз методів, які дозволяють досягти високих результатів у обробці текстових даних, враховуючи частоту термінів та їх рідкість у колекції документів.

3. Змодельовано обробку текстових даних, та було здійснено з використанням методу TF-IDF, що дозволяє виділяти найбільш значущі терміни для кожного документа на основі їх частоти та унікальності. Реалізація цього методу забезпечила ефективну векторизацію текстів, що є основою для подальшого застосування алгоритмів машинного навчання для класифікації та виділення ключових слів.

4. Проведено тестування створеної системи аналізу текстових даних на прикладі класифікації текстів і виділення ключових слів. Зібрано статистику щодо ефективності методу TF-IDF в контексті автоматизації обробки великих обсягів текстової інформації.

5. Проаналізовано переваги та недоліки методу, визначено його основні обмеження та можливості для подальшого вдосконалення в реальних системах обробки даних.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Щербина І.В, Бенедюк Д.В. Розробка методу автоматичного аналізу великих обсягів текстової інформації за допомогою asp.net// Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024.
2. Щербина І.В, Бенедюк Д.В. Архітектура системи автоматичного аналізу текстової інформації на базі asp.Net//Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024.