Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Массалітіна Дмитра Дмитровича*
(ПІБ)

академічної групи *122М-19-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

на тему: *Інформаційна технологія для класифікації наукових текстів на основі методу модифікованої логістичної регресії*

_____ *Д.Д. Массалітін*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	Проф. Мещеряков Л.І.			
економічний	Доц. Касьяненко Л.В.			

Рецензент				

Нормоконтролер	Доц. Сироткіна О.І.			

Дніпро
2020

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

І.М. Удовик
(підпис) (прізвище, ініціали)

« » 20 20 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122М-19-1* *Массалітіну Дмитру Дмитровичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Інформаційна технологія для класифікації наукових текстів на основі методу модифікованої логістичної регресії*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 22.10.2020 р. № 888-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес класифікації наукових текстів та практичне використання технологій обробки природної мови в освітніх додатках, з метою підвищення ефективності освітнього процесу.

Предмет досліджень – методи, моделі машинного навчання та обробки природньої мови у задачах класифікації наукових текстів.

Мета роботи – вдосконалення та пришвидшення процесу класифікації текстів за допомогою моделі логістичної регресії, з метою застосування її у освітніх додатках для покращення освітнього процесу.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна полягає у тому, що вдосконалено та розширено можливості методу логістичної регресії на основі комбінування його з методом ранжування, що в результаті дозволило використати метод логістичної регресії для навчального асистента.

Практична цінність полягає у тому, що в результаті роботи, було створено прототип навчального асистента, що використовує модель логістичної регресії для

класифікації текстів. Використані методи та підходи у прототипі можуть застосовуватись як при розробці «інтелектуальних» навчальних систем, так й в практиці викладання дисциплін, пов’язаних з обробкою природної мови.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми, постановка задачі та збір інформації	15.09.2020-30.09.2020
Дослідження існуючих підходів до обробки мовної інформації, пошук оптимального методу класифікації наукових текстів для навчального асистента.	01.10.2020-05.11.2020
Створення навчального асистента на основі логістичної регресії, скомбінованої з методом ранжування, для підвищення ефективності освітнього процесу	06.11.2020-06.12.2020

Завдання видав _____ *Мещеряков Л.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Массалітін Д.Д.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *15.09.2020*

Термін подання кваліфікаційної роботи до ЕК *10.12.2020*

РЕФЕРАТ

Пояснювальна записка: 103 стор., 31 рис., 2 таблиці, 4 додатка, 70 джерел.

Об’єкт дослідження: процес класифікації наукових текстів та практичне використання технологій обробки природної мови в освітніх додатках, з метою підвищення ефективності освітнього процесу.

Предмет дослідження: методи, моделі машинного навчання та обробки природньої мови у задачах класифікації наукових текстів.

Мета магістерської роботи: вдосконалення та пришвидшення процесу класифікації текстів з допомогою моделі логістичної регресії, з метою застосування її у освітніх додатках для покращення освітнього процесу.

Методи дослідження. Для створення рекомендаційного та навчального асистента були використані засоби та методи машинного навчання, теорії множин, лінейної алгебри й обробки природної мови.

Наукова новизна полягає у тому, що вдосконалено та розширено можливості методу логістичної регресії на основі комбінування його з методом ранжування, що в результаті дозволило використати метод логістичної регресії для навчального асистенті.

Практична цінність полягає у тому, що в результаті роботи, було створено прототип навчального асистента, що використовує скомбіновану з методом ранжування модель логістичної регресії для класифікації текстів. Використані методи та підходи у прототипі можуть застосовуватись як при розробці «інтелектуальних» навчальних систем, так й в практиці викладання дисциплін, пов’язаних з обробкою природної мови.

У розділі «Економіка» проведено розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПО, тривалості його розробки й результати маркетингового дослідження.

Список ключових слів: класифікація текстів, логістична регресія, машинне навчання, обробка природної мови, NLP, SVM, word2vec, парсинг, Python, PostgreSQL, scikit-learn, word embedding.

ABSTRACT

Explanatory note: 103 pages, 31 figures, 2 tables, 4 applications, 70 sources.

Object of research: the process of classification of scientific texts and practical employment of natural language processing technologies in educational applications with the aim of improving educational process.

Subject of research: methods, machine-learning models for natural language processing of scientific text classification problems.

Purpose of Master's thesis: improvement and acceleration of texts' classification with a model for educational applications with a view to increasing the efficiency of the educational process.

Research methods. The tools and methods of machine learning, set theory, linear algebra, and natural language processing were employed so as to create an intelligent assistance bot.

Originality of research is the improved and extended capabilities of logistic regression with combination of it with a ranking method, with a ranking method, which enabled its application in the intelligent assistance bot.

Practical value of the results is that we have created a prototype of an educational assistant, which uses a model of logistic regression combined with a ranking method for texts' classification. Applied measures and techniques can be applied in development other intelligent educational systems and in the practice of teaching NLP-related disciplines.

In the Economics section we have calculated the complexity of software development, its cost, duration and demonstrated marketing research results.

Keywords: text classification, logistic regression, machine learning, natural language processing, NLP, SVM, word2vec, parsing, Python, PostgreSQL, scikit-learn, word embedding.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

NLP – Natural Language Processing (обробка природної мови);

AI – Artificial intelligence (штучний інтелект) ;

UML – Unified Modeling Language;

SVM – support vector machine (метод опорних векторів);

RNN – Recurrent neural network (рекурентна нейронна мережа);

LSTM – long short-term memory (долгая краткосрочная память);

GRU – Gated recurrent units (вентильний рекурентний вузол);

ML – Machine learning (машинне навчання);

TF – term frequency;

IDF – inverse document frequency.

ВСТУП

Актуальність теми. Тема класифікації текстів й інформаційних джерел надзвичайно важлива сьогодні, так як ми живемо в інформаційну епоху, для якої є характерним інформаційний простір, переповнений інформацією різного типу та якості, від художніх творів й новин до наукових праць. Хоча вільний доступ до широкого різноманіття публікацій є безсуперечною перевагою, разом з цим ця обставина кидає свої виклики людині.

Людині інтелектуальної праці, студенту чи науковцю у всі часи було надзвичайно важливо постійно оновлювати свої знання, пізнавати й вивчати нове, на жаль, у віруючому інформаційному потоці буває важко сконцентруватись на одній темі й досконально вивчити її. Тож технології класифікації текстів набувають все більшої актуальності й особливо очевидна їх важливість у часи пандемії, коли навчання та робота в основному перейшли у дистанційний формат. Як окремим людям, так бізнесу й науковим установам важливо точно й ефективно класифікувати й пріоритизувати інформацію, що надходить, у цьому їм допомагають інструменти машинного навчання та статистики.

Задача класифікації текстів відноситься до наукової області NLP (Natural Language Processing). NLP – це міждисциплінарний напрям, що містить в собі елементи лінгвістики, комп'ютерних наук та AI, цей напрям займається проблемою взаємодії між комп'ютером та людською мовою, зокрема процесами обробки та аналізу великих об'ємів мовної інформації. До кола задач, що вирішує NLP входить розпізнавання мовлення, розуміння людської мови, генерування текстів та ведення діалогу.

Перші праці, які можна займались вивченням проблеми аналізу та генерації людської мови й думок можна віднести до пізнього середньовіччя, наприклад Лейбніц намагався створити механічну машину для генерації думок. Проте перші ґрунтовні дослідження у області NLP почали проводитись у середині 1950-х років у Джорджтаунському університеті сумісно з IBM.

Основним завданням досліджень цього періоду було переклад з російською на англійську, їх результатом була перша система автоматичного перекладу, що базувалася на лінгвістичних правилах. Хоча результати Джорджтаунського експерименту були багатообіцяючі, цей підхід виявився обмеженим, тож науковці шукали нові методи, які концентрувалися вже на розумінні тексту. З ростом обчислювальної потужності комп'ютерів дослідники поступово переходили до статистичних моделей опрацювання мови. Нарешті, сьогодні завдяки хмарним обчисленням розробники можуть проектувати нейронні мережі для обробки мови, що самі аналізують семантику тексту й визначають взаємозв'язками між словами та реченнями.

Нейронні мережі - це найбільше досягнення області машинного навчання сьогодні. Однак, ця технологія все ще має свої недоліки, зокрема, мережі вимагають великих обчислювальних ресурсів та об'ємів даних для навчання. Хоча їх використання виправдано у випадку, коли потрібно створити систему, що може вдало симулювати співбесідника, для задачі класифікації текстів можуть використовуватись простіші методи.

Дана робота ставить за мету наглядно продемонструвати на прикладі навчального бота-асистента, що навіть давно відомі методи класифікації тексту, зокрема логістична регресія, сьогодні актуальні й можуть вирішувати практичні завдання.

Мета дослідження полягає **у** вдосконаленні та пришвидшенні процесу класифікації текстів з допомогою моделі логістичної регресії, з метою застосування її у освітніх додатках для покращення освітнього процесу.

Завдання дослідження. Для виконання поставленої задачі у ході роботи були сформульовані й вирішені наступні завдання:

1. Описати принцип роботи й будову моделей для класифікації текстів.
2. Проаналізувати й порівняти методи обробки природної мови й представлення текстової інформації для задач класифікації.
3. Проаналізувати й порівняти моделі класифікації текстів.

4. Дослідити особливості комбінування методів представлення текстової інформації й моделей-класифікаторів.

5. Обґрунтувати використання логістичної регресії для задач класифікації текстів.

6. Створити навчального асистента, що базується на моделі логістичної регресії для класифікації скомбінованої з методом ранжування.

Об’єкт дослідження: процес класифікації наукових текстів та практичне використання технологій обробки природної мови в освітніх додатках, з метою підвищення ефективності освітнього процесу.

Предмет дослідження: методи, моделі машинного навчання та обробки природньої мови у задачах класифікації наукових текстів.

Методи дослідження. Для створення рекомендаційного та навчального асистента були використані засоби та методи машинного навчання, теорії множин й обробки природної мови.

Наукова новизна полягає у тому, що вдосконалено та розширено можливості методу логістичної регресії на основі комбінування його з методом ранжування, що в результаті дозволило використати метод логістичної регресії для бота-асистента.

Практична значення. В результаті проведеної роботи, було створено прототип навчального асистента, що використовує модель логістичної регресії для класифікації текстів. Використані методи та підходи у прототипі можуть застосовуватись як при розробці «інтелектуальних» навчальних систем, так й в практиці викладання дисциплін, пов’язаних з обробкою природної мови.

Особистий внесок автора:

1. Описані принципи роботи й побудови моделей для класифікації текстів.

2. Було проаналізовано та порівняно методи представлення текстової інформації та моделі класифікації текстів.

3. Дослідити особливості комбінування методів представлення текстової інформації та моделей-класифікаторів.

4. Було створено навчального асистента, що демонструє успішне використання логістичної регресії для задач класифікації.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить 103 сторінки, в тому числі 61 сторінку тексту основної частини з 32 рисунками та 2 таблицями, список використаних джерел із 70 найменувань на 7 сторінках, 3 додатка на 16 сторінках.

РОЗДІЛ 1
ПРОБЛЕМИ ТА ЗАВДАННЯ ОБРОБКИ ПРИРОДНЬОЇ МОВИ

1.1. Загальний огляд області машинної обробки природної мови

Обробка природної мови (NLP) – це широкий напрям науки про штучний інтелект (AI), що виник на початку 20 століття. Першим й основним завдання цього напряму був автоматичний переклад мови, з часом його основні задачі й методи поступово змінювались та розширювались. На разі, до кола задач NLP окрім перекладу також входять проблеми розпізнавання та розуміння мови людини, класифікації й генерування текстів.

Однак, навіть у середньовіччі були перші праці, які присвячувались проблемам обробки людської мови й розумінню її властивостей. Наприклад, першою науковою працею, що стосувалася проблем області обробки людської мови, можна вважати працю 1666 року «Про витвір комбінаторики» Готфріда Вільгельм Лейбніца. У цій праці Лейбніц описав теорію автоматичного виробництва знань на основі комбінації символів, що створювались за визначеними правилами.

Лейбніц вважав, що людські думки, незалежно від їх складності, підпорядковуються деякому набору правил й являються комбінаціями фундаментальних концепцій, приблизно як речення являються комбінацією слів, а слова - комбінацією букв. На ці роздуми його надихнула машина середньовічного містика Раймунда Луллія, що являла собою круговий паперовий механізм з концентричними кільцями, на яких були записані символи, повертаючи ці кільця, можна було отримати різні слова. На відміну від Лулія, Лейбніц прагнув створити свою машину не для теологічних дебатів, а для філософський й наукових цілей.

Представлення про механізм, який може видавати розумні думки, відповідало духу часу, в якому жив Лейбніц. Він так й не зміг побудувати

машину для генерації висловів, хоча ця ідея дозволила йому створити у 1673р. перший механічний калькулятор на базі теорій «покрокового обчислення».

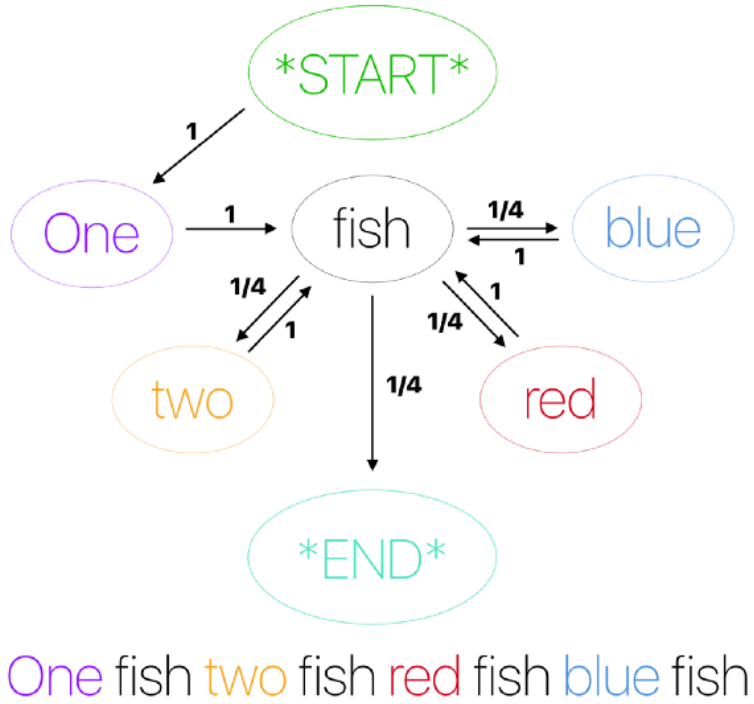

Рис. 1.1. Робота ланцюга Маркова на прикладі англійського вислову

Пізніше, у 1913р. російський математик А.А. Марков застосував методи теорію вірогідності для аналізу художніх текстів, він намагався виявити їх статистичні властивості. Його ідея полягала у тому, що мова – це система, у якій кожне наступне слово залежить від попередніх, тож він вважав, що можна побудувати ймовірнісну модель мови. Результатом цих досліджень стала модель «ланцюги Маркова», яка дозволяла генерувати вислови. «ланцюги Маркова» до сих пір використовуються у ролі простих генераторів фраз й у інших ймовірнісних задачах.

Клод Шеннон оприлюднив «Математичну теорію зв'язків» у 1948 році. У першому контрольованому експерименті він розпочав генерацію речення, випадково обираючи літери з 26 латинських и одного пробілу. Отримав незрозумілу комбінацію: «OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL».

У подальших дослідах виявилось, що розвинута й розширена статистична модель дозволяє отримати більш зрозуміле повідомлення. Шеннон створив статистичну платформу англійської мови, чим довів можливість генерації мови.

1.1.1. Обробка природньої мови у сучасності

Область NLP як науковий напрям бере свій початок у 40-х роках 20 століття й до сих пір має важливе значення для науки про штучний інтелект. Праці в області AI не тільки прагнуть втілити здібності та властивості людини у комп'ютері, але також ставлять за мету краще та детальніше розуміння процесів мислення й прийняття рішень людиною. Оскільки людина обмінюється, зберігає дані та синтезує інформацію за допомогою мови, то тема обробки людської мови комп'ютером постійно буде в фокусі досліджень штучного інтелекту.

На перший погляд, проблема перекладу може здатись незначною й нескладною, але саме задача автоматичного перекладу була довгий час основним рушієм розвитку технологій NLP й AI в цілому. Проблема перекладу як й у випадку з живим перекладачем полягає у вірному розумінні контексту, адже неможливо зробити правильний переклад з однієї мови на іншу (особливо якщо текст має багатозначні слова), не маючи загального уявлення про тему. Етап розуміння передбачає, що перекладач може потім переказати оригінальний текст іншими словами. Якщо машина може перекладати тексти на рівні людини без додаткового навчання під конкретні теми, то у цьому випадку її можна вважати сильним штучним інтелектом, так як якісний переклад потребує загального уявлення про реальність та, власне, розуміння людської мови. Отже, можна зробити висновок, що автоматичний переклад є одним з головних показників розвитку в області обробки природніх мов та штучного інтелекту.

Разом з тим потрібно відмітити, що NLP сьогодні не обмежується лише перекладом. Окрім перекладу, технології NLP знаходять застосування й у інших завданнях, зокрема, у виявленні зв'язку між сутностями у тексті (наприклад, аналіз запиту людини, збір даних з тексту), аналізу тональності повідомлення

(аналіз коментарів та відгуків), кластеризації (формування добірок статей та новин залежно від вподобань користувача) й генерації контенту (сьогодні частина новин текстів Bloomberg News генерується за допомогою системи Cyborg).

Загалом сучасну історію NLP та машинного перекладу можна розділити на 4 етапи, які характеризуються різним підходами та формулюваннями завдань. Початковим етапом NLP прийнято вважати період між початком 1940-х років та до кінця 1960-х років, другий до кінця 1970-х й третій до 80-х, четверта фаза це сучасність. Коротко, ці 4 фази можна охарактеризувати так: перша фаза вирішувало завдання та проблеми машинного перекладу за допомогою семантики, друга фаза вже почала звертатись до AI, третя фокусувалася на мовних граматико-логічних конструкціях, четверта концентрувалася на зборі та опрацюванні лексичних даних й корпусу текстів [28].

Остання фаза ілюструє, що розвиток NLP йшов дещо циклічно, так як він знову повернувся до завдань першого періоду, а саме до складання всеосяжних словників, які характерні для систем машинного перекладу. Окрім цього, потрібно відмітити, що сучасні дослідники також повернулись до ідеї збору лінгвістичної інформації з корпусу текстів. Однак, така циклічність у розвитку також демонструє ступінь прогресу, який був зроблений у обчислювальних можливостях ЕВМ та таких областях як парсинг та обчислювальна граматика, тобто сучасні дослідники повернулись до старих положень та ідей зі значно ширшою науковою базою, що дозволяє переосмислити й покращити старі ідеї.

1.1.2. Підхід до обробки природньої мови з позицій формальної логіки

Початкова фаза розвитку NLP концентрувалася на проблемах машинного перекладу. Для початкової фази розвитку NLP був характерним підхід з позицій формальної логіки, вважалось, що мова - це знакова система, що підпорядковуються ряду формальних правил. Першим досягненням цього періоду був перший вдалий машинний переклад тексту розміром 60 речень з

російської на англійську, він був продемонстрований у дуже обмеженій та рудиментарній формі у 1954 році під час сумісного експерименту IBM та Джорджтаунсього університету [28].

Іншою знаковою точкою першого етапу розвитку технік NLP була Теддінгтонська міжнародна конференція з машинного перекладу 1961 року, вона підсумовувала масштабну роботу багатьох державах щодо формулювання та систематизування морфології, синтаксису та семантики мов, що дозволило істотно покращити автоматичні перекладачі. Фактично, у цей час праці у області NLP були тісно пов'язані з лінгвістикою, так як потрібно було закласти базис, формалізувати мовні правила, конструкції та закономірності, щоб комп'ютер міг опрацьовувати мову за допомогою відповідного логічно-математичного апарату.

Основний напрям роботи цього періоду можна охарактеризувати як спробу розробити автоматизовану систему, що на основі словника дослівно перекладає тексти. Хоча дослідникам вдалось досягти значних успіхів у автоматичному переклади для того часу, цей спосіб добре працював лише для окремих випадків, його було трудно застосовувати до великих й складних текстів. Як результат, з'явилася потреба у вирішенні синтаксичної та семантичної неоднозначності при перекладі, рішенням цієї неоднозначності стали методи перекладу, що базуються на контексті, тобто семантиці (смислу) тексту.

Семантичне рішення комбінувало як правила, що включали в себе окремі слова так семантичні категорії разом зі словосполученнями. Однак, у процесі вирішення виникла інша проблема, у таких мовах як німецька виникали довгі та нечіткі залежності, що побудило розробників до розробки перших парсерів речень й граматики.

Більшість NLP дослідів концентрувались на синтаксису природніх мов, частково це було спричинено тим, що розробникам було важливо опрацьовувати синтаксис для отримання інформації з речення, частково тому що вважалось, що саме підхід базований на синтаксисі речень буде ефективним. Хоча дослідники теж приділяли увагу семантичним методам, перевага надавалась синтаксичному способу, так як можливі семантичні двозначності вирішували за допомогою

використання слів з множинними значеннями при перекладі. Найважливішими результатами від цих дослідів була розробка перших базових принципів парсингу, обробки синтаксичних конструкцій й отримання інформації з мови, що доступна для обчислювальних методів.

У підсумку, через обмеженість обчислювальних ресурсів й нестачу досвіду у області лінгвістики та NLP дослідникам не вийшло досягти поставленої мети й розробити повноцінний автоматичний переклад. Хоча базований на лінгвістичних правилах підхід не зміг вирішити проблем перекладу, але була закладена база для подальших розробок й досліджень.

1.1.3. Обробка природньої мови та наука про штучний інтелект

Дослідження природних мов та штучного інтелекту, що проводили ентузіасти, потребувало часу, щоб оговтатися від сподівань і очикувань. На це пішло май же чотирнадцять років, до 1980р.

Певним чином зупинка штучного інтелекту започаткувала новий етап розвитку NLP, коли від попередніх концепцій машинного перекладу відмовились, а нові ідеї сприяли новим дослідженням, включаючи експертні системи. Поєднання лінгвістики та статистики, яке було популярним на початку досліджень NLP, було замінено темою чистої статистики. Починаючи з 1980-х років започаткували фундаментальну переорієнтацію. Прості наближення замінили глибокий аналіз, а процес оцінки став більш суворим [28].

До 1980-х років більшість систем NLP використовували складні "рукописні" правила. Але наприкінці 1980-х відбулася революція в NLP. Це було результатом як постійного збільшення обчислювальної потужності, так і переходу на алгоритми машинного навчання. Хоча деякі з ранніх алгоритмів машинного навчання, такі як дерево рішень, базувалися на цьому підході, на рисунку 1.2 продемонстровано приклад роботи такої моделі.

Рис.1.2. Приклад ймовірності гри в гольф залежно від погодних умов.

Створювані системи, подібні до рукописних правил старої школи, дослідження все більше зосереджувались на статистичних моделях, які здатні приймати м'які, імовірнісні рішення. Протягом 1980-х років IBM відповідала за розробку декількох успішних, складних статистичних моделей.

У 1990-х рр. Популярність статистичних моделей для аналізу природничих мов різко зросла. Чисті статистичні методи NLP стали надзвичайно цінними, щоб не відставати від величезного потоку Інтернет-тексту. Н-грами стали корисними, розпізнаючи та відстежуючи скупчення лінгвістичних даних, чисельно.

У 1997 році були запроваджені моделі LSTM (рис. 1.3) та модель з періодичною нейронною мережею (RNN), які знайшли свою нішу в 2007 році для обробки голосу та тексту.

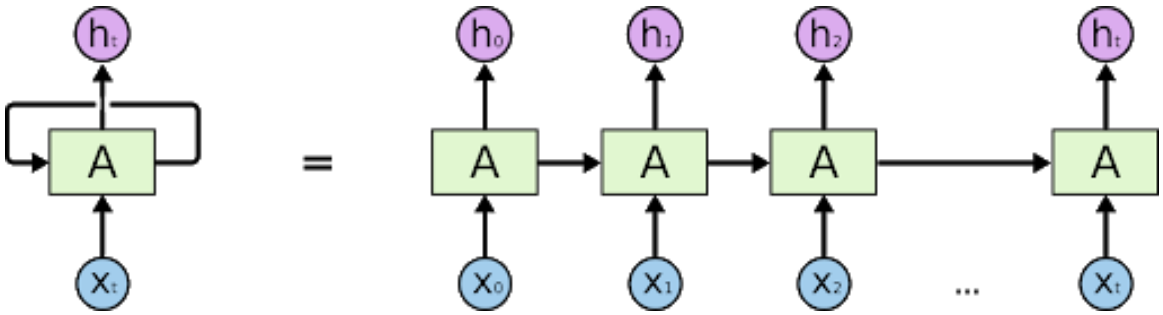

Рис. 1.3. Модель LSTM

В даний час моделі нейронних мереж (рис. 1.4) вважаються передовою науково-дослідною роботою в розумінні NLP і генерування тексту та мовлення.

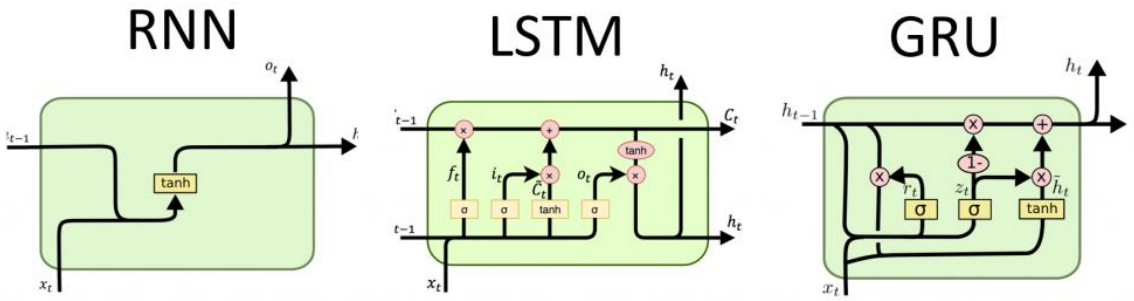

Рис 1.4. Види нейронних мереж.

Важливим етапом розвитку NLP були 1970-1980 роки, для цієї епохи був характерним стрімкий розвиток експертних систем. Методи цієї епохи все ще базувалися на лінгвістичних правилах, алгоритми розбору та побудови речень використовувались сукупність знань про морфологію, синтаксис та семантику мов. Прикладом цього підходу є теорія «Зміст-Текст» Мельчука, що вводить концепцію тлумачно-комбінаторного словника (на рис. 1.4 продемонстрована побудова глибокого синтаксичного дерева для фрази "I think Mary’s brother is late”).

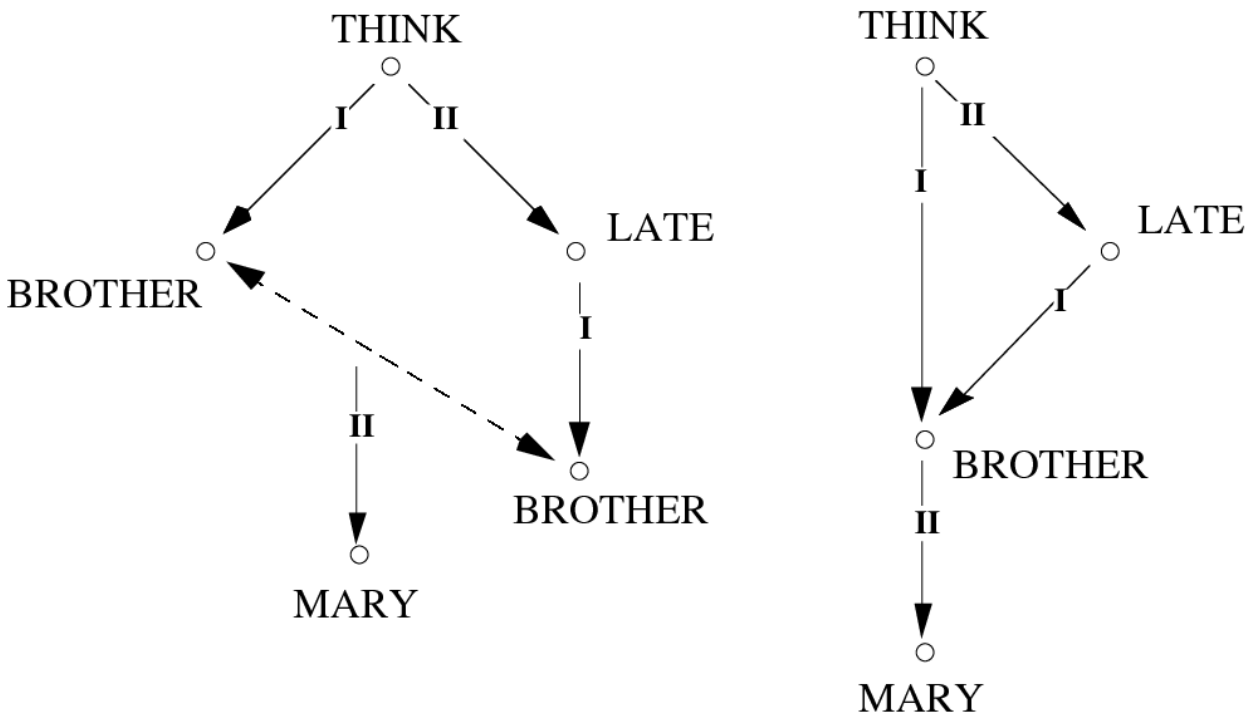

Рис 1.5. Глибинне синтаксичне дерево фрази

Однак, задача машинного перекладу все ще залишалась невирішеною, а експертні системи врешті-решт зустрілись з трудомісткістю формалізації людських знань вручну. Отже, лінгвістичні теорії до цього часу були розроблені й вони пояснювали будову тверджень в окремих європейських мов, але використовувати ці напрацювання на практиці в автоматичних системах

перекладу було складно, тож експертні системи побудовані на лінгвістичних правилах залишились експериментальними.

У 1990-2000 роки на допомогу прийшли методи машинного навчання та базовані на них технології аналізу великих даних. Автоматична побудова алгоритмів обробки мови, як й будь-яка інша автоматизація, знижує витрати, проте до цього періоду область NLP була істотно доповнена та систематизована лінгвістами в результаті попередніх дослідів. Машинному навчанню довелося перейняти у лінгвістики базові принципи й підходи до вирішення практичних задач. Прикладом цього підходу є популярна бібліотека Stanford CoreNLP, що дозволяє виконувати усі етапи стандартного пайплайна машинної обробки текстів: токенізація (розбиття тексту на слова), зведення слів до нормальної

форми, визначення їх граматичних ознак (себто визначення частин речі), виявлення іменованих сутностей та зв'язків між ними, побудова синтаксичних дерев (рис. 1.1). Для кожної з цих проміжних задач лінгвістами були створені та розмічені корпуса текстів, на яких навчались відповідні моделі (у 1992 році був створений Консорциум лінгвістичних даних - Linguistic Data Consortium). Таким чином, машинне навчання використовується для вирішення задач, визначеними лінгвістами, з використанням накопичених знань).

Рис 1.6. Демонстрація пайплайну машиної обробки Stanford CoreNLP: визначення частин речі, сутностей, встановлення зв'язків між ними

З ростом доступних обчислювальних потужностей й появою в 2010-х роках технологій глибинного навчання, набуває популярність інший протилежний підхід – наскрізне вирішення завдань (end2end), де всі проміжні стадії обробки мови глибокі нейронні мережі формують автоматично. При цьому, вони ще й покращають кінцевий вихідний результат. Якість сучасного машинного перекладу на основі глибинного навчання для деяких мов вже близька до людського перекладу.

З end2end підходом в глибинному навчанні мові, у свою чергу, все більшу грає роль навчання «без вчителя» з використанням великих дешевих масивів нерозмічених (тобто не класифікованих й не поділених на складові частини)

текстів. Надлишок даних дозволяє навчати швидко достатньо складні язикові моделі з великою кількістю параметрів й недосяжною до цього якістю вирішення практичних задач. Цілком очевидно, що створення таких моделей вручну навіть великим колективом лінгвістів майже нереально.

Складність розуміння природньої мови в основному пов'язана с багатозначністю слів, значення яких, зазвичай, залежить від контексту. У нейронних моделях значення слів представлено у формі векторів, над якими можна проводити різні складні операції. Ці вектори й автоматично навчаються у процесі.

Наприклад, word 2 vec автоматично породжує семантичний словник мови (рис 1.3), тобто набір семантичних векторів слів, у якому кожний вектор містить усі можливі значення одного слова. Значення слів при перекладі визначається за допомогою їх контексту, тобто за допомогою оточуючих їх слів. Треновані таким чином вектори можна потім використовувати для вирішення будь-яких задач NLP замість звичайних словників [41].

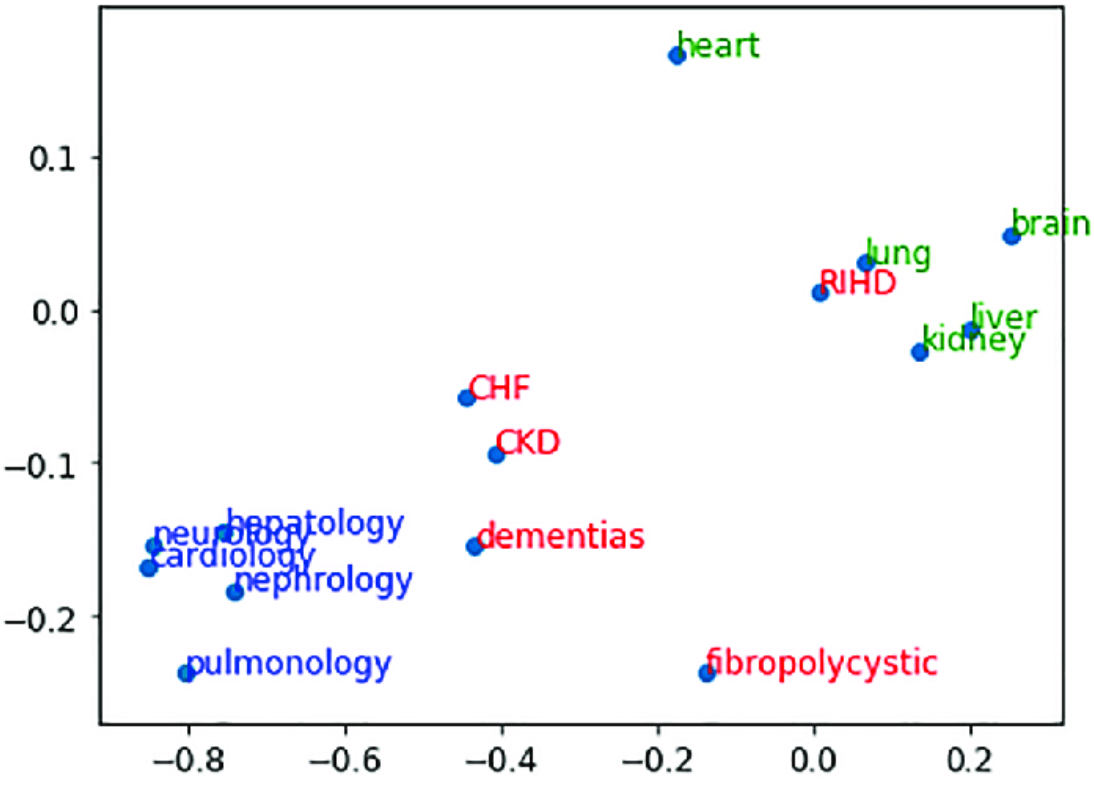

Рис 1.7. Візуалізація семантичного словника медичних термінів, породженого word2vec моделлю, у 2-вимірному просторі

В навчених таким чином нейронних моделях мови робиться наступний крок у розвитку машинного перекладу. Вони здатні у процесі визначати значення всіх слів у конкретному контексті, тобто кодують сенс самого контексту. При цьому сучасні моделі здатні працювати з досить великим контекстом, що може містити тисячі слів.

Прикладом використання end2end підходу являється перекладач Google, який складається з двух нейронних мереж кодера й декодера. Ця система працює по наступному принципу: кодер знаходить значення фрази, враховуючи порядок й смислові вектори її слів, а потім декодер з отриманого коду генерує фразу на іншій мові.

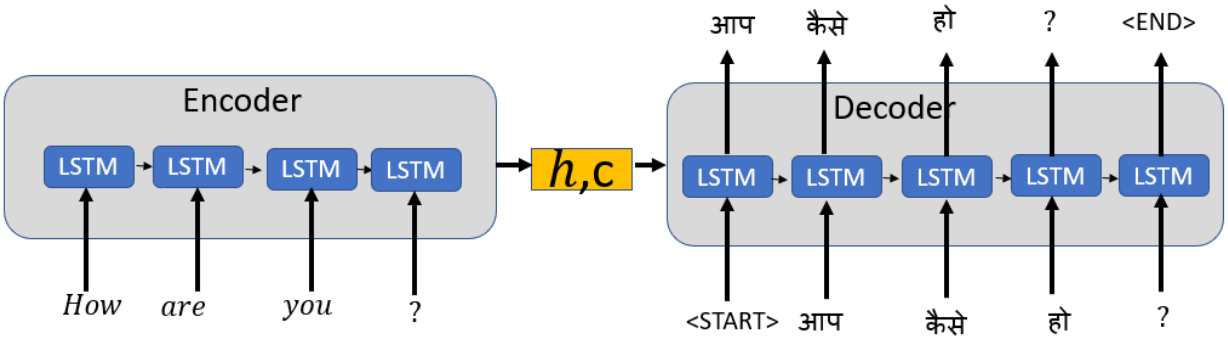

Рис 1.8. Використання end2end підходу у Google translate

В цілому глибинне навчання істотно підвищило й продовжує покращувати якість обробки людської мови, а для деяких задач вже й взагалі порівняно з результатом роботи людини. Це у свою чергу змушує замислитись над ціновими характеристиками цих рішень, критичними для будь-якого ринкового продукту. Справа у тому що, хоча end2end являється піком розвитку технологій NLP, разом з тим цей підхід потребує великих масивів даних, обчислювальних потужностей й дешевої електрики, тому такі складні й розвинуті рішення у найближчому майбутньому будуть доступні лише великим компаніям. Однак, можна очікувати, що при досягненні деякого порогу якості рішень почнеться технологічні змагання за їх собівартість.

1.2. Застосування технік обробки природної мови

Ідея використовувати механізми і алгоритми для генерації мови надихала людей з різних культур в різні моменти нашої історії. Однак, саме в онлайні ця здатна на багато що форма словоутворення може знайти собі відповідний притулок - в оточенні, де особистість співрозмовників стає все більш неоднозначною, а, можливо, і менш важливою. Ми ще побачимо, до яких наслідків для мови, спілкування і нашого самовідчуття як людей (так сильно зав'язаного на нашу здатність говорити на природній мові) все це може привести.

GPT-2 - це приклад техніки NLP під назвою «моделювання мови», в якій обчислювальна система вбирає статистичні закономірності мови з метою його імітації. Як предсказательная система на вашому телефоні - вибирає варіанти вводяться слів на підставі тих, що ви вже використовували - GPT-2 може взяти рядок тексту і передбачити, яке в ній буде наступне слова на основі ймовірностей, властивих даному тексту. GPT-2 можна вважати нащадком статистичного моделювання мови початку XX ст.

Результати вийшли вражаючими. У записі в блозі OpenAI повідомляється, що GPT-2 може генерувати штучний текст у відповідь на запити, імітуючи будь-який запропонований стиль тексту. Якщо відправити запит у вигляді рядка з поезії, вона може згенерувати у відповідь рядок в стилі поета романтичної епохи. Якщо дати системі рецепт торта, ви отримаєте у відповідь новий рецепт.

Система відповідає точно не кожен раз, але це, тим не менш, виглядає як часткова реалізація мрії Готфріда Лейбніца про машину, яка генерує мову і здатною відповісти на всі питання людини.

1.3. Висновки до першого розділу

У цьому розділі ми прослідили основні етапи розвитку напряму NLP від перших ранніх спроб опанувати та систематизувати мову у середньовіччі й до

потужних нейронних мереж, що здатні перекладати тексти та вести діалог майже на рівні людини.

В цілому шлях розвитку NLP можна описати наступним чином. Спочатку формувався теоретичний базис лінгвістами, потім мову намагалися представити у вигляді формальних правил логіки, але особливості людської мови не дозволяли звести її до обмеженого ряду логічних правил, після цього були статистичні праці у області лінгвістики й лише потім зі зростанням обчислювальних потужностей нейронні мережі стали застосовувати для виявлення закономірностей у мові.

Кожен з цих етапів надзвичайно важливий. Наприклад, лінгвістичні правила самі по собі не використовуються сьогодні у системах обробки текстів, проте вони заклали фундаментальну теоретичну базу, яка дала розробникам можливість створювати системи, що здатні опрацьовувати текст. Статистичні дослідження у області мови були важливі для розуміння статистичних властивостей мови та як текстову інформацію можна подати у числовому вигляді, щоб комп’ютер міг з нею працювати. Великі нейронні мережі хоча й дали поштовх розвитку напрямку NLP не змогли б досягти цих успіхів без надбань попередніх етапів.

РОЗДІЛ 2
МЕТОДИ ТА ПІДХІДИ ДО КЛАСИФІКАЦІЇ ТЕКСТІВ

2.1. Принцип автоматичної класифікації текстів

Перед тим як розглянути основні використовувані методи та підходи до класифікації текстів й обрати оптимальний варіант, потрібно для початку описати проміжні стадії та завдання, які потрібно вирішити у процесі створення системи для класифікації текстів. Розуміння складності проміжних завдань й чітка постанова задачі дозволить нам зробити оптимальний вибір. Таким чином, перед початком розробки ми повинні відповісти на наступні запитання.

По-перше, будь-яка модель автоматичної обробки текстів потребує навчальних даних, базуючись на яких, вона буде робити припущення до якого класу належить той чи інший текст. У навчальної вибірки є такі характеристики як розмір, чистота або якість (формат тексту, відсутність помилок тощо.) та збалансованість даних. Під збалансованістю даних мається на увазі наскільки рівномірно тексти будуть розподілені між усіма класами, теж саме відноситься й до частоти слів - одні слова можуть зустрічатись частіше, а інші рідше, усе це впливає на точність моделі. Загалом, ці характеристики переважно залежать від джерела інформації, з якого збирається вибірка. Отже, перед розробкою потрібно розуміти звідки можна зібрати навчальну вибірку.

По-друге, потрібно вирішити проблему представлення текстової інформації у математичному вигляді. Очевидно, що комп’ютер не може розуміти значення слів у такий же спосіб, як й людина, тому потрібно якимось чином кодувати значення слів або отримувати числові характеристики тексту. Окрім розуміння як ми будемо отримувати значення слів чи тексту, потрібно також й визначати, а які саме потрібні характеристики для класифікації текстів. У кінцевому підсумку, ми повинні отримати числову характеристику тексту, на основі якої ми будемо навчати нашу модель й визначати метрики для неї.

По-третє, потрібно встановити задовільну межу точності класифікатора та брати до уваги час й обчислювальні можливості, що є в наявності у розробника. Сьогодні існує багато розвинутих моделей для аналізу тексту у відкритому доступі від таких технічних гігантів як Google, Amazon, Microsoft та інших, вони можуть дійсно давати точні результати, але перед початком роботи потребують налагоджування та оптимізації під конкретні випадки, що може потребувати великих об’ємів даних. Однак, залежно від вимог до системи, складності текстів, кількості класів та мов, можна підібрати більш прості для інтерпретації та побудови моделі, що будуть давати порівняно точні результати.

У висновку, проблему обробки текстів можна звести до 4-х пунктів: вимоги до точності, навчальна вибірки й її джерела, модель представлення даних та класифікатор. Ми зможемо визначатись з вибором моделі представлення даних й класифікації тільки після того як розглянемо кожний з вище наведених етапів та відповімо на всі запитання

2.2. Проблема отримання й пошуку інформації з текстів

Здається, що для AI природня мова не повинна бути важкою, зазвичай говорячи про AI людина уявляє асистента, що спокійно розмовляє людською мовою. Однак результати досліджень більш ніж півстоліття демонструють, що вивчення людської мови є надзвичайно складним завданням для інтелектуальних алгоритмів. В школі людина вивчає граматику, синтаксис, морфологію, в цілому мову можливо звести до формальних правил, будь-яке твердження можна розібрати на частини. На перший погляд здається що запрограмувати правила не повинно бути складно.

Проте уявимо, що ми бажаємо написати просте правило для побутового помічника. Наприклад, ми говоримо фразу «Будь-ласка вийди за хлібом», ця тривіальне прохання зрозуміле для людини, але якщо вдуматись то саме слово «сходи» несе деяку долю невизначеності, адже не зрозуміло куди потрібно сходити й таких прикладів може бути багато. Навіть якщо ми поставимо ціль

зробити лише помічника, що купує хліб, то все рівно виникнуть проблеми, якщо система не знає слова батон чи булка. Отже, навіть у простому випадку застосувати синтаксичні правила буде важко, адже мова описує навколишній світ й щоб розуміти зміст сказаного, потрібно мати деяку ступінь обізнаності.

Такий приклад може виглядати перебільшенням, так як здається що задачі людей в основному прості й вони говори зрозуміло. Проте навіть сьогодні автоматичні помічники й асистенти розуміють нас не краще, ніж п’ятирічна дитина. Складність обробки мови людини спричинена 2 факторами: по-перше, ,як було зазначено вище, слова можуть мати різне значення залежно від контексту, по-друге, навіть літературна мова включає в себе настільки велику кількість слів, що усі їх важко описати й визначити.

2.2.1. Закон Ципфа

Уявимо, що дослідник має мету створити всеосяжний словник, тобто написати статтю для кожного слова. Він збирає велику кількість текстів (корпус) й створюєте список усіх слів й потім упорядковуєте їх у порядку спадання частоти. Після цього він починає пояснювати найбільш вживані слова. Кожен день він вимірює який процент корпусу було охоплено. У перший день охоплення корпусу 15%, на другий 5, на третій 1, а потім приріст зводиться до долі відсотків. Можна припустити, що було обрано невдалий корпус текстів, але навіть якщо взяти інший набір текстів, то виявиться така ж закономірність, але з іншими словами. Даний ефект пояснюється законом Ципфа (рис 2.1).

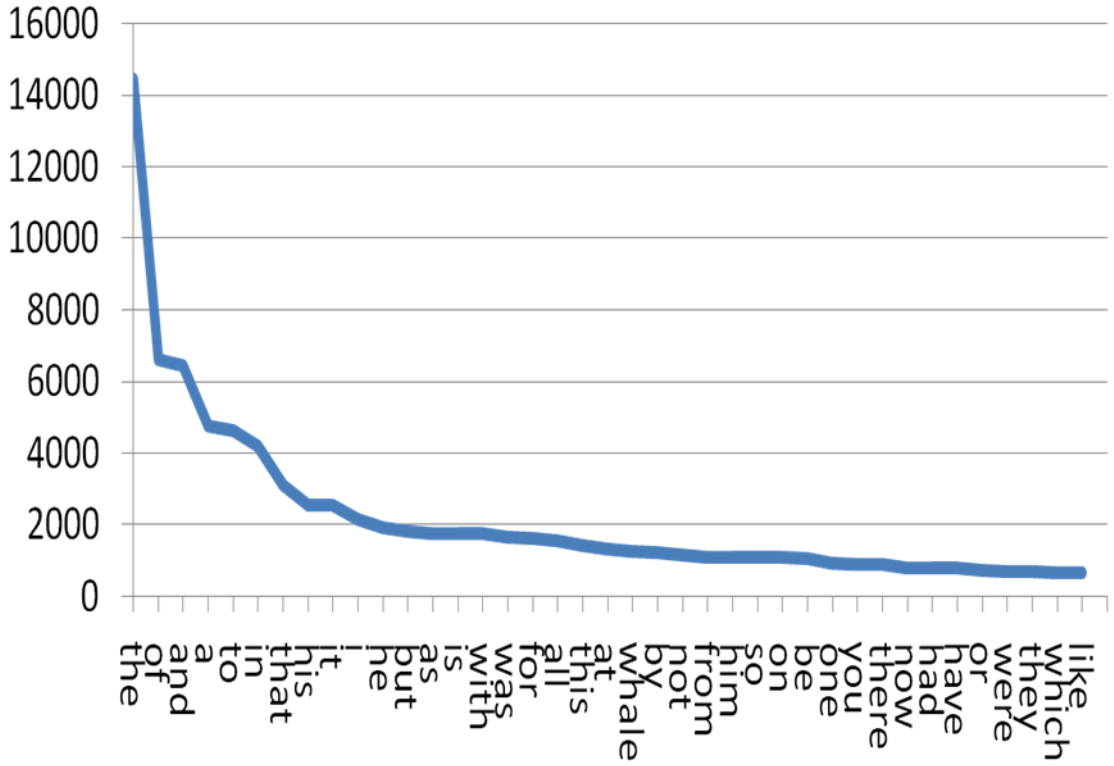

Рис 2.1. Закон Ципфа на прикладі текстів з wikipedia англійською мовою

Аналогічно це правило працює й для інших явищ, пов’язаних з людською мовою. Наприклад, можливо запрограмувати 50% чи 60% запитів для асистента, але такий асистент все рівно не буде справляти враження інтелектуального помічника [41].

2.3. Математичні моделі представлення текстових даних для задачі класифікації текстів

Обсяг текстових даних величезний й проблема текстових даних полягає в тому, що їх потрібно представляти у форматі, який комп’ютер міг би використовувати для вирішення практичних завдань, пов’язаних з мовою, у нашому випадку для класифікації текстів. Мається на увазі, що нам потрібно звести слова та речення до цифрових значень на основі яких наш алгоритм буде класифікувати тексти. Моделі представлення даних можна розбити умовну на 2 групи: статистичні методи та векторне представлення слів [35].

Для статистичних методів характерний простий математичний підхід до формування ознак тексту, що не враховує порядок слів й контекст. Однак, при цьому текст повинен бути підготовлений для обробки, тобто усі слова повинні бути зведені до нижнього регістру та словникової форми, прибрані знаки пунктуації, лишні пробіли та відступи, тощо. Також потрібно зауважити, що такі моделі, очевидно, не призначені для задач складніше аніж класифікація текстів, вони не вирішують проблему Ципфа, тобто не можуть вгадати значення нового слова з контексту. Ця група включає 2 методи Bag-of-words та TF-IDF:

1. Bag-of-words (модель «торба слів») – це спрощене подання тексту, яке представлене у вигляді мультимножин (частот) його слів або комбінацій букв (н-грам).

2. TF-IDF (від англ. TF — term frequency, IDF — inverse document frequency) – статистичний показник, що використовується для оцінки важливості слова у контексті документа, що є частиною колекції документів або корпусу. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання слова у інших документах колекції. Суть цього методу оцінки полягає у тому, що слова часто вживані лише у певному наборі текстів будуть отримувати високу вагу, а розповсюджені та загальні слова для усіх документів будуть отримувати низьку вагу. Даний показник можна представити у вигляді добутку 2 статистичних показників тексту $tf-idf\left(t,d,D\right)=tf\left(t,d\right)\times idf\left(t,D\right)$.

TF – це відношення числа входжень обраного слова до загальної кількості слів документа, дана характеристика описується наступною формулою (2.1).

$$tf\left(t,d\right)=\frac{n_t}{\sum_k n_k}, \tag{2.1}$$

де

n_t – це число входжень t в документ d;

$\sum_k n_k$ – сума числа входжень кожного слова k в документі.

IDF – інверсія частоти, з якою слово зустрічається у документах колекції, цей коефіцієнт описується формулою (2.2).

$$idf\left(t,d\right)=\log\frac{\left|D\right|}{\left|\left\{d_i\in D\middle|t\in d_i\right\}\right|}, \tag{2.2}$$

де

$\left|D\right|$ – загальне число документів у колекції

$\left|\left\{d_i\in D\middle|t\in d_i\right\}\right|$ – число документів з колекції D, у яких зустрічається слово t хоча б один раз.

Методи трансформації слів у семантичні вектори - це рішення, що переважно побудовані на основі великих нейронних мереж «без вчителя» [1], на вхід вони отримають великі корпуси текстів, а потім приводять слова до векторного вигляду. На відміну від статистичних методів ці рішення враховують контекст й формують вектори слів виходячи з контексту у якому вони зустрічаються. Ідея використання мереж полягає у тому, що вони самі розпізнають семантичні закономірності слів й формують багатовимірні вектори значень, вектори синонімічних слів знаходяться поблизу у просторі (див. рис. 1.3), а ніяк не пов’язаних – далеко. До переваг даної групи входить те, що на вхід ці моделі приймають необроблені та нерозмічені тексти будь-яких мов. До даних моделей відносяться такі моделі як word2vec, Bert та Gensim, вони відрізняються за деталями виконання, але базовий принцип у них один. У наступній частині ми розкриємо їх принцип роботи докладніше.

2.3.1. Алгоритм word2vec як спосіб вирішення проблем, пов’язаних із законом Ципфа

Алгоритми машинного навчання дозволяють автоматично виявляти закономірності у даних автоматично без потребі у складних правилах. Однак,

при застосуванні цих методів до мовних даних довгий час зберігалася проблема, що неможливо зрозуміти значення нового слова, якщо воно не зустрічалося в процесі навчання моделі. Наприклад, при побудові моделі можна пропустити один з багатьох синонімів до слова, який хоча й не є широковживаним у текстах, але може зустрітись. Проте, складання всеосяжного словника синонімів не є ефективним рішенням згідно з законом Ципфа (див. рис. 2.1). Один з найбільш важливих алгоритмів, що істотно покращив розуміння семантики слів, був алгоритм word2vec, розроблений у 2013 році командою Google під керівництвом чеського дослідника Томаса Міклова [41].

Сформуємо задачу іншим чином, нам не потрібно шукати усі можливі синоніми до кожного слова, але нам потрібно розуміти значення слів математично. Під цим ми розуміємо, шо кожне слово потрібно знайти відповідний вектор у багатовимірному просторі (у перший працях Миколова використовувався простір з 50 чи 100 вимірів), так щоби близьким за змістом словам відповідали геометрично близькі вектори, а далеким словам – далекі.

У табл. 2.1 наведено результат роботи алгоритму word2vec на прикладі декількох слів. Ми бачимо, що найближчими сусідами до слова «green» виявились інші назви кольорів, а до слова «huge» - означення, що позначають розмір, поряд зі словом «probability» знаходяться терміни з теорії ймовірності.

Таблиця 2.1

Таблиця слів-синонімів, породжена word2vec для англомовного тексту

probability	green	huge
likelihood	violet	large
chance	red	massive
distribution	yellow	vast
random	purple	enormous
particle	white	substantial
discrete	brown	immense
parameter	black	significant
variables	dark	small

Найбільш цікавим є те, що на вхід алгоритму подається великий корпус текстів з мінімальною обробкою (наприклад, без видалення чи відділення пробілами знаків пунктуації від слова, зведення всіх слів до нижнього регістру чи приведення їх до нормальної форма). Система сама вивчає поняття семантичної схожості слів й створює непогані списки синонімів.

2.3.2. Дистрибутивна гіпотеза у векторному представленні слів

Для того щоб зрозуміти як будується дана модель й чому вона працює необхідно згадати дистрибутивну гіпотезу (рис 2.2).

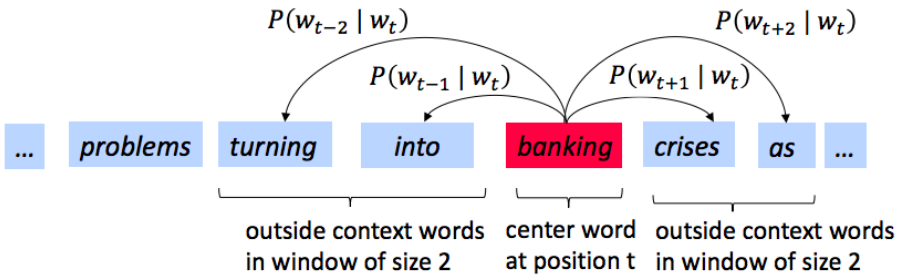

Рис 2.2. Ілюстрація до дистрибутивної гіпотези.

Дистрибутивна гіпотеза – це поняття з лінгвістики, що бере свій початок від праць, опублікованих в 50-х роках 20 століття, й розвинуте на початку 2000-х [40], з розширенням можливостей статистичного аналізу лінгвістичних явищ. Ідея даної гіпотези полягає у тому, що значення слів переважно визначається контекстом, в котрому вони зустрічаються. Наприклад, знання про те, що слова «рогалик» або «паляниця» доволі часто зустрічається у схожому контексті (наприклад, «спекти», «розломити», «приготувати» тощо.), що дозволяє зробити висновок, що у значенні цих слів є спільний аспект. Дана гіпотеза була перевірена статистичними й психологічними експериментами.

Отже, спираючись на дану гіпотезу дослідники побудували модель, котра повинна передбачати слова, виходячи з їх контексту, або навпаки вгадувати

сусідів, коли дано слово. Обидві версії були розглянуті й обидві давали приблизно схожий результат.

Точніше сказати, що для кожного слова модель навчалась передбачати правильне імовірнісне розподілення сусідніх слів (на відстані 1 чи 2 слова зліва й справа у реченні). Як ми бачимо у таблиці 2.1, для слів «green» й «blue» розподіл ймовірностей для сусідів дуже схожий, тому модель побудувала близьке представлення цих слів.

Дану працю можна вважати першим успішним прикладом використання методу «transfer learning» - це використання навченої моделі для аналізу інших текстів. Іншими словами, сутність цього методу полягає у тому, що ми навчаємо модель вирішувати задачу A, а потім використовуємо внутрішнє представлення даних, вивчених моделлю, для вирішення задач B й C [1]. Цей метод особливо важливий, коли ми маємо багато даних для вирішення задачі A, а для B й інших завдань недостатньо. У випадку алгоритму word2vec завдання A є передбаченням слова згідно його оточення. Для отримання вхідних даних для аналізу підходять будь-які тексти, що написані людьми, з мінімальною попередньою автоматичною обробкою. На виході ми отримуємо модель, що здатна зводити значення слів вхідного тексту до векторного представлення (див. рис. 1.3), яке потім може бути використано для інших задач іншими моделями. Таким чином, Отримане математичне представлення слів використовується для різних завдань, пов’язаних з пошуком слів, це може бути як пошук інформації, класифікація або порівняння текстів, так й переклад або генерація нових текстів, адже достатньо лиш замінити слова на синонімічні й ми отримуємо новий текст.

Проте потрібно зазначити, що для навчання моделей word2vec потрібні великі об’єми текстів, багато часу й обчислювальні потужності, які можуть забезпечити хмарні рішення Google, Microsoft або ферми з десятків чи сотень GPU залежно від об’ємів даних та бажаної швидкості.

2.3.3. Рекурентні нейронні мережі й мовні моделі

Однак, для розуміння значень речень часто недостатньо лише розуміння окремих слів. Наступним важливим етапом у розвитку алгоритмів роботи з текстами стало створення рекурентний нейронних мереж (RNN – Recurrent Neural Networks), що здатні вирішувати завдання побудови мовних моделей [17].

Нижче на рис. 2.3 приведено схематичну структуру RNN для обробки природної мови.

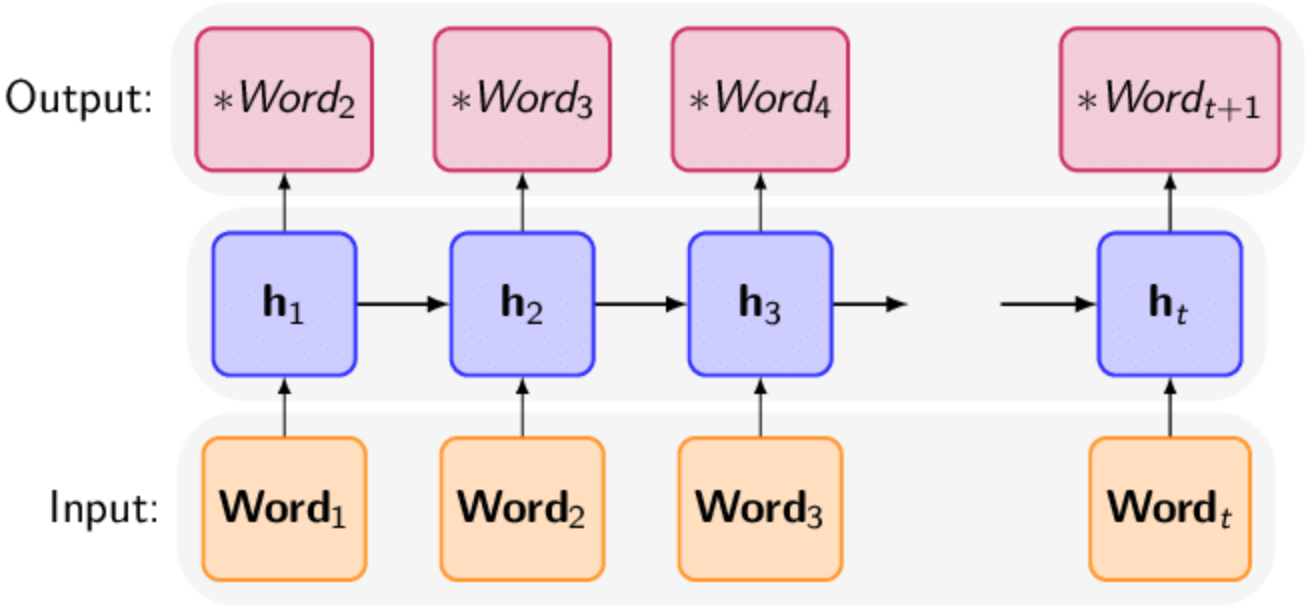

Рис 2.3. Схема RNN мережі

Мовна модель – це розподілення ймовірностей послідуючих слів у реченні, якщо не було дано початок. Наприклад, речення «Я пішов до пекарні за …» може бути продовжено з великою ймовірністю словами «хлібом», «тортом», у той час як у інших слів таких як «сигарети», «ліки» вірогідність менша. У реченні «міжнародна космічна» найбільш вірогідним продовженням буде «станція» або «команда».

RNN працює наступним чином. Вона «читає» речення зліва направо й на кожному кроці формує вектор (відомий як «приховане представлення»), в якому повинна бути вся закодована важлива інформація про пройдений відрізок тексту. На основі прихованого представлення передбачується наступне слово. Потім цей

крок повторюється, тобто модель приймає на вхід наступне слово, але вже нове значення обчислюється з урахуванням старого прихованого представлення, як результат прочитаний контекст не втрачається, після цього отримується новий контекст й на основі нього вгадується нове слово й цей процес повторюється.

Описаний вище принцип RNN є характерним для нейромережевих моделей коміркового типу, кожна комірка – це функція, що обчислює наступне приховане представлення на кожному кроці рекурентної мережі.

Найбільш вдалими та ефективними мережами коміркового типу виявились мережі на базі довгої короткочасної пам’яті (LSTM – Long Short Term Memory) та вентильного рекурентного вузла GRU (GRU – Gated Recurrent Unit). Обидві ці функції реалізують складні перетворення вхідних векторів, що дозволяє не втрачати інформацію, яка з’явилась у контексті декілька десятків кроків назад, що забезпечує врахування довгого контексту речення. Це важливо, наприклад, у великих фразах, які починаються словами «Я впевнений, що…» або «Я не згоден з твердженням…».

Звісно, сама по собі задача передбачення наступних слів в реченнях зустрічається не так вже й часто. Це може бути корисним лише у деяких випадках, як наприклад, відображення підказок при написанні тексту або при виправленні помарок (потрібно обрати правильній варіант залежно від контексту речення).

Однак, важливість цієї задачі у тому, що отриманні приховані представлення, вивчені моделлю, містять важливу інформацію про семантику всього прочитаного речення, й можуть бути використані для більш успішного рішення інших задач.

Як й у випадку з word2vec важливо, що мовна модель може навчатись на будь-якій кількості текстів будь-яких типів без спеціальної обробки. Для багатьох інших методів потребується велика кількість оброблених та класифікованих текстів, що являється дуже трудомісткою та важким завданням, особливо якщо потрібна велика точність. Прикладом завдань, які можуть

вирішувати подібні моделі: класифікація текстів, виділення сутностей з тексту, кластеризація, оцінка семантичної схожості речень.

2.4. Порівняльна характеристика моделей представлення даних

Ми дослідили сучасні широко використовувані моделі представлення даних, кожна з моделей має свої переваги й недоліки.

Потрібно відмітити, що мовні моделі та семантичні вектори слів, що будуються нейромережами, є найбільш досконалими та розвинутими моделями сьогодні. Ці моделі передають значення слів у математичній формі, дозволяють вирішувати проблеми, пов’язані з законом Ципфа, тож їх можна використовувати для різних завдань.

Однак, вибір моделі залежить від вимог до швидкості навчання, точності моделі, об’єму даних та типу задач, що вирішується.

У нашому випадку, вирішується лише задача класифікації текстів, власне сенс невідомих слів для нас не є важливим, тому модель TF-IDF виглядає більш прийнятною.

До TF-IDF недоліків можна віднести те, що класифікатор на базі цієї ознаки потребує попередньо оброблені тексти, але зараз ця проблема вирішується за допомогою спеціалізованих бібліотек. TF-IDF досить зручна для задач класифікації, так як не потребує великого об’єму даних й досить наглядно демонструє відмінності між наборами термінів у текстах різних класів.

Отже, можна зробити висновок, що TF-IDF являється непоганою альтернативою більш громіздким та розвинутим моделям у випадку класифікації текстів.

2.5. Огляд наявних методів та моделей класифікації текстів

Сьогодні існує багато методів розпізнавання текстів, вони діляться на 2 великі групи: базовані на правилах та на моделях машинного навчання (евристичний підхід). У кожного з цих підходів є свої переваги та недоліки.

Базований на правилах підхід до класифікації текстів полягає у тому, що лінгвісти аналізують стилістичні особливості властиві визначеним класам текстів, закономірності та лінгвістичні конструкції мови, з яких пізніше формуються формалізований набір правил. Цей підхід може мати більшу точність порівняно з евристичним, за умови, що формат та різновид текстів чітко окреслений. Однак недоліком цього методу є те, що його використання дещо обмежене за розміром тексту та його структурою, тож він може невірно працювати на текстах незвичного формату чи структури. Окрім цього, попередній досвід використання цього підходу показав, що систему базовану на правилах важко розширяти за кількістю категорій, адже кожний випадок потребує повторного лінгвістичного аналізу та створення нових правил, які б не суперечили старим. Отже, сьогодні цей підхід виглядає малоперспективним й не представляє особливого інтересу для комп’ютерних наук [28].

Евристичний підхід працює на основі імовірнісних моделей машинного навчання, які у свою чергу підрозділяються за типом навчання на 2 групи: з вчителем та без. Основна відмінність між цими 2 принципами полягає у тому, що модель з вчителем включає в себе етап навчання, який потребує прикладів вхідних даних та правильних відповідей. Під час цього етапу відповіді моделі порівнюються з наданими прикладами й на основі цього «вчитель» регулює коефіцієнти моделі допоки не буде досягнута бажана точність. Прикладами моделей з вчителем можуть бути такі моделі як наївний Баєс, метод опорних векторів, згорткові та рекурентні нейронні мережі, логістична регресія тощо.

Моделі без вчителя у свою чергу намагаються самостійно віднайти взаємозв’язки та закономірності між наданими об’єктами й розбити їх на групи

за спільними ознаками. До моделей без вчителя можуть бути віднесені моделі на основі латентно-семантичного аналізу (ЛСА), методу к-середніх та інші.

Хоча між цими 2 групами методів є істотні відмінності, вони мають ряд загальних переваг та недоліків. Перевагами цих моделей являється те, що їх можна порівняно легко розширити новими класами та регулювати їх параметри. Окрім цього, евристичні методи здатні можуть відносно точно класифікувати незвичні чи невідомі їм тексти й не потребують активного втручання спеціалістів предметної області при кожному новому випадку. До недоліків цих методів можна віднести наступні фактори: їм бажано мати велику вибірку для навчання чи кластеризації та потрібно уважно підходити до конструювання та вибору ознак класифікації. На додачу до цього, для моделі з вчителем важливо уникати перетренування, щоб вона могла коректно класифікувати тексти поза навчальної вибірки. Сьогодні, саме евристичні моделі в основному вирішують завдання класифікації та відбору в різноманітних сервісах.

У своїй роботі ми будемо розглядати моделі класифікації текстів з вчителем з декількох причин. По-перше, результати моделей без вчителя можуть бути спірними й неочевидними, по-друге, в роботі важливо щоб класи мали зрозумілі для користувача імена, модель же без вчителя просто розбиває множину об’єктів на умовні групи без осмислених назв, по-третє, моделі з вчителем більш популярні та надійні, частково з причини, що їх похибку можна отримати під час навчання. Отже, для даної роботи підходять лише моделі класифікації з вчителем.

2.5.1. Наївний Баєс

Одним з найбільш популярних та простих методів є метод наївного Баєсу, він, наприклад, часто використовується для простої фільтрації спаму. В основі цього методу лежить теорема Баєсу. Слово наївний у назві присутнє завдяки такій особливості, що цей метод для спрощення робить припущення, що усі

явища незалежні один від одного, тобто поява одного слова ніяк не пов’язана з іншим, у даному випадку роль контексту повністю нівелюється.

Теорема Баєсу, - це теорема, шо дозволяє точніше визначити ймовірність події, якщо до цього сталася інша статистично пов’язана з нею подія, вона описується наступною формулою (2.3).

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}, \tag{2.3}$$

де

$P(A)$ – апріорна вірогідність гіпотези A;

$P(A|B)$ – вірогідність гіпотези A, якщо подія B настала;

$P(B|A)$ – вірогідність гіпотези B, якщо подія A настала;

$P(B)$ – апріорна вірогідність гіпотези B.

Принцип даного методу полягає у тому, що він працює з вірогідністю появи ключових слів для кожного класу тексту, тобто у даному випадку приналежність тексту до одного з класів статистично залежна подія від набору слів. Таким чином, для кожного класу вираховується ймовірність, що текст з даним набором слів належить до нього, вкінці обирається клас з найбільшою вірогідністю [35].

Найбільша перевага цього алгоритму, що він простий та ефективний коли вхідні дані й обчислювальні ресурси обмежені. Цей метод сугубо статистичний, тому йому не потрібно робити множинні ітерації задля зменшення похибки на відміну від регресії й нейронних мереж.

Однак, ця модель має й недоліки, точність цієї моделі задовільна, але гірше ніж в інших моделях, також вона рано досягає ліміту навчання, тобто при досягненні деякого порогу нові приклади для навчання не дають покращення точності моделі.

Даний метод добре працює з малими об’ємами даних та на простих випадках, наприклад його часто використовують для фільтрації спаму, також він може бути корисним у тому випадку, коли потрібно перевірити новий набір ознак для моделі.

2.5.2. Нейромережі глибинного навчання

Інший відомий підхід це нейронні мережи (рис 2.5) глибинного навчання (Deep Learning). Глибинне навчання - це концепція, що за допомогою математичної абстракції нейронної мережі можна симулювати роботу людського мозку, а саме процес отримання інформації та прийняття рішень.

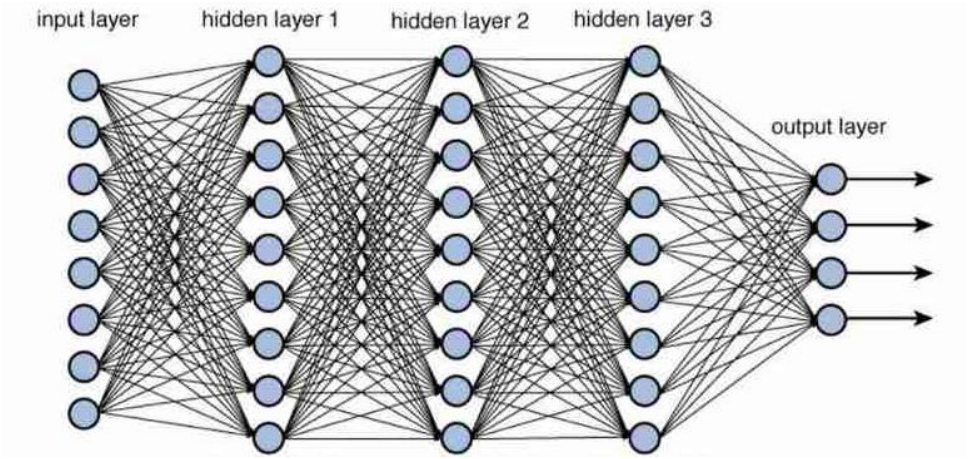

Рис 2.5. Приклад структури нейромережі глибокого навчання

Нейрона мережа - це зважений граф, кожне ребро у якому має різну вагу, у цій моделі вузол моделює нейрон, а його ребра виконують роль дендритів та аксонів, сигнал, що передається між нейронами – це результат активаційних функцій нейронів помножений на вагу відповідного ребра [34]. З одного боку, алгоритми глибинного навчання досить складні, вимагають багато часу для побудови, великих об’ємів даних, а отже й обчислювальної потужності для ефективного навчання. З іншого боку, нейронні мережі показають найкращі результати серед моделей, можуть мають досить високий поріг навчання, завдяки чому датасет можна довго нарощувати й покращувати показники. З цих

причин, нейронні моделі зараз найбільш популярні у великих рішеннях, що потребують високої точності та класифікують велику кількість класів.

2.5.3. Логістична регресія

Інша широко вживана модель для задач класифікації – це логістична регресія (рис 2.6.).

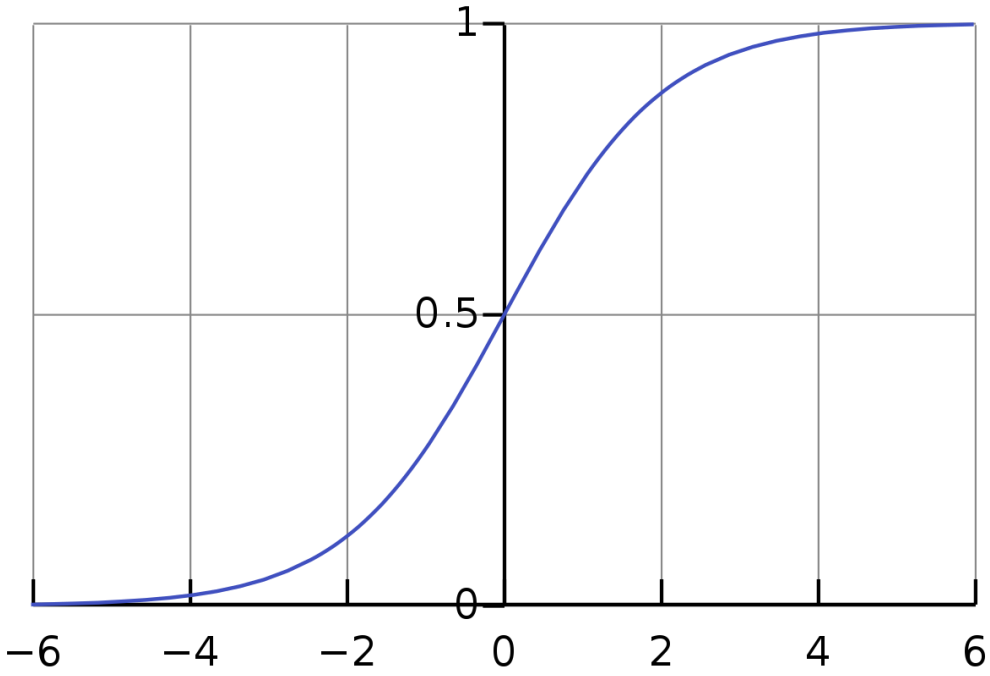

Рис 2.6. Логістична регресія

Логістична регресія – це лінійний класифікатор й статистична модель, що використовується для прогнозування вірогідності настання деякого випадку шляхом порівняння його з логістичним рівнянням, що має форму сигмоїди. Дану регресію використовують для випадків, коли залежна зміна може приймати лише 2 значення, наприклад 1 або 0, тож зміна має розподіл Бернулі. Ця регресія видає відповідь у вигляді вірогідності настання бінарної події (1 або 0) й описується наступною формулою:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}, \tag{2.4}$$

де

z – це сума добутків параметрів x на їх відповідні вагові коефіцієнти , представити це можна у вигляді формули $z = \theta^T x = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$, де θ_0 - це зміщення функції, яке завжди множиться на 1.

Принцип логістичної регресії для класифікації текстів полягає у тому, що вона приймає параметри тексту, такі як частота або ймовірність появи слова у тексті, й обчислює вірогідність належності тексту до одного з класів.

Проте перед тим як прогнозувати події за допомогою моделі її потрібно навчити. Алгоритм навчання логістичної регресії наступний:

1. Спочатку випадковим чином встановлюються початкові параметри моделі й обчислюється результат відповідь для кожного набору вхідних параметрів навчальної вибірки.

2. Вимірюється функція втрати (тобто те наскільки близька наша модель до правильних відповідей) на навчальній вибірці за допомогою формули (2.5), яка виводиться з розподілення Бернуллі.

$$L_{CE}(z) = -[y \log \sigma(z) + (1 - y) \log(1 - \sigma(z))], \tag{2.5}$$

3. Потім за допомогою градієнтного спуску (рис. 2.7), коефіцієнти регресії поступово крок за кроком регулюються так, щоб отримати мінімальний показник втрати й досягти оптимальної точності. Коротко цей крок можна описати у вигляді формули (2.6):

$$\bar{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(y^{(i)}, x^{(i)}; \theta\right), \tag{2.6}$$

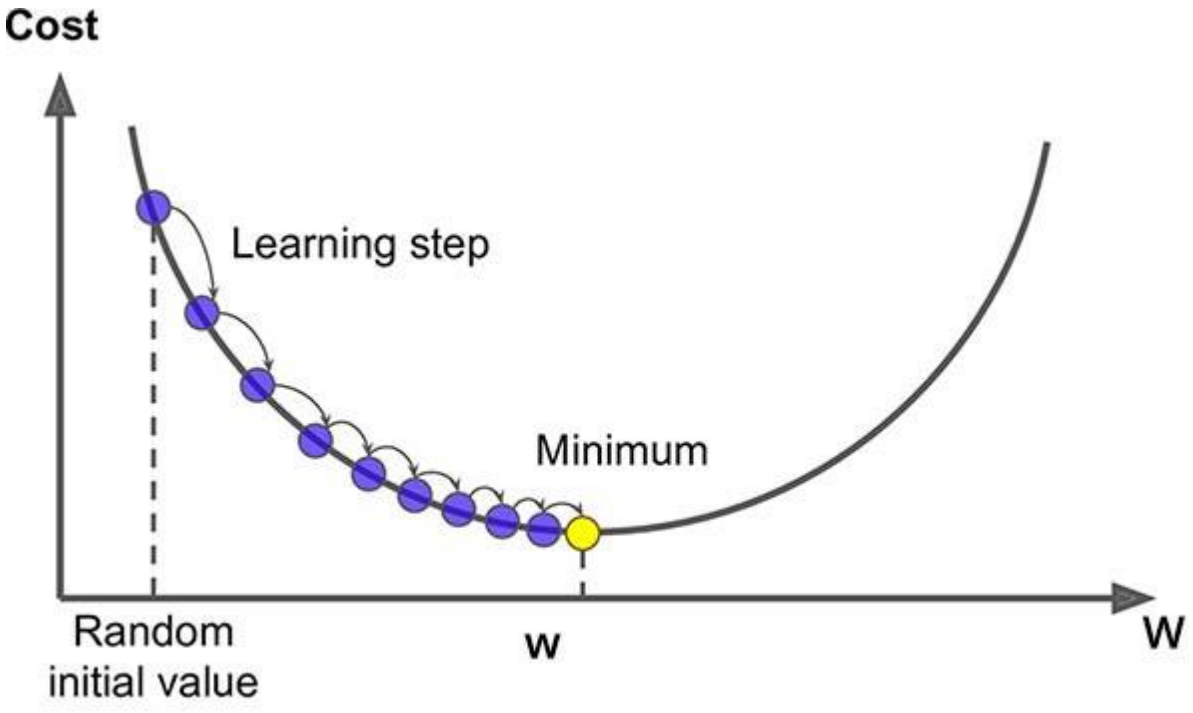

Рис. 2.7. Оптимізація методом градієнтного спуску на прикладі однієї змінної

Хоча логістичну регресію часто порівнюють з методом наївного Баєсу, принцип роботи у цих моделей істотно відрізняється. Наївний Баєс – це генеративний класифікатор, тобто він намагається визначити наскільки вхідні параметри властиві одному з класів, а логістична регресія – це дискримінативний класифікатор, він шукає відмінності між класами й проводить умовну межу між об’єктами [29]. Завдяки таким особливостям логістична регресія показує краще результати на великих об’ємах даних, у той час як Баєс підходить для простих невеликих задач чи для випробовування нових ознак класифікації.

2.5.4. Опорно-векторні машини

Опорно-векторні машини (SVM від Supported Vector Machines) - це набір алгоритмів, що базуються на методі опорних векторів (рис. 2.8).

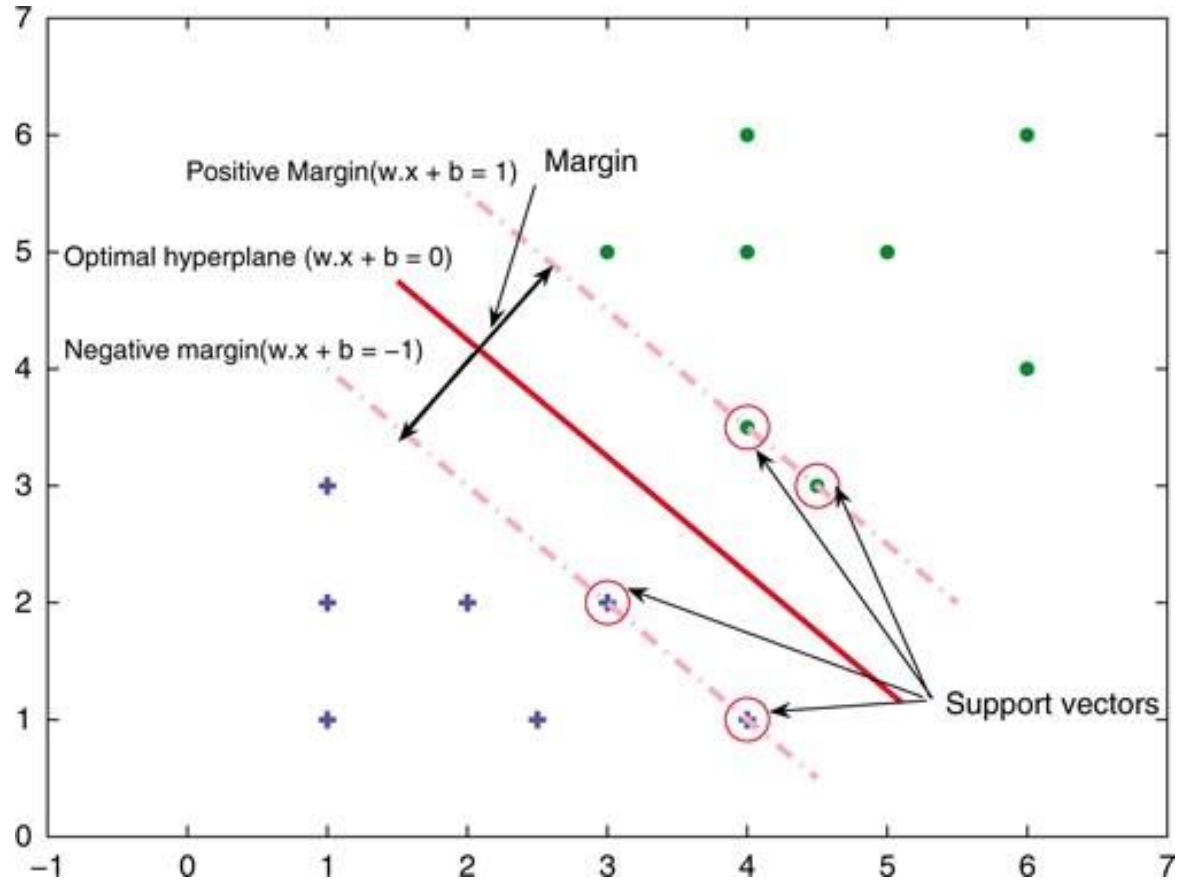

Рис. 2.8. Принцип роботи SVM – гіперплощина, яка діле простір на частини так, щоб відділити класи сутностей

Ці алгоритми можуть використовуватись як для регресійного аналізу, так й для задач класифікації. Цей класифікатор часто порівнюють з логістичною регресію, тому що обидва класифікатори лінійні, тобто в результаті роботи вони умовно ділять простір за допомогою гіперплощини (у випадку 2-вимірного простіру це просто лінія). Відмінність лише у методах, якими досягається ця мета. Логістична регресія поступово зменшує показник врат до потимального значення, а SVM прагне провести таку гіперплощину, що максимально далека від обох класів. У випадку коли класів більше SVM відділяє один клас від інших й проводить більше гіперплощин.

Окрім цього, SVM можуть використовуватись й для кластеризації, у цьому випадку йде мова про навчання без вчителя. У нашій роботі ми розглядаємо лише метод навчання з вчителем, так як набір класів було визначено й нам важливо щоб класи мали власні зрозумілі для користувача імена.

2.6. Порівняльна характеристика основних методів класифікації текстів

У попередніх розділах ми зробили огляд сучасних досліджень в області NLP й методів для вирішення проблем, пов'язаних з отриманням та аналізом текстової інформації, їх способів використання, недоліків й переваг. Розглянуті методи класифікації текстової інформації у наступну порівняльну таблицю.

Таблиця 2.2

Перелік методів класифікації тексту

Метод	Переваги	Недоліки
Базований на правилах	Оптимальні правила побудовані на лексичних закономірностях характерних для текстів цих класів	Займає багато часу, залежність між правилами, можливі «парадокси», нові класи вимагають нових правил
Логістична регресія	Проста оцінка параметрів й інтрепретація результатів, добре працює для більшості задач класифікації, особливо якщо у задачі лише 2 класи	Потребує відносно великих наборів даних для навчання, можливе «перетренування» моделі, не підходить для нелінійних випадків
Наївний Баєс	Швидкий класифікатор, сходяться раніше, ніж дискримінаційні моделі, такі як логістична регресія, вимагає меншої підготовки, застосовується як до бінарних, так і до багатокласних задач	Неможливо досягти взаємодії між ознаками. Розраховані ймовірності лише відносно точні.
SVM	Параметри регуляризації допомагають уникнути проблеми «перетрненування», дозволяють використовувати ядрові методи для того щоб включати експертні знання	Вибір правильного ядра й тренування моделі може займати багато часу та зусиль. Результати складніше інтерпретувати, ніж для регресії.
Нейронні мережі	Може узагальнити майже будь-які функції чи випадки, близькі по принципу роботи до людського мозку	Вимагає великих об'ємів навчальних та тестових вибірок, багато операцій приховані, складно покращати точність

Завдяки активному використанню нейромережевих алгоритмів і створення нової архітектури, що підходять для вилучення сенсу з текстів, область NLP дуже швидко розвивається. Вчені зробили великий прогрес в області генерації текстів, машинного перекладу і вилучення інформації, навіть вираженої неявно. Однак, навіть простіші моделі сьогодні працюють ефективніше ніж раніше завдяки покращеним методам обробки текстів, тому має сенс враховувати й їх при побудові моделі для класифікації текстів.

2.7. Висновки до другого розділу

В цьому розділі ми порівняли різні методи класифікації та представлення даних для задач класифікації текстів. У ході дослідження було виявлено, що логістична регресія добре підходить для класифікації текстів, при правильному підборі параметрів регресія може бути непоганою альтернативою іншим більш складним методам, тому що її результати простіші для інтерпретації й вона менш вимоглива до об’ємів тренувальних та навчальних вибірок, ніж глибинні нейронні моделі.

У порівнянні з іншими лінійними класифікаторами такими як Баєс та SVM, логістична регресія краще підходить для задач класифікації з двох причин. По-перше, логістична регресія краще працює з великими об’ємами ніж Баєс, у неї також вище поріг навчання. По-друге, хоча SVM може давати трохи точніші результати, але логістична регресія на відміну від SVM дає ймовірнісну відповідь, у випадку класифікації текстів це корисно тим, що у тексті можуть бути декілька взаємопов’язаних тем, тож у цьому випадку доречніше віднести до декількох категорій.

РОЗДІЛ 3
ВИКОРИСТАННЯ ЛОГІСТИЧНОЇ РЕГРЕСІЇ ДЛЯ КЛАСИФІКАЦІЇ ТЕКСТІВ У ПРАКТИЧНИХ ЗАВДАННЯХ

3.1. Застосування модифікованої логістичної регресії на прикладі додатку-асистента для освітніх цілей

Людині інтелектуальної праці загалом завжди було важливо постійно покращувати свої навички й поновлювати знання у своїй професійній чи культурній області. Сьогодні завдяки здобуткам постіндустріальної епохи, в якій ми живемо, ми можемо отримувати нову інформацію й навчатись буквально на ходу.

Однак, вочевидь й ця епоха кидає свої виклики, один з них це переповненість інформаційного середовища контентом різного типу та якості, тож для інтернет-користувача може бути важко сконцентруватись та опанувати одну чи декілька тем. Особливо явно ця проблема постала у часи глобальної пандемії, коли бізнес, розваги й, що важливіше, навчання перейшло у дистанційний формат.

Отже, було вирішено, що має сенс розробити додаток, який зможе допомогти користувачу нормувати й розподіляти час на різні теми відповідно до своїх пріоритетів. Реалізацію даної ідеї може бути бот-асистент, який контролює прогрес навчання користувача, класифікує тексти, які він отримує від користувача й рекомендує наступні статті для ознайомлення по запиту, базуючись на пріоритетах й прогресі користувача.

Додаток-асистент для навчання – це одна з невеликих новинок, яку потребує наш час, до того ж він є наочною демонстрацію технік й методів розпізнавання текстів у практичних задачах. Окрім цього, оскільки через додаток буде проходити багато статей та інших видів текстових джерел, то у нього буде потенціал для росту й покращення показників.

3.1.1. Бізнес-вимоги до додатку

Перед тим як починати розробку будь-якого додатку, потрібно визначатись з бізнес-вимогами для того щоби раціонально підібрати технології для реалізації ідеї. Виходячи з нашої ідеї, можна встановити наступні початкові вимоги для бота:

1. Додаток повинен вірно розпізнавати статті на англійській мові, що належать до 4 тем математика, IT, економіка та історія. Такі класи були обрані для випробовування ідеї, так як ці теми істотно відрізняються, при цьому у деяких статтях вони можуть переплітатись (наприклад IT та математика), тож це непогане випробовування для класифікатора. Англійська мова була обрана, тому що вона найрозповсюдженіша у світі, тож й інформаційних джерел на ній більше всього.

2. Потрібно мати можливість швидко та зручно випробовувати модель у реальних умовах.

3. Потрібно подбати про безпеку, конфіденційність й цілісність даних користувачів.

4. Бажано щоб розробка клієнтською частини займала якнайменше часу, щоб мати можливість приділити більше уваги й часу самій моделі класифікації.

5. Бажано мати можливість швидко набрати клієнтську базу, щоб отримати відгуки щодо якості класифікації текстів, окрім цього, велика кількість клієнтів допоможе зібрати більше текстових джерел, проаналізувати їх, знайти недоліки у моделі й покращити продукт.

З вище перелічених пунктів, випливає висновок, що більш за все для наших цілей підходить реалізація додатку у формі бота. Зараз надзвичайно популярні месенджери, які дозволяють розміщати на своїх платформі ботів для будь-яких цілей. Таким чином, ми вирішуємо декілька проблем одночасно: по-перше, аспекти авторизації й безпеки в основному будуть у полі відповідальності стороннього додатку, по-друге, клієнтську базу можна бути знайти значно легше завдяки популярності таких рішень, по-третє, нам не потрібно витрачати багато

час на клієнтську частину. Усе це вкупі, дозволяє нам витратити більше часу на оптимізацію й тестування, власне, моделі класифікації - основи нашого додатку.

Визначившись з формою додатку, нам потрібно обрати платформу на якій буде працювати наш бот. Після недовгих пошуків, було вирішено використати Telegram для розміщення бота, оскільки він популярний, має безкоштовне, зручне, добре задокументоване API та бібліотеки під нього, тож запустити у цьому месенджері бота нічного не коштує й разом з тим порівняно просто.

3.2. Вибір вхідних ознак для моделі класифікації

Спочатку перед навчанням обирається параметр за яким будуть класифікуватись тексти, тобто як було зазначено вище, потрібно перевести текстову інформацію у числову форму.

Як ми описали у попередніх розділах сьогодні загалом є 2 великі групи моделей представлення даних текстової інформації: статистичні моделі та семантичні вектори, які отримуються за допомогою нейромереж «без вчителя».

Семантичні вектори – це найбільш досконала та розвинута сьогодні форма представлення текстових даних, адже дозволяє вгадувати значення нових слів й враховувати контекст.

Проте, використання показника TF-IDF для завдань класифікації теж прийнятне, так як він дає можливість оцінити важливість слів для кожної теми окремо, що у свою чергу важливо для коректної класифікації текстів. Використання TF-IDF зменшує вагу широковживаних слів й збільшує вагу для тих слів, що характерні лише для окремого класу документів, що й задовольняє потребам задачі класифікації.

У результаті досліджень було вирішено застосувати показник TF-IDF, так як досвід розробників показує, що моделі з використанням показника TF-IDF дають більшу точність ніж з word2vec [36].

Нижче продемонстровано точність логістичної регресії з використанням TF-IDF (рис 3.1) та word2vec (рис 3.2) від Google, що була навчена на корпусі

текстів з новин розміром 300 більйонів слів. Класифікатор випробовується на розміченому датасеті з 40000 текстів з програмування з ресурсу stack overflow:

```
accuracy 0.783
               precision    recall  f1-score   support

         java       0.70      0.62      0.66       613
         html       0.91      0.91      0.91       620
      asp.net       0.97      0.94      0.95       587
           c#       0.78      0.77      0.78       586
ruby-on-rails       0.77      0.81      0.79       599
       jquery       0.59      0.58      0.58       589
        mysql       0.77      0.76      0.76       594
          php       0.82      0.86      0.84       610
          ios       0.70      0.72      0.71       617
   javascript       0.61      0.59      0.60       587
       python       0.64      0.63      0.64       611
            c       0.83      0.83      0.83       594
          css       0.78      0.78      0.78       619
      android       0.85      0.85      0.85       574
       iphone       0.80      0.83      0.81       584
          sql       0.65      0.65      0.65       578
  objective-c       0.82      0.84      0.83       591
          c++       0.91      0.91      0.91       608
    angularjs       0.96      0.94      0.95       638
         .net       0.78      0.83      0.80       601

  avg / total       0.78      0.78      0.78     12000

Wall time: 981 ms
```

Рис. 3.1. Класифікація методом логістичної регресії з TF-IDF

```
accuracy 0.6379166666666667
               precision    recall  f1-score   support

         java       0.63      0.59      0.61       613
         html       0.73      0.76      0.75       620
      asp.net       0.65      0.67      0.66       587
           c#       0.53      0.52      0.52       586
ruby-on-rails       0.70      0.77      0.73       599
       jquery       0.44      0.39      0.41       589
        mysql       0.65      0.61      0.63       594
          php       0.73      0.80      0.76       610
          ios       0.60      0.61      0.61       617
   javascript       0.56      0.52      0.54       587
       python       0.55      0.50      0.52       611
            c       0.61      0.61      0.61       594
          css       0.65      0.65      0.65       619
      android       0.60      0.57      0.59       574
       iphone       0.70      0.71      0.71       584
          sql       0.42      0.42      0.42       578
  objective-c       0.68      0.71      0.70       591
          c++       0.76      0.78      0.77       608
    angularjs       0.82      0.83      0.82       638
         .net       0.66      0.71      0.68       601

  avg / total       0.63      0.64      0.64     12000
```

Рис. 3.2. Класифікація методом логістичної регресії з word2vec

Як бачимо з результатів випробувань, класифікатор з використанням TF-IDF дає точність 78% проти 63% з навченою word2vec моделлю. Отже, word2vec не є універсальним рішення.

Загальна word2vec модель потребує налаштування та доопрацювання під конкретні категорії текстів, тому для задачі класифікації було вирішено використовувати TF-IDF як більш простий, швидкий й водночас ефективний спосіб представлення даних для задач класифікації

3.3. Обґрунтування використання методу логістичної регресії у боті-асистенті

Для класифікації текстів було обрано метод логістичної регресії, так як цей метод не потребує багато часу й об'ємів даних для навчання та результати цього методу легше для інтерпретації в порівнянні з нейромережами.

Якщо порівнювати логістичну регресію з іншими лінійними класифікаторами, то регресія показує загалом кращі результати при класифікації текстів, ніж метод наївного Баєса завдяки своїй дикскримінативній припроді. SVM моделі мають вищу точність, ніж регресія, проте вони не дають ймовірнісної відповіді, а просто відносять текст до одної з категорій. Однак, частіше за все у текстах буває декілька взаємопов'язаних тем, тож класифікація у цьому випадку менш очевидна й правильним рішенням буде віднести текст одразу до декількох категорій.

Отже, з усіх можливих варіантів логістична регресія як легка та невимоглива модель краще за всіх інших підходить для класифікації текстів.

3.4. Процес розробки навчального асистента

Для створення бота-асистента й моделі для класифікації було вирішено використати мову програмування Python 3 (версії 3.7.5), так як дана мова

пропонує велику кількість спеціалізованих бібліотек для роботи з великими даними, машинним навчанням та обробки природної мови.

У процесі розробки були використані наступні пакети: фреймворк flask для створеннсерверних програм, бібліотека для машинного навчання scikit-learn, набор бібліотек NLTK (Natural Language Toolkit) для роботи з природніми мовами, бібліотека BeautifulSoup для парсингу сайтів, requests для завантаження змісту сайтів для їх подальшої обробки та python-telegram-bot для зручної інтеграції з месенджером. Для зберігання інформації про статті, користувачів та їх статистику було використано базу данних PostgreSQL

3.4.1. Процес навчання класифікатора

Перший крок у розробці моделі-класифікатора після обрання моделі та ознак для класифікації - це створення навчальної вибірки. Як основне джерело навчальних даних було обрано інтернет енциклопедію Wikipedia, так як вона має широкий вибір наукових текстів, які впорядковані за темами, що облегшує пошук й відбір даних для навчання. Окрім цього, усі статті структуровані, що спрощує їх обробку та отримання з них корпусу слів.

За допомогою бібліотек для запитів й парсингу request та BeautifulSoup було зібрано та оброблено 100 статей, у процесі їх обробки були видалено математичні формули, зображення й HTML теги. Усі 100 були рівномірно розподілені між 4 темами, а саме економікою, історією, математикою та IT. Загальний розмір вибірки становить приблизно 1000000 слів.

Статистичний показник TF-IDF не враховує контекст, синтаксичні особливості англійської мови й пунктуацію, тому після того як були отримані статті за посиланнями, кожна з них була оброблена наступним чином:

1. Всі букви були приведені до нижнього реєстру.
2. Були Прибрані усі числа та окремі букви з тексту.
3. Були видалені знаки пунктуації.
4. Всі множинні пробіли були зведені до одинарних.

5. Всі слова в тексті були лемматизовані, тобто зведені до словникової форми, за допомогою бібліотеки NLTK для обробки природніх мов.

Після цього вся навчальна вибірка за допомогою бібліотеки ScikitLearn переводиться у TF-IDF наступним чином:

```
from sklearn.feature_extraction.text import TfidfTransformer, TfidfVectorizer
tfidfconverter = TfidfVectorizer(max_features=3500, min_df=5, max_df=0.8, stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(documents).toarray()
```

TfidfVectorizer задає параметри для TF-IDF конвертора, у даному випадку max_features встановлює максимальну кількість термінів 3500, stop_words вилучає усі шумові слова англійської мови, min_df встановлює обмеження, що термін повинен зустрічатись мінімум у 5 текстах, max_df виключає слова, що трапляються більше ніж у 80% документів

Далі навчальна вибірка розбивається на 2 частини – навчальну й тестову, навчальна частина становить 70 текстів, а тестова – 30 відповідно, це потрібно для того щоб перевірити точність моделі.

Далі модель навчається:

```
classifier = LogisticRegression(verbose=1, solver='liblinear',random_state=0, C=5, penalty='l2',max_iter=1000)
classifier.fit(X_train, y_train)
```

LogistiRegression задає параметри для логістичної регресії, solver встановлює алгоритм зменшення втрат функції, C встановлює ступінь регулярізації моделі, penalty вставновлює метод регулярізації, penalty встановлює тип регулярізації l2, max_iter обмежує кількість кроків градієнтного спаду до 1000.

Через те, що було обрано liblinear метод ця модель не працює як класична багатокласова регресія типу softmax, замість цього при класифікації використовується алгоритм «один проти всіх». Таким чином, для кожного класу окремо вираховується ймовірність, що текст до нього належить й у результаті обирається клас з найбільшою вірогідністю.

L2 – це тип регулярізації, що застосовується під час навчання моделі й потрібен для того, щоб логістична регресія уникнула явище «перетренування». Даний параметр записується таким чином:

$$R(\theta) = \|\theta\|_2^2 = \sum_{j=1}^{m} \theta_j^2 \tag{3.1}$$

Після застосування тренувальна функція приймає наступний вигляд:

$$\bar{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(y^{(i)}, x^{(i)}; \theta\right) - C \sum_{j=1}^{m} \theta_j^2 \tag{3.2}$$

Таким чином, цей параметр дозволяє моделі вірно класифікувати тексти, яких не було у навчальних вибірках.

Після навчання модель класифікує тестовий набір й видає точність класифікації на тестовому наборі розміром 30 текстів (рис. 3.3).

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	0.88	1.00	0.93	7
2	1.00	0.90	0.95	10
3	1.00	1.00	1.00	4
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Рис 3.3. Результат навчання логістичної регресії

Дана модель має загальну точність 97%, що задовольняє нашим вимогам й дозволяє нам використовувати її у боті-асистенті.

3.4.2. Архітектура та принцип роботи навчального асистента

З вище наведених причин, у нашій роботі було вирішено розробити навчального асистента у вигляді бота. Бот – це серверний додаток, що працює з клієнтською частиною через API (прикладний програмний інтерфейс), клієнтською частиною у цьому випадку виступає месенджер (Telegram), що приймає текст від користувача, передає його боту та виводить відповіді сервера на запити.

Вся бізнес-логіка бота виконується на стороні сервера (рис. 3.4), він опрацьовує запити та відповідає на них текстовими повідомленнями, зберігає статистику користувача у базі даних, розпізнає та рекомендує статті.

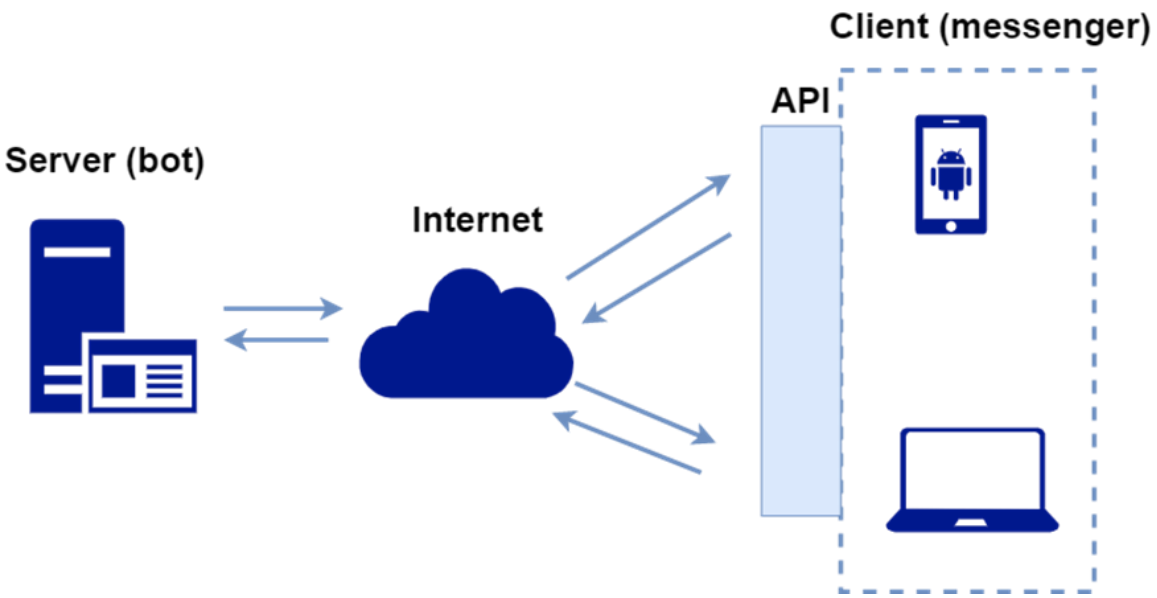

Рис 3.4. Загальна структура бота

Сервер взаємодіє з клієнтом через Rest API за допомогою POST запитів, в яких вказується id чату та користувача. Для інтеграції з месенджером використовується бібліотека-обгортка python-telegram-bot, яка надає ряд функцій для простої інтеграції з API Telegram.

Серверний додаток складається з 3 модулів:

1. classification_module.py – модуль, який виконує класифікацію статей, цей модуль використовує навчену модель логістичної регресії, яка збережена у файлі 'text_classifier'. Цей модуль приймає масив посилань на статті й віддає масив кортежів, які мають форму (посилання, клас, розмір).

2. database_module.py – модуль, який відповідає за взаємодію з БД, реєструє користувача та оновлює його статистику, список статей.

3. app.py – головний модуль, цей же модуль запускається як основна програма на платформі Heroku. Він об’єднує в собі всі інші модулі, встановлює з’єднання з базою даних, взаємодіє з сервером Telegram-а й вираховує рейтинг кожної теми, щоб згідно з пріоритетами користувача та його загальним прогресом визначити тему, якій йому потрібно приділити увагу. Вираховується рейтинг теми наступним чином:

$$M_i = \frac{k_i R_i \left(1 - p_i\right)}{\max\left(R\right)}, \tag{3.3}$$

де

k_i - це коефіцієнт теми, він може приймати значення від 1 до 4, 1 для найменш важливих тем й 4 для найбільш;

R_i – це загальний розмір теми (сума кількості слів всіх текстів цієї категорії) для користувача;

p_i – це частка прочитаної текстової інформації, можна виразити як відношення прочитаних слів n_i до загального розміру теми R_i;

max(R) – це розмір найбільшої теми з усіх 4.

Після того як рейтинги для кожної категорії текстів вирахувано, бот рекомендує одну випадкову ще не прочитану повністю статтю, яка відноситься до класу з найбільшим рейтингом.

Telegram пропонує 2 методи отримання інформації про нові повідомлення: polling (періодичне опитування сервера Telegram) та webhook (сервер Telegram сповіщає сервер бота сам). У цій роботі було обрано метод webhook; все що для

нього потребується – це розгорнутий веб-сервер на публічному домені або відкритий порт на власному ПК до якого може звертатись сервер месенджера. Для локального тестування було використано програму ngrok, а фінальну версію було розміщено на платформі Heroku.

Для того щоб Telegram сповіщав сервер потрібно зареєструвати його для Telegram, робиться це за допомогою GET запиту за адресою https://api.telegram.org/bot{Token}/setWebhook?url={Token/URL}, де Token - це унікальний ідентифікатор бота, який надається месенджером при створенні бота, а URL - це публічна адреса сервера боту.

На рис. 3.5 продемонстровано алгоритм взаємодії навчального бота з БД у вигляді Sequence діаграми.

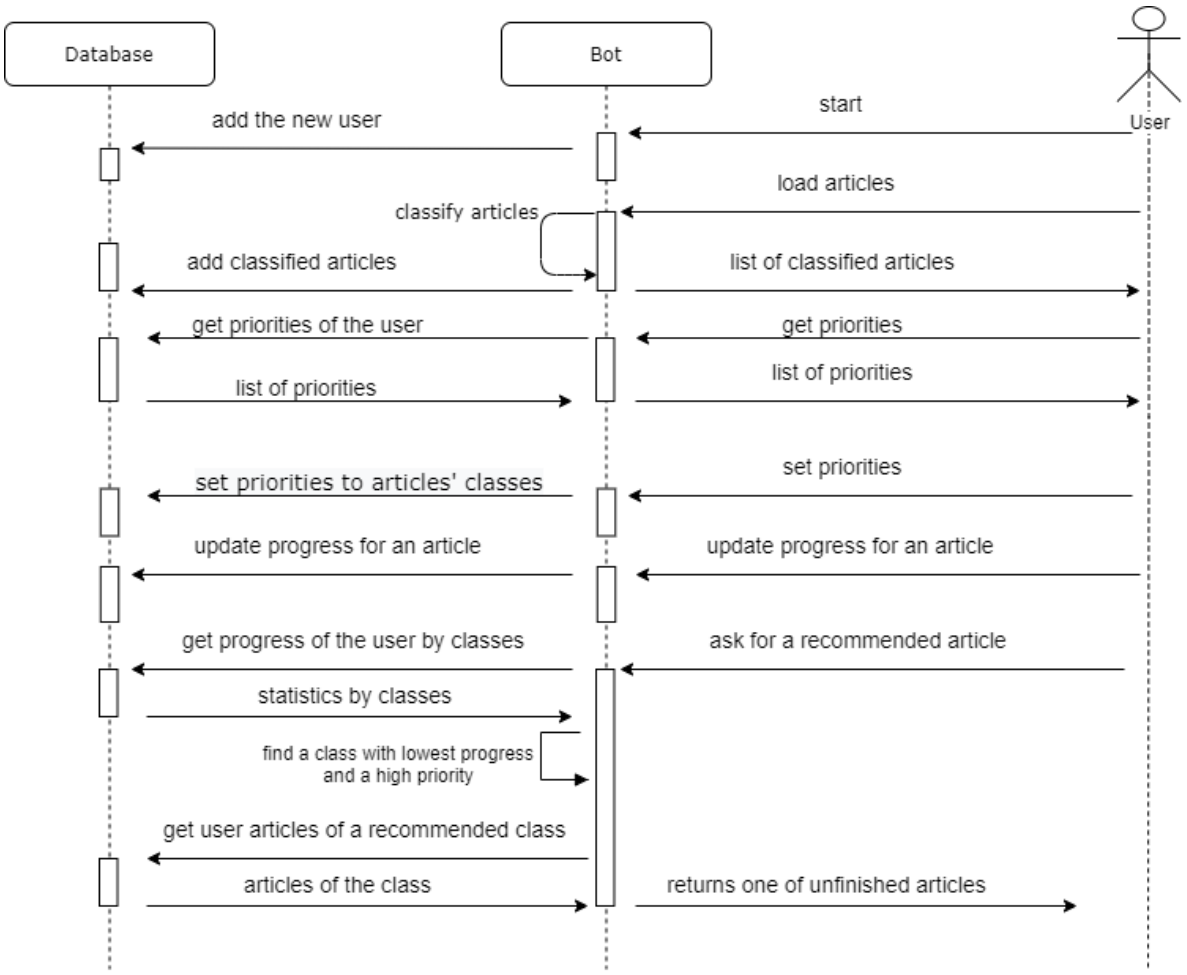

Рис. 3.5. Sequence діаграма бота

Бот використовує 4 таблиці БД для збереження даних користувачів, її структура продемонстрована на рис. 3.6.

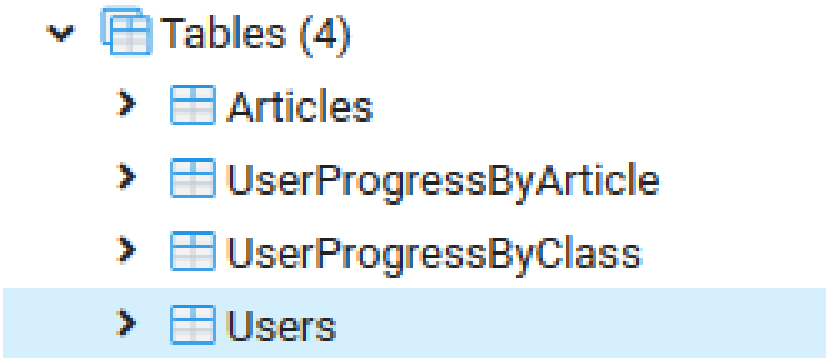

Рис 3.6. Загальна структура БД

1. Users – таблиця користувачів (рис 3.6), зберігає id користувачів, але може бути розширена, на неї посилаються усі інші таблиці. Кожного разу коли новий користувач активує бота, тобто надсилає повідомлення start, його ID додається до таблиці.

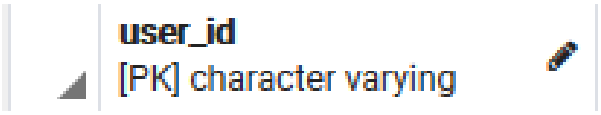

Рис. 3.7. таблиця Users

2. Articles - таблиця для усіх статей, що були завантажені до бота коли-небудь. Вона зберігає посилання на статті, розмір статей та їх клас, параметр (link) є й унікальним ключом. Коли користувач завантажує статтю за допомогою команди load, то вона додається до таблиці. Якщо стаття, що завантажується, вже є у цій таблиці, то вона не класифікується заново – замість цього вона просто зберігається до користувача у статистику до таблиці UserProgressByArticle.

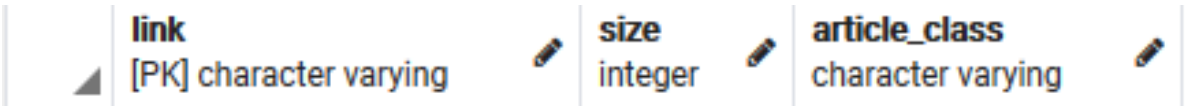

Рис. 3.8. таблиця Articles

3. UserProgressByArticle – таблиця для прогресу користувачів за статтями (рис 3.8). Вона зберігає посилання на статті (article_link) , id користувача та його прогрес за статтями (article_progress) у відсотковому співвідношенні (від 0 до 1.0), цей показник змінюється кожен раз, коли користувач оновлює свій прогрес за допомогою команди /update-progress. Ця таблиця оновлюється, коли користувач завантажує нову статтю /update-articles.

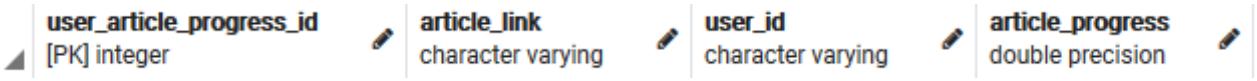

Рис. 3.9. таблиця UserProgressByArticle

4. UserProgressByClass – таблиця для зберігання прогресу користувача за темами (рис. 3.10). Вона зберігає id користувача, прогрес користувача за темою (class_progress) у відсотковому співвідношенні (від 0 до 1.0), пріоритет класу (user_class_priority) від 1 до 4, де 1 – це найвищий пріоритет та розмір класу (class_size) для користувача.

Розмір класу формується як сума розмірів усіх статей, що користувач читає, кожен раз коли користувач додає нові статті командою /load, цей показник росте.

Прогрес користувача за темою також перераховується кожного раз, у коли користувач оновлює свій прогрес хоча б для одної зі статей або коли завантажуються нові статті.

UserProgressByClass бере участь у всіх процесах, що відбуваються у боті, так як додання нового користувача, зміна прогресу для одної зі статей, зміна пріоритетів або додання нової статті веде до оновлення цієї таблиці.

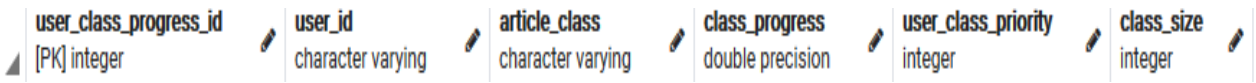

Рис. 3.10. таблиця UserProgressByClass

3.4.3. Результати роботи бота-асистента

Алгоритм роботи бота та його взаємодії з користувачем наступний:

1. Користувач активує бота, ID користувача заноситься до таблиці Users.

2. Користувач за допомогою команди /load завантажує список бажаних статей, у нашому випадку це 4 статті з невідомого для моделі джерела про клан Токугава, обробку природньої мови, ринкову енокоміку та метод зворотного поширення помилки, ці тексти відповідають всім наявним категоріям. Кожний текст проходить ту ж саму процедуру обробки, що й тексти для навчальної вибірки, себто видаляються шумові слова, всі слова лемматизуються тощо. На рис 3.11 видно, що бот вірно класифікував всі статті.

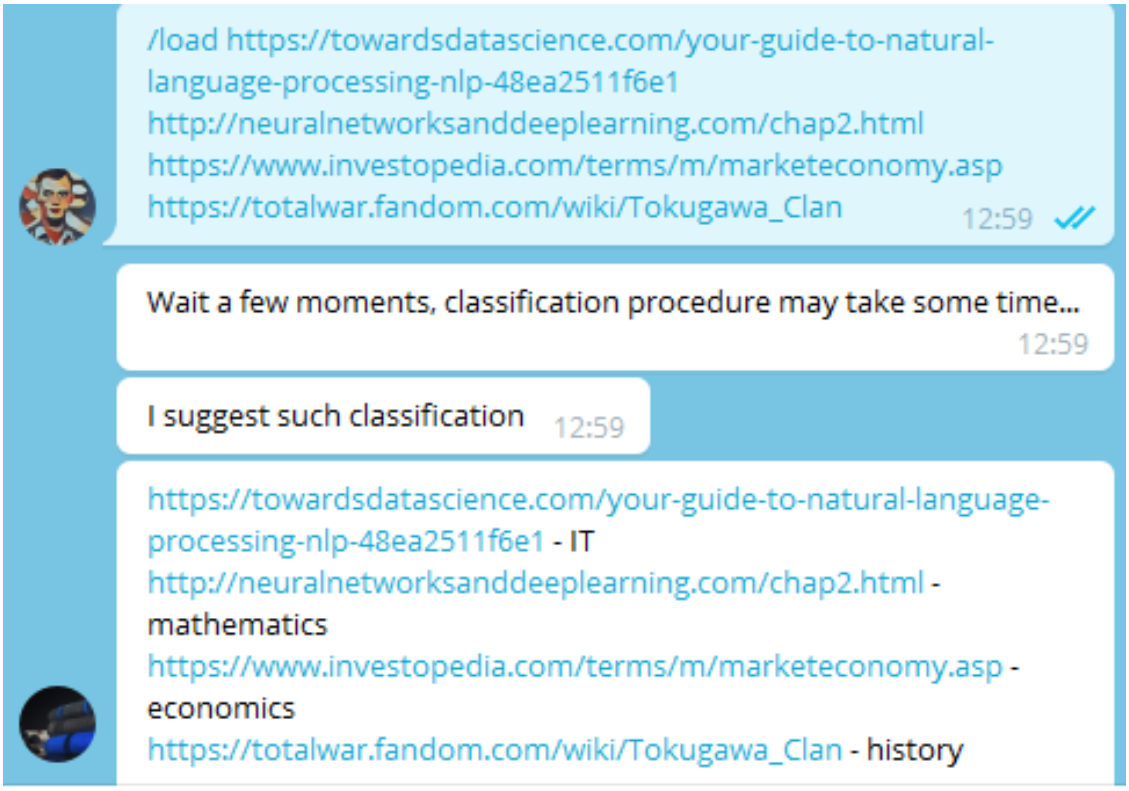

Рис. 3.11. Класифікація ботом 4 статей

3. Користувач проглядає свої пріоритети, що встановлені за замовчуванням (рис 3.12).

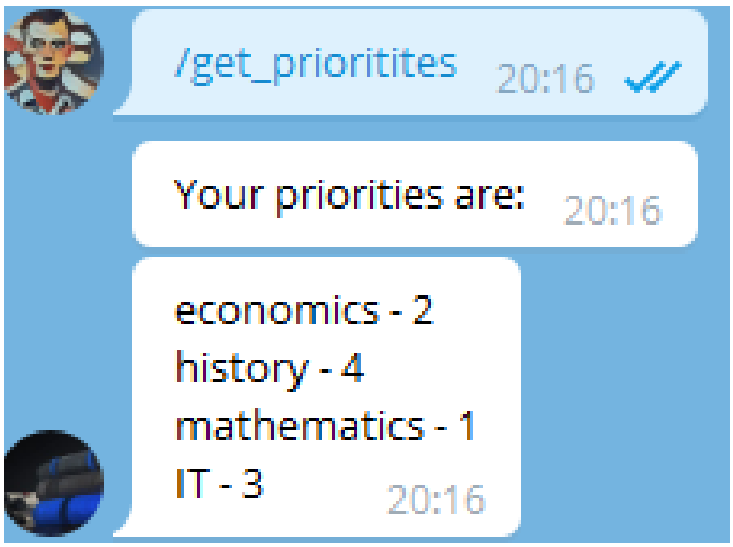

Рис. 3.12. Пріоритети користувача

3. Користувач запитує бота, яку статтю йому рекомендується прочитати. Як було зазначено раніше, бот рекомендує статтю залежно від пріоритету категорії текстів й обсягу непрочитаних слів у цій категорії.

На рис. 3.13 бот пропонує користувачу прочитати прочитати статтю з математики, тому що вона має найвищий пріоритет й великий обсяг.

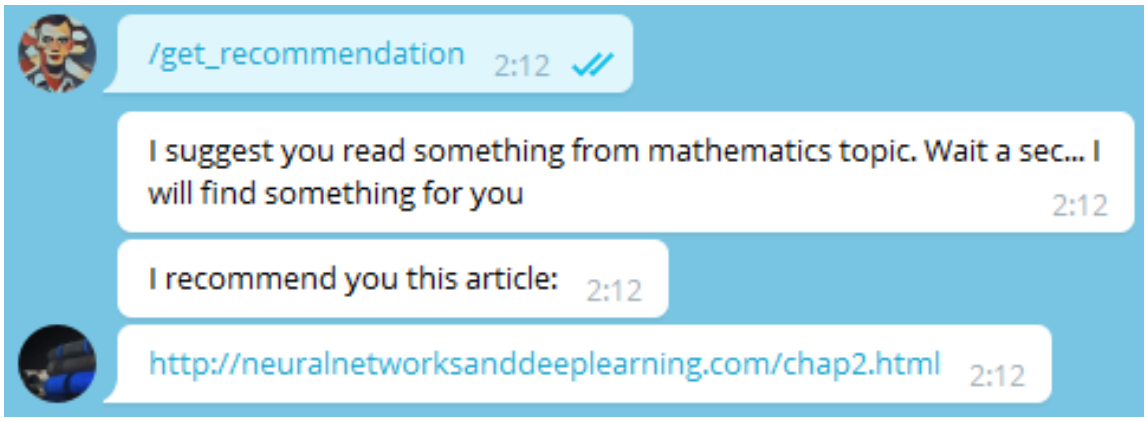

Рис. 3.13. Користувач отримує рекомендацію від бота

4. Якщо користувача все задовольняє, то він може встановити прогрес (рис. 3.14) від 0 до 100 у процентах для вже прочитаної статті. Наприклад, тут ми встановлюємо для єдиної статті з математики прогрес 100, що значить, що вона повністю прочитана користувачем.

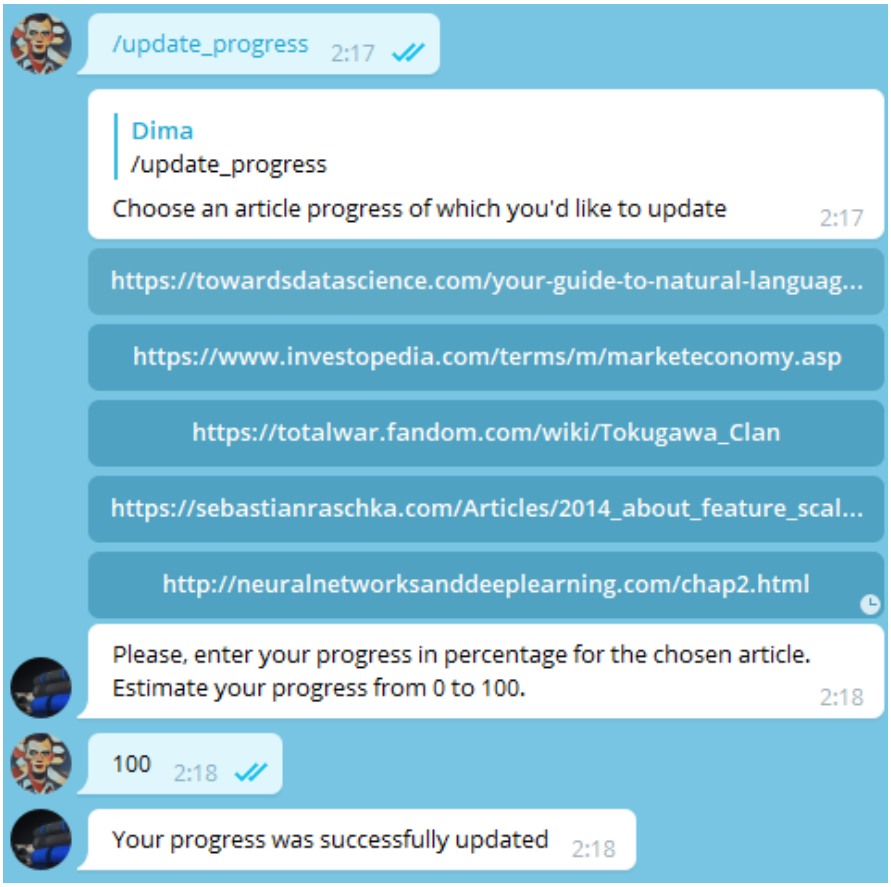

Рис 3.14. Оновлення прогресу статті користувачем

5. Після цього користувач знову запитує у бота рекомендацію (рис. 3.15) й отримує відповідь, що йому пропонується прочитати статтю з IT теми. Хоча IT має 3 пріоритет, тобто нижчий ніж у економіки, в цій категорії вже 2 статті й вони великого розміру, тому бот пропонує користовачу цю тему.

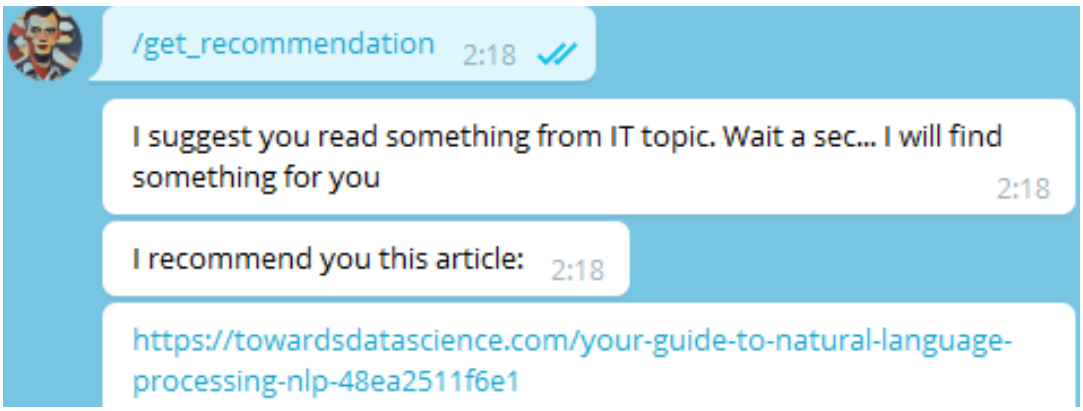

Рис 3.15. Користувач повторно отримує рекомендацію

Проведене випробовування продемонструвало, що логістична регресія комбінована з методом ранжування може успішно використовуватись у практичних завданнях, таких як створення рекомендаційних систем (у нашому випадку це був навчальний асистент).

З цього можна зробити висновок, що навіть сьогодні метод логістичної регресії можна успішно використовувати й розглядати як цілком життєздатну альтернативу іншим більш складним методам класифікації текстів.

3.5. Висновки до третього розділу

Метою даного розділу було обґрунтувати та продемонструвати життєздатність логістичної регресії, поєднаної з методом ранжування, у практичних задачах на прикладі навчального бота-асистента.

В даному розділі було детально продемонстровано підхід до розробки NLP додатків та аспекти, які розробнику слід приймати до уваги при проектуванні NLP додатку та виборі ознак й методу класифікації. Наприклад, моделі для створення векторів слів, такі як word2vec, не потребують спеціальної обробки навчальних та вхідних текстів й хоча на сьогоднішній день вони є найбільш досконалими, вони не завжди підходять для задач класифікації, тому є сенс розглядати більш прості традиційні підходи як TF-IDF.

В ході розробки було розглянуто декілька методів класифікації й представлення текстових даних й виявлено, що навчена word2vec модель, яка не концентрується на окремих темах чи класах, гірше класифікує тексти одної й тої ж тестової вибірки, ніж TF-IDF модель.

TF-IDF модель на відміну від векторів слів більш обмежена у своєму використанні й потребує попередньої обробки текстів, але перший фактор не є суттєвим для задач класифікації, а другий фактор нівелюється завдяки спеціалізованим бібліотекам, що вдало й швидко оброблюють тексти та виключають зайву інформацію.

В результаті, було зроблено висновок, що логістична регресія може успішно застосовуватись для класифікації наукових текстів, якщо вона використовується разом з TF-IDF ознакою. Незважаючи на свою математичну простоту, цей метод досить точно класифікував тексти з невідомих для нього джерел навіть з невеликим об’ємом навчальних даних завдяки правильно підібраним параметрам та ознакам.

РОЗДІЛ 4
ЕКОНОМІКА

4.1. Визначення трудомісткості розробки програмного забезпечення

Одним з важливих завдань для підприємця або керівника проекту є визначення трудомісткості та витрат на створення програмного продукту. В цьому розділі ми наведемо приклад розрахунку витрат на розробку навчального асистента з використанням логістичної регресії як основного методу для класифікації текстових джерел інформації.

Вхідні дані:

годинна заробітна плата програміста, грн / год — 300;

вартість машино-години ЕОМ, грн / год — 50.

Нормування роботи в процесі створення ПЗ істотно ускладнюється в силу творчого характеру роботи програміста, тому трудомісткість розробки ПЗ може бути розрахована на основі системних моделей з різною точністю оцінки.

Трудомісткість розраховується за формулою (у людино-годинах):

$$t = t_u + t_a + t_n + t_{відл} + t_д + t_o, \qquad (4.1)$$

де

t_o – витрати праці на підготовку і опис поставленого завдання (50 год.);

t_u – витрати праці на дослідження алгоритму вирішення задач;

t_a – витрати праці на розробку блок-схем алгоритму;

t_n – витрати праці на програмування за готовим блок-схемою;

$t_{відл}$ – витрати праці на відладку програм на ПЕОМ;

$t_д$ -– витрати праці на підготовку документації.

Сукупні витрати праці визначаються виходячи з умовного числа операторів у розроблюваному ПЗ.

Усі витрати рахуються у людино-годинах

Розрахунок очікуваних витрат праці:

$$Q = q \cdot C \cdot (1 + p), \tag{4.2}$$

де

q – передбачуване число операторів;

c – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

Приймемо, що q – 1400; c –1,19 тому що була використана мова високого рівня й в області NLP вже було достатньо напрацювань; p – 0,1.

$$Q = 1400 \cdot 1{,}19 \cdot (1 + 0{,}1) = 1832{,}6, \tag{4.3}$$

Витрати праці на вивчення опису завдання визначаються з урахуванням уточнення опису і кваліфікації програміста за формулою:

$$t_u = \frac{QB}{(75...85)K}, \tag{4.4}$$

де

B – коефіцієнт збільшення витрат праці, що залежить від неточності опису, доробок тошо.

K – Коефіцієнт кваліфікації програміста, який визначається в залежності від стажу роботи за фахом.

Внаслідок неповного опису завдання та коригувань B приймається 1,5, а K = 1,1, тому що стаж роботи від 3 до 5 років;

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{1832,6 \cdot 1,5}{75 \cdot 1,1} = 33,32 \quad (4.5)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} = \frac{1832,6}{23 \cdot 1,1} \approx 72 \quad (4.6)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} = \frac{1832,6}{23 \cdot 1,1} \approx 72 \quad (4.7)$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{відл} = \frac{Q}{(4...5)K} = \frac{1832,6}{4 \cdot 1,1} = 416,5 \quad (4.8)$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до} \quad (4.9)$$

де

$t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)K} = \frac{1832,6}{17 \cdot 1,1} \approx 98 \quad (4.10)$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації.

$$t_{до} = 0,75 \cdot t_{др} = 0,75 \cdot 98 = 73,5 \quad (4.11)$$

Отже, витрати праці на підготовку документації $t_д$ = 98 + 73,5 = 171,5

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 33,32 + 72 + 72 + 416,5 + 171,5 = 815,32 \quad (4.10)$$

4.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення програмного забезпечення $К_{пз}$ включають витрати на заробітну плату розробників програми ($З_{зп}$) і витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$).

$$К_{по} = З_{зп} + З_{мв} \quad (4.12)$$

Витрати на заробітну плату визначаються за формулою:

$$З_{зп} = t \cdot С_{пр}, \quad (4.13)$$

де

t – загальна трудомісткість розробки ПЗ;

$С_{пр}$ – середня годинна заробітна плата програміста (300);

$$З_{зп} = 815,32 \cdot 300 = 244596 \text{ грн} \quad (4.14)$$

Витрати машинного часу, необхідного для налагодження програми на ЕОМ (Змв) визначаються за формулою:

$$З_{мв} = t_{відл} \cdot С_{мч}, \quad (4.15)$$

де

$t_{відл}$ – трудомісткість налагодження програми на ЕОМ;

$С_{мч}$ – вартість машино-години ЕОМ.

$$З_{мв} = 416{,}5 \cdot 40 = 16660 \text{ грн}, \quad (4.16)$$

Таким чином витрати на створення програмного забезпечення, складуть:

$$К_{по} = З_{зп} + З_{мв} = 16660 + 244596 = 261256 \text{ грн} \quad (4.17)$$

Очікувана тривалість розробки ПО:

$$T = \frac{t}{B_k \cdot F_p}, \quad (4.18)$$

де

Вк – число розробників;

Fp– місячний фонд робочого часу, цей показник становить при 40-ка годинному робочому тижні Fp = 176 годин.

$$T = \frac{815{,}32}{1 \cdot 176} \approx 4.6 \text{ місяців} \quad (4.19)$$

4.3. Маркетингові дослідження

В процесі дослідження технік та методів класифікації наукових текстів було створенно навчального бота-асистента для особистого використання. Метою навчального асистента є покращення навчального процес користувача та ефективний розподіл його часу на засвоєння обраних тем відповідно до пріоритетів.

На разі навчальний асистент може працювати лише з англомовними джерелами з 4 тем: економіка, IT, історія та математика. В перспективі перелік тем, які бот може розпізнавати та його можливості можуть бути розширені.

Принцип взаємодії користувача з ботом наступний:

1. Користувач завантажує список посилань на статті до бота, бот класифікує кожну статтю за темою.

2. Користувач встановлює пріоритети тем.

3. Користувач запитує бота якій темі потрібно приділити увагу й отримує від нього статтю, яку рекомендується прочитати.

4. Коли користувач прочитує статтю, то він повідомляє боту яку статтю та яку її частку у процентному відношенню він прочитав.

5. Коли користувач знову потребує рекомендації, то він повторно звертається до боту.

Як результат, навчальний асистент зменшує час, що витрачається користувачем на вибір статті та планування навчання. Таким чином, асистент пришвидшує та покращує навчальний процесс користувача.

Навчальний асистент було розроблено у вигляді бота для Telgegram месенджера, тож він не розповсюджується у вигляді комп'ютерного чи мобільного додатку, що встановлюється на пристрій. Бот – це серверний додаток, тому доречніше аналізувати витрати та спосіб надання послуг як для SaaS (Software as a service – програмне забезпечення як послуга) додатку. Отже, у нашому випадку витрати на обслуговування программного забезпечення – це вартість оренди хостингу та обчислювальних потужностей для додатку.

Зараз навчальний асистент не потребує ніяких витрат на своє обслуговування, так як він розгорнутий по безкоштовному плану на платформі Heroku. Безкоштовний план призначен для особистих малих проектів, протев він має ряд обмежемень (https://www.heroku.com/pricing). Наприклад, він надає порівняно мало обчислювальної потужності й дозволяє працювати сервісу не більше ніж 550 годин на місяць, тобто не більше 23 повних днів, також додаток відключається кожні 30 хвилин, коли не отримує запитів, отже цей план для робочого етапу нам не підходить, тому нам потрібно розглянути інші плани.

Спочатку, якщо припустити що у нашого бота буде аудиторія не більше 100 постійних користувачів, то нам підійде початковий план, який коштує 7

доларів (приблизно 210 грн.) на місяць. Початковий план пропонує такі ж характеристики, що й безкоштовний, але при цьому дешевий, здатен обслуговувати відносно невелику кількість запитів від 100 користувачів, не має обмежень у часі й не відключається кожні 30 хвилин, тож наш додаток буде доступний протягом всього місяця. Отже, наші початкові витрати на обслуговування навчального асистента складатимуть 210 грн на місяць.

Даний асистент може зацікавити широку аудиторію, але перш за все він орієнтован на студентів та людей молодого віку. Початкова ідея асистенту була у тому, що у користувача можуть буть різні інтереси й великий обсяг непрочитаних книг та статей, тож мета бота підсказати людині статтю для прочитання. Таким чином, користувач рівномірно та збалансовано опановує різні теми й поступово зменшує кількість непрочитаних статей.

Цей проект досить специфічний, але він може знайти свою аудиторію. Одним з ефективним способів залучення аудиторії є написання статті на IT ресурсах як Habrhabr або Tproger, така практика широко використовується навіть великими компаніями, які хочуть продемструвати свій продукт. У цій статті буде розкриватись процесс розробки асистента та використані у ньому методи класифікації текстів, це не тільки безкоштовнио, але разом з тим дозволить нам набрати зацікавлену аудиторію, яка буде слідкувати за розвитком проекту, також ми отримаємо корисний досвід та перші думки й коментарі щодо продукту. Інший додатковий спосіб це реєстрація боту на платформі storebot.me, ця платформа зіставляє рейтинги ботів й пише час від часу про цікаві проекти у своєму блозі, усе це теж безкоштовно.

Навчальний бот-асистент знаходиться у фазі робочого прототипа, тобто випробовується сама ідея та застосовані у боті технології й методи, шукаються й виправляються недоліки, тому зараз він не має ніяких платних функцій й від нього не очікується отримання прибутку.

Однак, в майбутньому коли бот зможе працювати з декількома мовами та розпізнавати більшу кількість тем, то можна буде ввести підписку на нього або заблокований функціонал. Наприклад, користувачі можуть перший місяць

спробувати бота безкоштовно, а потім платити 0.5 долара в місяць за більший обсяг статей, що бот аналізує за місяць, це повинно покрити витрати на хостинг, так як навіть 100 платників дадуть 50 долларів в місяць.

Можливи конкрурентими навчального асистента є ITS (Intelligent tutoring system – інтелектуальні системи навчання) такі як Knewton та Lalilo. У навчального асистента та ITS схожа мета – покращення та персоналізація процессу навчання, за винятком того, що ITS мають більш широкий функціонал та націлені на застосування у навчальних установах, а не для особистого користування. Сучасних ITS систем небагато й вартість їх рішень неможливо оцінити, так як ціна пропозиції формується окремо для кожного випадку. У нашому дослідженні ми не порівнюємо бота-асистента з іншими продуктами, так як прямих конкурентів нема, а ITS не є прямими конкурентами й ціна їх невідома.

4.4. Економічна ефективність

Як було зазначено вище, у ході дослідження був створений навчальний асистент як додаток для особистого використання, тож дане програмне забезпечення не було призначено для впровадження на підприємстві. Отже, ми не можемо обчислити економічну ефектівність цього продукту, але ми можемо визначити його соціальний ефект.

Соціальним ефектом від нашого продукту є покращення та збалансування процесу навчання (зокрема самонавчання) людини за рахунок того, що асистент сам вирішує які статті потрібно прочитати користувачу згідно його потреб та пріоритетів. Таким чином, час на вибір статті та планування процесу навчання зводеться до мінімуму, а користувач поступово зменшує кількість непрочитаних статей й рівномірно охоплює всі цікаві для нього теми.

Окрім цього, напрацювання та технології, використані у цьому продукті, можуть бути повторно застосовані в інших інтелектуальних освітніх системах.

ВИСНОВКИ

У ході роботи було досліджено основні проблеми та завдання напряму NLP з акцентом на методах класифікації текстів. Перші праці в області NLP ставили за мету розробити систему автоматичного перекладу мов. Задача перекладу й породила основні проблеми цього напрямку, на які намагалися відповісти математики, лінгвісти та вчені в галузі кібернетики. З часом, NLP перейшло й до інших задач, таких як розуміння й генерація людської мови, створення штучних співрозмовників, аналіз та класифікація текстів. Загалом розвиток NLP як окремого сформованого напряму досліджень можна охарактеризувати наступним чином.

Спочатку дослідники намагалися представити людську мову як набір формальних правил логіки, але особливості мови не дозволили цього зробити. Згодом після цих невдалих спроб до проблеми обробки мови повернулися з позиції статистичної лінгвістики з метою встановлення принципів побудови висловів у природніх мовах. Лише на початку 21 сторіччя зі зростанням обчислювальних потужностей нейронні мережі стали застосовувати для виявлення закономірностей у мові та взаємозв’язків між словами у тексті.

Кожен з цих етапів надзвичайно важливий. Наприклад, лінгвістичні правила самі по собі не використовуються сьогодні у системах обробки текстів, проте вони заклали фундаментальну теоретичну базу, яка дала розробникам можливість створювати системи, що здатні опрацьовувати тексти. Статистичні дослідження у області мови були важливі для розуміння як текстову інформацію можна подати у числовому вигляді, щоб комп’ютер міг її опрацювати. Великі нейронні мережі хоча й дали поштовх розвитку напрямку NLP не змогли б досягти цих успіхів без надбань попередніх етапів.

Для вирішення проблеми класифікації текстів у нашій роботі задачу було розбити на 2 складові: модель представлення мовної інформації та методи класифікації текстів. Ефективність методів класифікації текстів істотно залежать від набору ознак, які вони отримають на вхід, тому у роботі було проаналізовано

та порівняно обидві складові, з метою реалізації оптимального алгоритму класифікації наукових текстів у навчальному асистенті.

У дослідженні ми порівняли 2 умовні групи моделей представлення текстових даних, які об'єднуються спільним підходом до формування ознак.

Перша група методів представлення текстових даних базується на статистичних методах. Для цієї групи характерний простий математичний підхід до формування ознак тексту, що не враховує порядок слів й контекст. Однак, при цьому текст повинен бути підготовлений для обробки, тобто усі слова повинні бути зведені до нижнього регістру та словникової форми, прибрані знаки пунктуації, лишні пробіли та відступи, тощо. Також потрібно зауважити, що такі моделі, очевидно, не призначені для задач складніше аніж класифікація текстів, вони не вирішують проблему Ципфа, тобто не можуть вгадати значення нового слова з контексту.

Ця група включає 2 методи Bag-of-words та TF-IDF:

1. Bag-of-words (модель «торба слів») – це спрощене подання тексту, яке представлене у вигляді мультимножин (частот) його слів або комбінацій букв (н-грам).

2. TF-IDF (від англ. TF — term frequency, IDF — inverse document frequency) – статистичний показник, що використовується для оцінки важливості слова у контексті документа, що є частиною колекції документів або корпусу. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання слова у інших документах колекції.

Друга група методів представлення текстових даних формує семантичні вектори слів або навіть речень й базується на дистрибутивній гіпотезі. Дистрибутивна гіпотеза робить припущення, що значення слів переважно визначається контекстом, в котрому вони зустрічаються. Принцип роботи цих моделей полягає у тому, що нейронна мережа «без вчителя» приймає на вхід великий об'єм текстів однією мовою й сама оптимально кодує вектори слів відповідно до контексту. Потім навчена таким чином модель кодує й інші тексти у такий же спосіб. Отримані вектори слів знаходяться близько один до одного у

просторі у тому випадку, якщо закодовані цими векторами слова, зустрічаються у схожому контексті, а тому мають близькі чи пов’язанні значення. Прикладом цього підходу є алгоритм word2vec, розроблений працівником Google Томашем Міколовим.

Недоліком векторних моделей є те, що вони працюють по принципу чорного ящика, тож їх результати не завжди можна просто інтерпретувати й для формування якісних семантичних векторів ці моделі потребують великих текстів й обчислювальних потужностей.

Об’єктивно, семантичні вектори слів та речень, що будуються нейромережами, є найбільш досконалими та розвинутими моделями сьогодні. Ці моделі передають значення слів у математичній формі, дозволяють вирішувати проблеми, пов’язані з неоднозначністю значення слів, тож їх можна використовувати для різних завдань.

Однак, у нашому випадку, вирішується лише задача класифікації текстів, власне сенс невідомих слів для нас не є важливим, тому модель TF-IDF виглядає прийнятною. TF-IDF наочно демонструє відмінності у змісті текстів різних категорій й являється непоганою альтернативою більш громіздким та розвинутим моделям у випадку класифікації текстів.

Сьогодні для задач класифікації використовуються наступні методи:

1. Базований на лінгвістичних правилах метод, що застосовувався на початку розвитку NLP й виявився неефективним.

2. Наївний Баєс – статистичний метод класифікації, що базуються на теоремі Баєсу з теорії ймовірності.

3. Логістична регресія - лінійний класифікатор й статистична модель, що використовується для прогнозування вірогідності настання деякого випадку шляхом порівняння його факторів з логістичним рівнянням.

4. SVM – метод векторних машин, що проводить між різними групами сутностей гіперплощини у просторі, які виділяють ці сутності.

5. Нейронні мережі – метод, що використовує математичну абстракцію нейронів задля симуляції процесу прийняття рішення на базі вхідних параметрів.

Згідно з проведеним дослідженням й досвідом розробників у сфері NLP логістична регресія добре підходить для класифікації текстів. При правильному підборі параметрів ця модель може бути непоганою альтернативою іншим більш складним методам, тому що її результати простіші для інтерпретації й вона менш вимоглива до об’ємів тренувальних та навчальних вибірок, ніж глибинні нейронні мережі.

У порівнянні з іншими лінійними класифікаторами такими як Баєс та SVM, логістична регресія краще підходить для задач класифікації з двох причин. По-перше, логістична регресія краще працює з великими об’ємами ніж Баєс, у неї також вище поріг навчання. По-друге, хоча SVM може давати трохи точніші результати, але логістична регресія на відміну від SVM дає ймовірнісну відповідь, у випадку класифікації текстів це корисно тим, що у тексті можуть бути декілька взаємопов’язаних тем, тож у цьому випадку доречніше віднести текст до декількох категорій.

Для вибору оптимальної моделі представлення даних в ході роботи було проаналізовано досвід використання розробниками моделей TF-IDF та моделі word2vec. В результаті виявилось, що точність класифікації моделі з word2vec нижче, ніж при застосуванні TF-IDF. У досліді використовувалась модель word2vec від Google, що була навчена на корпусі текстів з новин розміром 300 більйонів слів. .Класифікатори випробовувались на розміченому датасеті з 40000 текстів з програмування 20 категорій з ресурсу stack overflow: В результаті, модель з TF-IDF показала точність 79% проти 64% у word2vec. На основі цього було зроблено висновок, що TF-IDF в цілому краще підходить для задач класифікації.

Після ретельного аналізу вище наведених методів та моделей було розроблено навчального асистента, який використовує логістичну регресію та TF-IDF показник для класифікації текстів. Згідно з задачею модель регресії повинна була класифікувати англомовні тексти з 4 тем, а саме з економіки, історії, IT та математики. Модель була навчена на невеликій вибірці зі 100 текстів з Вікіпедії, для навчання використовувались 70 текстів, а для тестування

– 30. Навчена таким чином модель регресії дала точність 97%. Отримана в результаті навчання модель була скомбінована з методом ранжування для того щоб навчальний асистент міг пропонувати статті залежно від пріоритетів користувача за темами. Асистент, побудований з даною моделлю, вдало класифікував статті з невідомих джерел.

В результаті, було зроблено висновок, що логістична регресія може успішно застосовуватись для класифікації наукових текстів, якщо вона використовується разом з TF-IDF ознакою. Незважаючи на свою математичну простоту, цей метод досить точно класифікував тексти з невідомих для нього джерел навіть з невеликим об’ємом навчальних даних завдяки правильно підібраним параметрам та ознакам.

Ми вважаємо, що для того щоб робити однозначні висновки щодо ефективності логістичної регресії у порівнянні з більш складними та масштабними моделями потрібно провести досліди на більшому об’ємі даних й з більшою кількістю класів та мов.

Однак, можна впевнено стверджувати, шо логістична регресія все ще актуальна в задачах класифікації текстів й повинна розглядатись при проектуванні інтелектуальних систем, так як при своїй відносній математичній простоті вона дає високу точність класифікації. Окрім цього, логістична регресія не потребує великих обчислювальних ресурсів й добре навчається на відносно невеликій вибірці.

Потрібно зазначити, що точність моделі лише на 50% залежить від власне самого правильно налагодженого класифікатора, інші 50% припадають на якісну навчально вибірку та вірно підібрані під задачу ознаки.

На наш погляд, проведене нами дослідження й розроблений у процесі продукт буде корисним як на практиці для покращення освітнього процесу, але звісно асистент повинен бути розширений, так й як приклад інтелектуального асистента для науковців, студентів та розробників, що цікавляться обробкою людської мови.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Artetxe M., Labaka G., Agirre E. (2019). An Effective Approach to Unsupervised Machine Translation / M. Artetxe, G. Labaka, E. Agirre // ArXiv. – Vol. 1902 – 2019 – P. 10.

2. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate. / D. Bahdanau, K. Cho, Y. Bengio// ArXiv. – Vol. 1409 – 2014 – P. 15.

3. Berger A., Della Pietra S. A., Della Pietra V. J. A maximum entropy approach to natural language processing. / A. Berger, S. A Della Pietra, V.J. Della Pietra // Computational Linguistics - Vol. 22 – 1996 - 39–71 P.

4. Beysolow T. Applied Natural Language Processing with Python / T. Beysolow – Apress, 2018 – 158 P.

5. Bingham C., Marascuilo L., Levin J. Multivariate Statistics in the Social Sciences: A Researcher's Guide. / C. Bingham, L. Marascuilo, J. Levin // Journal of the American Statistical Association. – Vol. 80. – 1985 – P. 243.

6. Bishop C. Pattern Recognition and Machine Learning / C. Bishop – Springer, 2006 – 758 P.

7. Brusilovsky, P. Adaptive and intelligent Web-based educational systems / P. Brusilovsky, C. Peylo // International Journal of Artificial Intelligence in Education.Special Issue on Adaptive and Intelligent Web-based Educational Systems – 2003. –№ 13 (2-4). – P. 159-172.

8. Buchanan B. G., Shortliffe E. H. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project / B. G. Buchanan, E. H. Shortliffe. – MA: Addison-Wesley, 1984. – 769 p.

9. Chen Y., Liu P., Teo C. Regularised Text Logistic Regression: Key Word Detection and Sentiment Classification for Online Reviews / Y. Chen, P. Liu, C. Teo. – 2020. – P. 23.

10. Christopher D., Schütze M. H. Foundations of Statistical Natural Language Processing / D. Christopher, M. H. Schütze – Manning, 1999. – P. 704.

11. Cox D. R. Analysis of Binary Data. / D. R. Cox –Department of Mathematics Imperial College London - Chapman and Hall, London. – 1969 – 231 P.

12. Devlin J., Chang M. W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin, M. W. Chang, K. Lee, K. Toutanova. – 2018. – P. 16.

13 Dubrov B. Predicting Stock Returns by Automatically Analyzing Company News Announcements. / B. Dubrov // SSRN Electronic Journal. –2015 - 40 P.

14 Encyclopedia of Artificial Intelligence. 2nd ed. / [S. C. Shapiro editor]. – New York: John Wiley & Sons, 1992. – Volume 1. – p. 434.

15. Foster D. Generative Deep Learning / D. Foster – O'Reilly, 2019 – 305 P.

16. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 2nd edition / A. Geron – O'Reilly, 2019 – 600 P.

17. Goldberg Y. A Primer on Neural Network Models for Natural Language Processing. / Y. Goldberd // Journal of Artificial Intelligence Research. – Vol. 57 – 2015 -76 P.

18. Gortmaker S., Hosmer D., Lemeshow S. (1994). Applied Logistic Regression. / S. Gortmaker, D. Hosmer // Contemporary Sociology. – Vol. 23. – 1994 – P. 159 - 166.

19. Hadelin de Ponteves AI Crash Course: A fun and hands-on introduction to machine learning, reinforcement learning, deep learning, and artificial intelligence with Python / Hadelin de Ponteves – Packt, 2019 –362 P.

20. Harrison M. Machine Learning Pocket Reference / M. Harrison - O'Reilly, 2019 – 485 P.

21. Hastie T., Tibshirani R. J., Friedman J.H. The Elements of Statistical Learning / T. Hastie., R. J. Tibshirani, J.H. Friedman – Springer – 2001 – 764 P.

22. Hearty J. Advanced Machine Learning with Python / J. Hearty – O'Reilly, 2016 – 278 P.

23. Henley A.J., Wolf D. Learn Data Analysis with Python / A.J. Henley, D. Wolf – Apress, 2018 – 103 P.

24. Hochreiter S., Schmidhuber J. (1997). Long Short-term Memory./ S. Hochreiter, J. Schmidhuber // Neural computation.- Vol. 9. – 1997 – P. 32.

25. Hsu C., Chang C., Lin C. J. A Practical Guide to Support Vector Classication. / C. Hso, C. Chang, C. J. Lin - Bioinformatics. – Vol. 1 – 2003 – P. 16.

26. James G. An Introduction to Statistical Learning / G. James – Springer, 2013 – 434 P.

27. Jessen H., Menard S. (1996). Applied Logistic Regression Analysis. / H. Jessen, S. Menard // The Statistician. – Vol. 45. – 1996 – P. 20.

28. Jones K. Natural language processing: a historical review. / K. Jones – 2001. – P. 14.

29. Jurafsky D. Speech and Language Processing. An Introduction to Natural Language Processing. Computational Linguistics, and Speech Recognition 3rd edition draft/ D. Jurafsky, J. H. Martin - Stanford University - Stanford University Press, 2019 – 621 P.

30. Kahane S. The meaning-text theory. / S. Kahane //Conference: Dependency and Valency (Ed.), An International Handbook of Contemporary – 2003. – Vol. 1. – P. 37.

31. Kavita G., Subotin M. (2015). A general supervised approach to segmentation of clinical texts. Proceedings. / G. Kavita, M. Subotin // 2014 IEEE International Conference on Big Data, IEEE Big Data 2014 – P. 33 – 40.

32. Kehler A. Probabilistic coreference in information extraction. / A. Kehler // In EMNLP – 1997 – P. 163–173.

33. Klenin J., Botov D., Dmitrin Y. Comparison of Vector Space Representations of Documents for the Task of Information Retrieval of Massive Open Online Courses./ J. Klenin, D. Botov, Y. Dmitrin // Communications in Computer and Information Science. – 2018. – P. 156 – 164.

34. Kulkarni A., Shivananda A. Natural Language Processing Recipes / A. Kulkarni, A. Shivananda – Apress, 2019 – 253 P.

35. Lane H. Natural Language Processing in Action / H. Lane, C. Howard, H. M. Hapke, A. Griffioen – Manning, 2019 – P. 545.

36. Li Susan Multi-Class Text Classification Model Comparison and Selection. [Електронний ресурс] / Susan Li – режим доступу: https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568#:~:text=Word2vec%20is%20a%20type%20of,layer%20encodes%20the%20word%20representation.

37. Manohar S. Mastering Machine Learning with Python in Six Steps / S. Manohar – Apress, 2017 – 374 P.

38. Mauro Di Pietro Text Classification with NLP: Tf-Idf vs Word2Vec vs BERT. [Електронний ресурс] / Mauro Di Pietro - режим доступу: https://towardsdatascience.com/text-classification-with-nlp-tf-idf-vs-word2vec-vs-bert-41ff868d1794

39. Menard S. Applied logistic regression analysis. / S. Menard // Thousand Oaks, CA: Sage. Sage University Paper series on Quantitative Applications in the Social Sciences. – Vol. 7 – 1995 – P. 106.

40. Mikolov T., Chen K., Corrado G.s, Dean J. Distributed Representations of Words and Phrases and their Compositionality / T. Mikolov, K. Chen, G.s Corrado, J. Dean // NIPS - 2013 – P. 1-9.

41. Mikolov T., Chen K., Corrado G.s, Dean J. Efficient Estimation of Word Representations in Vector Space. 2013. / T. Mikolov, K. Chen, G.s Corrado, J. Dean // Proceedings of Workshop at ICLR. – 2013

42. Moroney L. AI and Machine Learning for Coders / L. Moroney – O'Reilly, 2020 – 390 P.

43. Müller A.C., Guido S. Introduction to Machine Learning with Python A Guide for Data Scientists / A.C. Müller, S. Gudio – O'Reilly, 2017 – 392 P.

44. Murphy, K. P. Machine learning: A probabilistic perspective. / K. P. Murphy - MIT press – 2012 – 1098 P.

45. Murray, T. Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art / T. Murray. // International Journal of Artificial Intelligence in Education. – 1999.– № 10 – P. 98-129.

46. Ng A. Y., and Jordan M. I. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. / A. Y. Ng, M. I. Jordan // Conference on Neural Information Processing Systems – Vol. 14 - 841–848 P.

47. Nielsen M. Neural Networks and Deep Learning / M. Nielsen – Online, 2016 – 560 P.

48. Nigam, K., Lafferty, J. D., McCallum A. Using maximum entropy for text classification. / K. Nigam, J. D. Lafferty, A. McCallum // International Joint Conferences on Artificial Intelligence – 1999 – P. 61–67.

49. Nilsson N. J. INTRODUCTION TO MACHINE LEARNING (draft) / N. J. Nilsson - Stanford University, 2020 – 188.P.

50. Peng J., Lee K., Ingersoll G. An Introduction to Logistic Regression Analysis and Reporting. Journal of Educational Research / J. Peng, K. Lee, G. Ingersoll //- The Journal of Educational Research. – 2002 – Vol. 96 - P. 3 - 14.

51. Perotta P. Programming Machine Learning From Coding to Deep Learning / P. Perotta - The Pragmatic Programmers, 2020 – 326 P.

52. Raicu I., Bologa R., Constaninescu R. MULTI-CLASS TEXT SUPERVISED CLASSIFICATION ON ROMANIAN FINANCIAL BANKING REVIEWS. / I. Raicu, R. Bologa, R. Constaninescu // Conference: 18th International Conference on INFORMATICS in ECONOMY. Education, Research and Business Technologies. – 2019. – P. 6.

53. Raschka S. Python Machine Learning - Third Edition / S. Raschka – Packt, 2019 –P. 770.

54. Ratnaparkhi A.. A linear observed time statistical parser based on maximum entropy models ./ A. Ratnaparkhi // Empirical Methods in Natural Language Processing. – Vol. 1 – 1997 – P. 1–10.

55. Rosenfeld R. A. maximum entropy approach to adaptive statistical language modeling. / R. A. Rosenfeld // Computer Speech and Language. – Vol. 10 –P.187–228.

56. Rygl J., Pomikálek. J., Řehůřek R., Růžička M., Novotný V., Sojka P. Semantic Vector Encoding and Similarity Search Using Fulltext Search Engines./ J. Rygl, J. Pomikálek. R. Řehůřek, M. Růžička, V. Novotný, P. Sojka // Proceedings of the 2nd Workshop on Representation Learning for NLP – 2017 – P. 81 – 90.

57. Sahlgren M. The distributional hypothesis. / M. Sahlgren // Italian Journal of Linguistics. - 2008 - Vol. 20 – P. 21.

58. Sankoff D. and Labov W. On the uses of variable rules. / D. Sankoff, W. Labov // Language in society – Vol. 8 – 1996 – P. 189–222.

59. Schutze H., Hull D. A., and Pedersen J. / H. Shutze, D.A. Hull, J. Pedersen // A comparison of classifiers and document representations for the routing problem. Conference of Special Interest Group on Information Retrieval - 229–237.

60. Sinha S. Robust designs for multivariate logistic regression. / S. Sinha // METRON – Vol. 71.

61. Taddy M. Business Data Science / M. Taddy - McGraw-Hill Education, 2019 – P. 352.

62. Thangara, M., Sivakami M. Text Classification Techniques: / M. Thangara, M. Sivakami // A Literature Review. Interdisciplinary Journal of Information, Knowledge, and Management. – 2018 – P. 117 - 135.

63. Thomas R., Zikopulos P. The AI Ladder: Accelerate Your Journey to AI / R. Thomas, P. Zikopulos – O'Reilly, 2020 –226 P.

64. Tibshirani R. J. Regression shrinkage and selection via the lasso. / R.J. Tibshirani // Journal of the Royal Statistical Society. Series B (Methodological) – Vol. 58 – P. 267–288.

65. Vaughan D. Analytical Skills for AI and Data Science / D. Vaughan – O'Reilly, 2020 – P.232.

66. Wang H., Raj B. On the Origin of Deep Learning. / H. Wang, B. Raj – arxiv – Vol 1702 – 72 P.

67. Wang S., Manning C. D. Baselines and bigrams: Simple, good sentiment and topic classification. / S. Wang, C.D. Manning // Annual Meeting of the Association for Computational Linguistics – 2012 – P. 90–94.

68. Witten I. H., Frank E. Data Mining: Practical Machine Learning Tools and Techniques (2nd Ed.). / I. H. Witten, E. Frank - Morgan Kaufmann Publishers, 2005 - 664 P.

69. Бурков А. Машинное обучение без лишних слов / А. Бурков - Издательский дом "Питер", 2020 – 192 с.

70. Вьюгин В.В Математические основы теории машинного обучения и прогнозировани / В.В. Вьюгин – Москва, 2013, Библ. 48. – 387 с.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Модуль web_scrape.py для завантаження та обробки навчальних текстів

```
import requests
import pprint
from bs4 import BeautifulSoup
import json
import os.path

url_sets = [
    {
        'category': 'economics',
        'urls': [
            'https://en.wikipedia.org/wiki/Economic_system',
            'https://en.wikipedia.org/wiki/Economic_ideology',
            'https://en.wikipedia.org/wiki/Economy',
            'https://en.wikipedia.org/wiki/Capitalism',
            'https://en.wikipedia.org/wiki/Communism',
            'https://en.wikipedia.org/wiki/Distributism',
            'https://en.wikipedia.org/wiki/Feudalism',
            'https://en.wikipedia.org/wiki/Inclusive_Democracy',
            'https://en.wikipedia.org/wiki/Market_economy',
            'https://en.wikipedia.org/wiki/Mercantilism',
            'https://en.wikipedia.org/wiki/Market_economy',
            'https://en.wikipedia.org/wiki/Mixed_economy',
            'https://en.wikipedia.org/wiki/Planned_economy',
            'https://en.wikipedia.org/wiki/Traditional_economy',
            'https://en.wikipedia.org/wiki/Non-monetary_economy',
            'https://en.wikipedia.org/wiki/Subsistence_economy',
            'https://en.wikipedia.org/wiki/Gift_economy',
            'https://en.wikipedia.org/wiki/Barter',
            'https://en.wikipedia.org/wiki/Participatory_economics',
            'https://en.wikipedia.org/wiki/Ecological_economics',
            'https://en.wikipedia.org/wiki/Development_economics',
            'https://en.wikipedia.org/wiki/Schools_of_economic_thought',
            'https://en.wikipedia.org/wiki/History_of_economic_thought',
            'https://en.wikipedia.org/wiki/Austrian_School',
            'https://en.wikipedia.org/wiki/Keynesian_economics'
        ]
    },
    {
        'category': 'mathematics',
        'urls': [
            'https://en.wikipedia.org/wiki/Mathematics',
            'https://en.wikipedia.org/wiki/Geometry',
            'https://en.wikipedia.org/wiki/Probability_theory',
            'https://en.wikipedia.org/wiki/Derivative',
            'https://en.wikipedia.org/wiki/Integral',
            'https://en.wikipedia.org/wiki/Differential_equation',
            'https://en.wikipedia.org/wiki/Function_(mathematics)',
            'https://en.wikipedia.org/wiki/Number_theory',
            'https://en.wikipedia.org/wiki/Algebra',
            'https://en.wikipedia.org/wiki/Mathematical_analysis',
            'https://en.wikipedia.org/wiki/Mathematics_in_medieval_Islam',
            'https://en.wikipedia.org/wiki/Algebraic_geometry',
            'https://en.wikipedia.org/wiki/Synthetic_geometry',
            'https://en.wikipedia.org/wiki/Non-Euclidean_geometry',
            'https://en.wikipedia.org/wiki/Differential_geometry',
            'https://en.wikipedia.org/wiki/Complex_geometry',
            'https://en.wikipedia.org/wiki/Mathematical_visualization',
            'https://en.wikipedia.org/wiki/Desargues%27s_theorem',
            'https://en.wikipedia.org/wiki/Euclidean_space',
            'https://en.wikipedia.org/wiki/Elliptic_geometry',
            'https://en.wikipedia.org/wiki/Hyperbolic_geometry',
            'https://en.wikipedia.org/wiki/Discrete_mathematics',
            'https://en.wikipedia.org/wiki/Differential_calculus',
```

```
            'https://en.wikipedia.org/wiki/Calculus',
            'https://en.wikipedia.org/wiki/Euclidean_vector'
        ]
    },
    {
        'category': 'history',
        'urls': [
            'https://en.wikipedia.org/wiki/History',
            'https://en.wikipedia.org/wiki/Ancient_Greece',
            'https://en.wikipedia.org/wiki/Ancient_Rome',
            'https://en.wikipedia.org/wiki/Roman_Empire',
            'https://en.wikipedia.org/wiki/Roman_Republic',
            'https://en.wikipedia.org/wiki/French_Revolution',
            'https://en.wikipedia.org/wiki/German_revolutions_of_1848%E2%80%931849',
            'https://en.wikipedia.org/wiki/Napoleonic_Wars',
            'https://en.wikipedia.org/wiki/World_War_I',
            'https://en.wikipedia.org/wiki/World_War_II',
            'https://en.wikipedia.org/wiki/Constitution_of_the_Roman_Kingdom',
            'https://en.wikipedia.org/wiki/Constitution_of_the_Roman_Republic',
            'https://en.wikipedia.org/wiki/Constitution_of_the_Roman_Empire',
            'https://en.wikipedia.org/wiki/Constitution_of_the_Late_Roman_Empire',
            'https://en.wikipedia.org/wiki/Roman_Senate',
            'https://en.wikipedia.org/wiki/Roman_assemblies',
            'https://en.wikipedia.org/wiki/Roman_magistrate',
            'https://en.wikipedia.org/wiki/Ancient_Rome',
            'https://en.wikipedia.org/wiki/History_of_Greece',
            'https://en.wikipedia.org/wiki/Human_history',
            'https://en.wikipedia.org/wiki/Mesopotamia',
            'https://en.wikipedia.org/wiki/Sasanian_Empire',
            'https://en.wikipedia.org/wiki/Syria%E2%80%93Turkey_border',
            'https://en.wikipedia.org/wiki/Fall_of_the_Western_Roman_Empire',
            'https://en.wikipedia.org/wiki/Invasion_of_Poland'
        ]
    },
    {
        'category': 'IT',
        'urls': [
            'https://en.wikipedia.org/wiki/Machine_learning',
            'https://en.wikipedia.org/wiki/Artificial_intelligence',
            'https://en.wikipedia.org/wiki/Functional_programming',
            'https://en.wikipedia.org/wiki/Object-oriented_programming',
            'https://en.wikipedia.org/wiki/Class_(computer_programming)',
            'https://en.wikipedia.org/wiki/Object_(computer_science)',
            'https://en.wikipedia.org/wiki/Computer_science',
            'https://en.wikipedia.org/wiki/Data_structure',
            'https://en.wikipedia.org/wiki/Programming_language',
            'https://en.wikipedia.org/wiki/Informatics',
            'https://en.wikipedia.org/wiki/Agent-oriented_programming',
            'https://en.wikipedia.org/wiki/Array_programming',
            'https://en.wikipedia.org/wiki/Concurrent_computing',
            'https://en.wikipedia.org/wiki/Inductive_logic_programming',
            'https://en.wikipedia.org/wiki/Ontology_language',
            'https://en.wikipedia.org/wiki/Tacit_programming',
            'https://en.wikipedia.org/wiki/Concatenative_programming_language',
            'https://en.wikipedia.org/wiki/Generic_programming',
            'https://en.wikipedia.org/wiki/Imperative_programming',
            'https://en.wikipedia.org/wiki/Procedural_programming',
            'https://en.wikipedia.org/wiki/Object-oriented_programming',
            'https://en.wikipedia.org/wiki/Polymorphic_code',
            'https://en.wikipedia.org/wiki/Inductive_programming',
            'https://en.wikipedia.org/wiki/Cluster_analysis',
            'https://en.wikipedia.org/wiki/Computational_complexity'
        ]
    }
]

for url_set in url_sets:
    url_set['texts'] = []
    for url in url_set['urls']:
        page = requests.get(url)
        print('page - ' + url + ' is loaded')
        soup = (BeautifulSoup(page.content, 'html.parser'))
        to_remove = soup.find_all('span')
        to_remove.extend(soup.find_all(class_="mwe-math-element"))
        for elem in to_remove:
            elem.decompose()

        paragraphs = soup.find_all('p')
        url_set['texts'].append('')
        index = len(url_set['texts']) - 1
```

```
        for paragraph in paragraphs:
            url_set['texts'][index] = url_set['texts'][index] + ' ' + paragraph.get_text()

APP_ROOT = os.path.dirname(os.path.abspath(__file__))

for url_set in url_sets:
    category = url_set['category']
    dirname = './train-data/' + category

    if not os.path.exists(os.path.join(APP_ROOT, dirname)):
        os.makedirs(dirname)

    for index in range(len(url_set['texts'])):
        file_path = './train-data/' + category + '/' + category  + '-' + str(index) + '.txt'
        path = os.path.join(APP_ROOT, file_path)
        text_file = open(path, "w")
        n = text_file.write(url_set['texts'][index])
        text_file.close()
```

Модуль logistic_regression.py для тренування моделі логістичної регресії на навчальній вибірці від парсера web_scrape.py

```
import numpy as np
import re
import nltk
from sklearn.datasets import load_files
import os.path
import pickle
import shutil
from nltk.corpus import stopwords

train_data = load_files(r"./train-data")
X, y = train_data.data, train_data.target
original_texts = X[:]
documents = []
print(train_data.target_names)

from nltk.stem import WordNetLemmatizer

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):

    # Remove all the special characters
    document = re.sub(r'\W', ' ', str(X[sen]))

    #Remove all number

    document = re.sub(r'[0-9]+', ' ', document)

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags=re.I)

    # Removing prefixed 'b'
    document = re.sub(r'^b\s+', '', document)

    # Converting to Lowercase
    document = document.lower()

    # Lemmatization
    document = document.split()

    document = [stemmer.lemmatize(word) for word in document]
    document = ' '.join(document)

    documents.append(document)

from sklearn.feature_extraction.text import TfidfTransformer, TfidfVectorizer
tfidfconverter = TfidfVectorizer(max_features=3500, min_df=5, max_df=0.8, stop_words=stopwor
ds.words('english'))
X = tfidfconverter.fit_transform(documents).toarray()
idf = tfidfconverter.idf_

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
original_texts_train, original_texts_test = train_test_split(original_texts, test_size=0.3,
random_state=0)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(verbose=1, solver='liblinear',random_state=0, C=5, penalty='
l2',max_iter=1000)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print('\nLOGISTIC REGRESSION')
print(classification_report(y_test,y_pred))
```

```
APP_ROOT = os.path.dirname(os.path.abspath(__file__))

if os.path.exists(os.path.join(APP_ROOT, './results/')):
    shutil.rmtree(os.path.join(APP_ROOT, './results/'))

index = 0
for result in y_pred:
    category = train_data.target_names[result]
    dirname = './results/' + category
    if not os.path.exists(os.path.join(APP_ROOT, dirname)):
        os.makedirs(dirname)
    path = os.path.join(APP_ROOT, dirname + '/' + category + '-' + str(index) + '.txt')
    text_file = open(path, "w")
    n = text_file.write(str(original_texts_test[index]))
    text_file.close()
    index = index + 1

with open('text_classifier', 'wb') as picklefile:
    pickle.dump(classifier,picklefile)

vocabulary_coefs = dict(zip(tfidfconverter.get_feature_names(), idf))
with open('vocabulary_coefs', 'wb') as picklefile:
    pickle.dump(vocabulary_coefs,picklefile)

with open('tfidfconverter_trained', 'wb') as picklefile:
    pickle.dump(tfidfconverter, picklefile)
```

Модуль classification_module.py для класифікації текстів з допомогою навченої моделі логістичної регресії

```
        import re
        import pickle
        from bs4 import BeautifulSoup
        from sklearn.datasets import load_files
        import requests
        import nltk
        # shouldn't be here really
        nltk.download('stopwords')
        nltk.download('wordnet')

        # basic model needs to be improved
        categories = ['IT', 'economics', 'history', 'mathematics']

        with open('text_classifier', 'rb') as training_model:
            model = pickle.load(training_model)

        with open('tfidfconverter_trained', 'rb') as tfidfconverter_trained:
            tfidfconverter_trained = pickle.load(tfidfconverter_trained)

        def classify_sites(url_list):
            result = []
            raw_texts = []
            try:
                raw_texts = fetch_texts_by_url(url_list)
            except BaseException as e:
                raw_texts = []
            if len(raw_texts) > 0:
                processed_texts = process_texts(raw_texts)
                transformed_texts = tfidfconverter_trained.transform(processed_texts).toarray()
                prediction_categories = map(lambda index: categories[index],
model.predict(transformed_texts))
                sizes = [len(text) for text in raw_texts]
                result = list(zip(url_list, prediction_categories, sizes))
            return result

        def fetch_texts_by_url(url_set):
            result = []
            for url in url_set:
                page = requests.get(url)
                soup = (BeautifulSoup(page.content, 'html.parser'))
                to_remove = soup.find_all('span')
                to_remove.extend(soup.find_all(class_="mwe-math-element"))
                for elem in to_remove:
                    elem.decompose()

                paragraphs = soup.find_all('p')
                text = ''

                for paragraph in paragraphs:
                    if(len(text) == 0):
                        text = paragraph.get_text()
                    else:
                        text = text + ' ' + paragraph.get_text()
                result.append(text)

            return result

        def process_texts(texts):
            result = []
            stemmer = nltk.stem.WordNetLemmatizer()
            for text in texts:

                # Remove all the special characters
                document = re.sub(r'\W', ' ', str(text))

                #Remove all number

                document = re.sub(r'[0-9]+', ' ', document)

                # remove all single characters
                document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)
```

```
        # Remove single characters from the start
        document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

        # Substituting multiple spaces with single space
        document = re.sub(r'\s+', ' ', document, flags=re.I)

        # Removing prefixed 'b'
        document = re.sub(r'^b\s+', '', document)

        # Converting to Lowercase
        document = document.lower()

        # Lemmatization
        document = document.split()

        document = [stemmer.lemmatize(word) for word in document]
        document = ' '.join(document)

        result.append(document)

    return result

def test_module():
    test_set = [
        'https://en.wikipedia.org/wiki/Fascism',
        'https://en.wikipedia.org/wiki/Neoliberalism',
        'https://en.wikipedia.org/wiki/Compiler',
        'https://en.wikipedia.org/wiki/Euclidean_geometry',
        'https://en.wikipedia.org/wiki/Inflation',
        'https://en.wikipedia.org/wiki/Battle_of_Britain',
        'https://en.wikipedia.org/wiki/Source_code',
        'https://en.wikipedia.org/wiki/Algorithm'
    ]

    result = classify_sites(test_set)

    print('TESTING CLASSIFICATION MODULE...')
    print(result)
    return None

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

Модуль database_module.py для роботи з БД

```
        from flask import Flask, request
        from flask_sqlalchemy import SQLAlchemy
        from flask_migrate import Migrate
        from textrec.classification_module import categories
        from app import db

        # app = None
        # db = SQLAlchemy()
        # migrate = Migrate(app, db)

        # need to improve naming - link and article_link are confusing

        class UserModel(db.Model):
            __tablename__ = 'Users'

            user_id = db.Column(db.String(), primary_key=True)
            user_progress_by_article = db.relationship('UserProgressByArticleModel', backref='users',
lazy=True)
            user_progress_by_class  =  db.relationship('UserProgressByClassModel',  backref='users',
lazy=True)

        class ArticleModel(db.Model):
            __tablename__ = 'Articles'

            link = db.Column(db.String(), primary_key=True)
            size = db.Column(db.Integer)
            article_class = db.Column(db.String(), nullable = False)

            progress_by_article  =  db.relationship('UserProgressByArticleModel',  backref='articles',
lazy=True)

        class UserProgressByArticleModel(db.Model):
            __tablename__ = 'UserProgressByArticle'

            user_article_progress_id = db.Column(db.Integer, primary_key=True)
            article_link = db.Column(db.String(), db.ForeignKey('Articles.link'), nullable=False)
            user_id = db.Column(db.String(), db.ForeignKey('Users.user_id'), nullable=False)
            article_progress = db.Column(db.Float())

        class UserProgressByClassModel(db.Model):
            __tablename__ = 'UserProgressByClass'

            user_class_progress_id = db.Column(db.Integer, primary_key=True)
            user_id = db.Column(db.String(), db.ForeignKey('Users.user_id'), nullable=False)
            article_class = db.Column(db.String(), nullable = False)
            class_progress = db.Column(db.Float())
            user_class_priority = db.Column(db.Integer)
            class_size = db.Column(db.Integer)

        def add_user_id(id):
            new_user = UserModel(user_id = id)
            db.session.add(new_user)
            db.session.commit()
            return

        def add_articles(articles):
            for article in articles:
                db.session.add(article)
            db.session.commit()
            return

        def get_articles():
            return ArticleModel.query.all()

        # not flexible (quick solution)
        def add_article_user_reports(article_reports):
            if len(article_reports) > 0:
                user_id = article_reports[0].user_id
                # checks old_articles and add only new aricles
                old_reports                                                                    =
UserProgressByArticleModel.query.filter(UserProgressByArticleModel.user_id == str(user_id)).all()
                old_reports_links = [report.article_link for report in old_reports]
                new_reports = [report for report in  article_reports if report.article_link not in
old_reports_links]
```

```
                for report in new_reports:
                    db.session.add(report)
                db.session.commit()

                # updates all categories every time when new articles are added
                if len(new_reports) > 0:
                    for category in categories:
                        class_progress,  class_size  =  count_progress_and_size_by_class(user_id  =
user_id, article_class = category)
                        update_class_user_report(
                            user_id = user_id, article_class = category,
                            progress = class_progress, class_size = class_size)
            return

        def get_article_user_reports(user_id):
            article_user_reports = UserProgressByArticleModel.query\
                .filter(UserProgressByArticleModel.user_id == str(user_id)).all()
            return article_user_reports

        def get_article_user_reports_for_category(user_id, article_class):
            article_progress_reports = db.session.query(
                UserProgressByArticleModel.article_progress,
                UserProgressByArticleModel.article_link,
                ArticleModel.article_class).\
                filter(UserProgressByArticleModel.user_id == str(user_id))\
                .join(ArticleModel).filter(ArticleModel.article_class == article_class).all()

            return article_progress_reports

        def  update_article_user_report(progress,  user_article_progress_id  =  None,  user_id  =  None,
article_link = None):
            if user_article_progress_id:
                progress_report = UserProgressByArticleModel.query.filter(
                    UserProgressByArticleModel.user_article_progress_id                              ==
user_article_progress_id).first()
                progress_report.article_progress = progress
                db.session.commit()

                article_class     =     ArticleModel.query.filter(progress_report.article_link     ==
ArticleModel.link).first().article_class
                class_progress, class_size = count_progress_and_size_by_class(
                    user_id = progress_report.user_id,
                    article_class = article_class)
                update_class_user_report(
                    user_id = progress_report.user_id, article_class = article_class,
                    progress = class_progress, class_size = class_size)
            else:
                pass
            return

        def add_class_user_reports(class_reports):
            for report in class_reports:
                db.session.add(report)
            db.session.commit()

        def update_class_user_report(user_id, article_class, progress, class_size):
            class_report = UserProgressByClassModel.query.filter(
                UserProgressByClassModel.user_id == str(user_id),
                UserProgressByClassModel.article_class == str(article_class)).first()
            class_report.class_progress = progress
            class_report.class_size = class_size
            db.session.commit()
            return

        def get_class_user_reports(user_id):
            class_user_reports = UserProgressByClassModel.query\
                .filter(UserProgressByClassModel.user_id == str(user_id)).all()
            return class_user_reports

        def set_user_priorities(user_id, ordered_topics):
            class_user_reports = UserProgressByClassModel.query\
                .filter(UserProgressByClassModel.user_id == str(user_id)).all()

            if(len(ordered_topics) != len(class_user_reports)):
                db.session.expire_all()
                raise Exception('Incorrect input. Input array is short')

            # clunky implementation
            lower_case_categories = [category.lower() for category in categories]
            for index in range(len(ordered_topics)):
```

```
                #checking that input is correct
                if(ordered_topics[index].lower() not in lower_case_categories):
                    db.session.expire_all()
                    raise Exception('Incorrect input. Provided categories don\'t exist')

                if(ordered_topics[index] in ordered_topics[:index]):
                    db.session.expire_all
                    raise Exception('Incorrect input. Duplicate categories')

                for report in class_user_reports:
                        if report.article_class.lower() == ordered_topics[index].lower():
                            report.user_class_priority = index + 1
                            break

                db.session.commit()
            return

        def count_progress_and_size_by_class(user_id, article_class):
            article_progress_reports = db.session.query(
                UserProgressByArticleModel.article_progress,
                ArticleModel.article_class,
                ArticleModel.size).\
                filter(UserProgressByArticleModel.user_id == str(user_id))\
                .join(ArticleModel).filter(ArticleModel.article_class == article_class).all()

            class_size = 0
            read_words = 0
            for row in article_progress_reports:
                read_words = read_words + round(row.size * row.article_progress)
                class_size = class_size + row.size
            if class_size == 0:
                return (0,0)
            else:
                return (round(read_words / class_size, 2), class_size)

        def get_user_class_progress(user_id):
            return      UserProgressByClassModel.query.filter(UserProgressByClassModel.user_id      ==
str(user_id)).all()
```

Модуль app.py – основний модуль навчального асистента, що об’єднує в собі попередні 2 модулі й відправляє відповіді на клієнта

```
from flask import Flask, request
import telegram
import re
import random
from telebot.credentials import bot_token, bot_user_name,URL
import urllib.parse
from requests import get
from textrec.classification_module import classify_sites, categories
from flask_sqlalchemy import SQLAlchemy
import os


global bot
global TOKEN
TOKEN = bot_token
bot = telegram.Bot(token=TOKEN)
# saves ids of users and ids of their article progress reports
pending_progress_updates = {}

app = Flask(__name__)
app.config.from_object(os.environ['APP_SETTINGS'])
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] =
"postgres://mimfyozwmzmfcs:909a85f7da63119c40a9a7f66ce554a6e1da0264a0dc3b91371d9096c1b64e68@ec2-18-210-214-86.compute-1.amazonaws.com:5432/db0llm2ic4ehj8"
db = SQLAlchemy(app)

import dbController.database_module as db_module

@app.route('/' + TOKEN, methods=['GET', 'POST'])
def respond():
    # retrieve the message in JSON and then transform it to Telegram object
    update = telegram.Update.de_json(request.get_json(force=True), bot)
    message_data = ''
    callback_data = ''
    user_id = ''

    if update.message is not None:
        message_data = update.message
        user_id = message_data.from_user.id
    elif update.callback_query.message is not None:
        message_data = update.callback_query.message
        callback_data = update.callback_query.data
        user_id = update.callback_query.from_user.id

    chat_id = message_data.chat.id
    msg_id = message_data.message_id

    response_text = ''
    markup = None
    pattern = re.compile("^\/load.*")

    # Telegram understands UTF-8, so encode text for unicode compatibility
    # add handler for callback_data

    if not bool(callback_data):
        try:
            text = message_data.text.encode('utf-8').decode()
        except:
            text = ''
    else:
        callback_handler(user_id = user_id, chat_id = chat_id, article_progress_id =
callback_data)
        text = ''
        return 'ok'

    if text == '/help':
        response_text = 'TODO WRITE HELP TEXT'
        bot.sendMessage(chat_id=chat_id, text=response_text)
        return 'ok'

    if text == '/start':
```

```
                try:
                    init_user(user_id)
                    response_text = 'You were added to database'
                except BaseException as e:
                    print(e)
                    response_text = 'You are already in db'
                bot.sendMessage(chat_id=chat_id, text=response_text, reply_to_message_id=msg_id)
                return 'ok'

            if pattern.match(text):
                urls = text.split()
                if(len(urls) > 1):
                    bot.sendMessage(chat_id=chat_id,  text='Wait  a  few  moments,  classification
procedure may take some time...')
                    urls = urls[1:]
                    result = articles_load_handler(urls, user_id)

                    response_text = ''
                    for res in result:
                        formatted_result = res.link + ' - ' + res.article_class + '\n'
                        response_text = response_text + formatted_result

                    if(len(result) == 0):
                        bot.sendMessage(chat_id=chat_id,  text='You  didn\'t  submit  any  articles,  or
their links are incorrect. Please, try again')
                    else:
                        bot.sendMessage(chat_id=chat_id, text='I suggest such classification')
                        bot.sendMessage(chat_id=chat_id, text=response_text)
                return 'ok'


            if text == '/update_progress':
                progress_update_dialog_handler(user_id = user_id, chat_id = chat_id, msg_id = msg_id)
                return 'ok'

            if text == '/get_prioritites':
                get_prioritites_handler(chat_id, user_id)
                return 'ok'

            set_prio_pattern = re.compile("^\/set_priorities.*")
            if set_prio_pattern.match(text):
                set_prioritites_handler(chat_id, user_id, text)
                return 'ok'

            if text == '/get_recommendation':
                suggest_article(chat_id, user_id)
                return 'ok'

            if user_id in pending_progress_updates:
                try:
                    progress_update_input_handler(chat_id, user_id, text)
                except BaseException as e:
                    print(e)
                    bot.sendMessage(chat_id=chat_id, text='Sorry, your message has incorrect format or
it doesn\'t match any command. Try again.')
                return 'ok'

            print('no handler was triggered')
            return 'no handler was triggered'

        def init_user(user_id):
            db_module.add_user_id(user_id)
            categories_default_order = ['IT','mathematics', 'history', 'economics']
            class_reports = [db_module.UserProgressByClassModel(user_id = user_id,
                                                                article_class                          =
categories_default_order[index],
                                                                user_class_priority = index + 1,
                                                                class_size = 0,
                                                                class_progress = 0.0)
                                                                for             index             in
range(len(categories_default_order))]
            db_module.add_class_user_reports(class_reports)
            return

        # handlers can be optimized later (read ORM) and moved somewhere else (partly)
        def articles_load_handler(urls, user_id):
            old_urls_list = []
            old_articles = db_module.get_articles()
            for article in old_articles:
                old_urls_list.append(article.link)
```

```
            urls_to_add = [url for url in urls if not url in old_urls_list]
            classification_results = []
            classification_results = classify_sites(urls_to_add)

            articles_to_add = []
            for result in classification_results:
                article = db_module.ArticleModel(link = result[0], article_class = result[1], size =
result[2])
                articles_to_add.append(article)

            db_module.add_articles(articles_to_add)

            submitted_articles = [article for article in (old_articles + articles_to_add) if
article.link in urls]

            article_reports = []
            try:
                for article in submitted_articles:
                    report = db_module.UserProgressByArticleModel(article_link = article.link, user_id
= user_id, article_progress = 0.0)
                    article_reports.append(report)

                db_module.add_article_user_reports(article_reports)
            except BaseException as e:
                print(e)
            return submitted_articles

        # handler for '/update_progress' command
        def progress_update_dialog_handler(user_id, chat_id, msg_id):
            user_reports = db_module.get_article_user_reports(user_id = user_id)
            pending_progress_updates[user_id] = {'article_progress_id': None}
            buttons = []
            for user_report in user_reports:
                buttons.append([telegram.InlineKeyboardButton(text=user_report.article_link,
                    callback_data=str(user_report.user_article_progress_id))])

            markup = telegram.InlineKeyboardMarkup(buttons)
            bot.sendMessage(chat_id=chat_id, text='Choose an article progress of which you\'d like to
update',
                            reply_markup=markup, reply_to_message_id=msg_id)
            return

        def callback_handler(user_id, chat_id, article_progress_id):
            if user_id in pending_progress_updates:
                pending_progress_updates[user_id] = {'article_progress_id': article_progress_id}

                bot.sendMessage(chat_id=chat_id,
                    text='Please, enter your progress in percentage for the chosen article. Estimate
your progress from 0 to 100.')
            return

        # updates user progress for a chosen article when the user inputs percentage
        def progress_update_input_handler(chat_id, user_id, text):
            progress = int(text)
            if progress > 100 or progress < 0:
                raise Exception('value is out of range')
            else:
                db_module.update_article_user_report(
                    user_article_progress_id                                                    =
pending_progress_updates[user_id]['article_progress_id'],
                    progress = round(progress/100, 2))
                bot.sendMessage(chat_id=chat_id, text='Your progress was successfully updated')
                del pending_progress_updates[user_id]
                # improve notification message
            return

        def get_prioritites_handler(chat_id, user_id):
            class_reports = db_module.get_class_user_reports(user_id)
            bot.sendMessage(chat_id=chat_id, text='Your priorities are:')
            response_text = ''
            for report in class_reports:
                response_text    =    response_text    +    report.article_class    +    '    -    '    +
str(report.user_class_priority) + '\n'
            if(len(response_text)):
                bot.sendMessage(chat_id = chat_id, text = response_text)
            else:
                bot.sendMessage(chat_id = chat_id, text = 'no prio')
            return
```

```
        def set_prioritites_handler(chat_id, user_id, text):
            topics = text.split()[1:]
            response_text = ''
            try:
                db_module.set_user_priorities(user_id = user_id, ordered_topics = topics)
                response_text = 'Your priorities were successfully updated!'
            except BaseException as e:
                print(e)
                response_text = 'Looks like your input was incorrect.' +\
                    'Please input all topics in order from the least important topic to the most.'
            bot.sendMessage(chat_id = chat_id, text = response_text)
            return

        def suggest_article(chat_id, user_id):
            recommended_category = determine_recommended_category(user_id)
            articles_to_suggest  = []
            bot.sendMessage(chat_id = chat_id, text = 'I suggest you read something from ' +
recommended_category + ' topic. Wait a sec... I will find something for you')
            if len(recommended_category) == 0:
                bot.sendMessage(chat_id = chat_id, text = 'Looks like you read all articles. Well done!
Load new articles!')
            else:
                articles          =          db_module.get_article_user_reports_for_category(user_id,
recommended_category)
                articles_to_suggest = [article for article in articles if article.article_progress <
1.0]

            if(len(articles_to_suggest) == 0):
                bot.sendMessage(chat_id = chat_id, text = 'Looks like you read all articles. Well done!
Load new articles!')
                return
            else:
                result_article = articles_to_suggest[random.randint(0, len(articles_to_suggest) - 1)]
                bot.sendMessage(chat_id = chat_id, text = "I recommend you this article:")
                bot.sendMessage(chat_id = chat_id, text = result_article.article_link)
            return

        def determine_recommended_category(user_id):
            progress_reports = db_module.get_user_class_progress(user_id)
            max_class_size = 1
            user_statistics = []
            for report in progress_reports:
                user_statistics.append({
                    'unread_words': round(report.class_size * (1.0 - report.class_progress)),
                    'category': report.article_class,
                    'priority': report.user_class_priority
                })
                if report.class_size > max_class_size:
                    max_class_size = report.class_size

            winner_rating = 0
            winner_category = ''
            coefs = range(1, len(categories) + 1)[::-1]
            for stat in user_statistics:
                rating = (coefs[stat['priority'] - 1] * stat['unread_words']) / max_class_size
                if rating > winner_rating:
                    winner_rating = rating
                    winner_category = stat['category']

            return winner_category

        @app.route('/db_init', methods=['GET', 'POST'])
        def initialize_db():
            try:
                db_module.db.create_all()
                return 'DB init SUCCESS'
            except BaseException as e:
                print(e)
                return 'DB init FAIL'

        @app.route('/delete_webhook', methods=['GET', 'POST'])
        def delete_webhook():
            s = bot.delete_webhook()
            if s:
                return 'webhook is removed'
            else:
                return 'YOU FAILED TO REMOVE WEBHOOK'

        @app.route('/set_webhook', methods=['GET', 'POST'])
        def set_webhook():
```

```
            url_for_setting = 'https://api.telegram.org/bot{0}/setWebhook?url={1}'.format(TOKEN, URL
+ '/' + TOKEN)
            result = get(url_for_setting)

            return result.text

        @app.route('/')
        def index():
            return 'HI'

        if __name__ == '__main__':
            # database_app = app
            print('Main app is running')
            app.run()
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу магістра

на тему: «Інформаційна технологія для класифікації наукових текстів на основі методу модифікованої логістичної регресії»

студента групи 122м-19-1 Массалітіна Дмитра Дмитровича

Керівник економічного розділу

доцент каф. ПЕП та ПУ, к.е.н. **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Диплом_Массалітін.doc	Пояснювальна записка до магістерської роботи. Документ Word.
Диплом_ Массалітін.pdf	Пояснювальна записка до магістерської роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_ Массалітін.pptx	Презентація до магістерської роботи