МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____Є. О. *Давиденко*

«17» ____лютого___2023 р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КЛАСИФІКАЦІЇ TELEGRAM-КОМЕНТАРІВ ДЛЯ АНАЛІЗУ НАСТРОЇВ ТА ВИЗНАЧЕННЯ СУСПІЛЬНОЇ ДУМКИ

Спеціальність «Інженерія програмного забезпечення»

121 – КРМ.1 – 608м.22150801

Студент _____*В.С. Раленко*

«17» __лютого___ 2023р.

Керівник PhD, старший викладач _____*К.О. Антіпова*

«17» ___лютого__ 2023р.

Консультант, д-р. біол. наук, проф. _____*Л. І. Григор'єва*

«17»____лютого__ 2023р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. *Давиденко*

«_____» _____ 20__ р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

Видано студенту групи 608м факультету комп'ютерних наук

Раленко Віктор Сергійович

1. Тема кваліфікаційної роботи:

Програмне забезпечення класифікації Telegram-коментарів для аналізу настроїв та визначення суспільної думки.

Затверджена наказом по ЧНУ від «03» листопада 2022 р. № 200

2. Строк представлення кваліфікаційної роботи «24» лютого 2023 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні:

очікуваним результатом є програмне забезпечення для класифікації Telegram-коментарів.

4. Перелік питань, що підлягають розробці:

аналіз предметної області, порівняльний аналіз аналогів; визначення вимог та функціоналу системи; моделювання, проектування програмного застосунку; кодування та тестування застосунку.

5. Перелік графічних матеріалів:

слайди презентації

6. Завдання до спеціальної частини:

Охорона праці та безпека у надзвичайних ситуаціях.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Григор'єва Л. І.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи PhD, старший викладач Антіпова К.О._____
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

Раленко В.С.

_____
(прізвище, ім'я, по батькові студента)

_____
(підпис)

Дата видачі завдання «22» грудня 2022р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Програмне забезпечення класифікації Telegram-коментарів для аналізу настроїв та визначення суспільної думки.

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРМ	12.09.2022	16.09.2022	виконано
2.	Складання календарного плану КРМ	17.09.2022	19.09.2022	виконано
3.	Огляд літератури за темою роботи	20.09.2022	21.10.2022	виконано
4.	Аналіз предметної області	21.10.2022	07.11.2022	виконано
5.	Розробка проєктних рішень	08.11.2022	25.11.2022	виконано
6.	Моделювання та конструювання ПЗ	26.11.2022	02.12.2022	виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	03.12.2022	13.01.2023	виконано
8.	Розробка спеціальної частини з охорони праці	14.01.2023	29.01.2023	виконано
9.	Відгук керівника КРМ	30.01.2023	31.01.2023	виконано
10.	Оформлення КРМ та презентації	01.02.2023	12.02.2023	виконано
11.	Попередній захист	13.02.2023	13.02.2023	виконано
12.	Рецензування	14.02.2023	15.02.2023	виконано
13.	Завершення оформлення КРМ та презентації	16.02.2023	17.02.2023	виконано
14.	Захист кваліфікаційної роботи	24.02.2023	24.02.2023	виконано

Розробив студент Раленко Віктор Сергійович _____

«__» _____ 2023р.

Керівник роботи PhD, ст. викладач Антіпова Катерина Олександрівна _____

«__» _____ 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
«Програмне забезпечення класифікації Telegram-коментарів для аналізу настроїв та визначення суспільної думки»
Студент 608м гр.: Раленко Віктор Сергійович
Керівник: PhD, старший викладач Антіпова К.О.

Актуальність роботи в тому, що месенджер Telegram набирає все більше популярності у світі та стає все більш привабливим для компаній, які розглядають месенджер як засіб для просування реклами своїх товарів та як маркетплейс. Стейкхолдерам важливо отримувати зворотній відгук від потенційної аудиторії. Велику частину зворотнього відгуку можна отримати за допомогою аналізу коментарів. Завдяки програмному забезпеченню класифікації коментарів, адміністратори та власники телеграм каналу зможуть автоматизовано отримувати статистику настроїв коментаторів.

Об’єктом дослідження є процес класифікації коментарів, отриманих з різних публічних каналів мессенджеру Telegram.

Предметом дослідження є алгоритми text mining.

Метою дослідження є розширення функціоналу Telegram для стейкхолдерів, які зацікавлені у використанні статистики, отриманої шляхом аналізу суспільної думки.

Завданням кваліфікаційної роботи є створення програмного забезпечення для класифікації текстових даних за настроєм з найточнішим алгоритмом серед наявних.

У першому розділі проведено аналіз предметної сфери маркетингу за допомогою соціальних мереж, та сформовано специфікацію вимог до програмного забезпечення. У другому розділі розглянуто моделі та підходи для обробки природньої мови та методи побудування класифікаторів. У третьому розділі було обрано технології, мови та компоненти програмного забезпечення, а також розроблено архітектуру. У четвертому розділі виконано програмну реалізацію класифікатору Telegram-коментарів.

Результатом є сервіс, який містить в собі елементи з різними алгоритмами класифікації, методом тестування обрано один як найточніший. Його можна використовувати як допоміжний сервіс так і незалежно від іншого коду.

Містить 80 сторінок, 19 рисунків, 3 таблиці та 25 джерел посилань.

ABSTRACT

of the Master's Thesis

«Telegram comment classification software for sentiment analysis and public opinion analysis»

Student of group 608m: Ralenko Victor Sergiyovich

Supervisor: PhD, Senior Lecturer Antipova K.O.

The relevance of the work is that the Telegram messenger is gaining more and more popularity in the world and is becoming more and more attractive for companies that consider the messenger as a means of promoting their products and as a marketplace. It is important for stakeholders to receive feedback from potential audiences. Much of the feedback can be obtained by analyzing comments. Thanks to the comment classification software, the administrators and owners of the Telegram channel will be able to automatically obtain statistics of the commenters' sentiments.

The object of the research is the process of classifying comments received from various public channels of the Telegram messenger.

The subject of research is text mining algorithms.

The purpose of the research is to expand the functionality of Telegram for stakeholders who are interested in using statistics obtained by analyzing public opinion.

The task of the qualification work is to create software for classifying text data by mood with the most accurate algorithm among the available ones.

In the first section, an analysis of the subject area of marketing with the help of social networks was carried out, and a specification of requirements for the software was formed. The second chapter deals with models and approaches for natural language processing and methods of building classifiers. In the third chapter, technologies, languages and software components were chosen, and the architecture was developed. In the fourth section, the software implementation of the Telegram comment classifier is performed.

The result is a service that contains elements with different classification algorithms, one of which was chosen as the most accurate by testing. It can be used as an auxiliary service and independently of other code.

Contains 80 pages, 19 figures, 3 tables and 25 reference sources.

ЗМІСТ

ВСТУП

Месенджер Телеграм набирає популярності у світі за кількістю користувачів та розширює свої можливості на шляху до перетворення на повноцінну соціальну мережу. Коментарі є дуже зручним способом виражати свою думку щодо певних подій, ситуацій, явищ та інших соціальних факторів. Для перетворення телеграму на повноцінну соціальну мережу, попри велику кількість відходжень від класичного розуміння терміну «мессенджер», потрібно ще значне розширення функціоналу в бік загальноприйнятих можливостей соціальних мереж.

Соціальні мережі мають інструменти для аналізу різних статистичних даних, таких як наприклад активність користувача, визначення його уподобань, тематики за якими найчастіше проявляється активність користувача, його публічні повідомлення на власній сторінці. Але найбільше ресурсів тратиться на аналіз коментарів. На даний час, месенджер Телеграм має такі інструменти як групові чати, закриті та доступні усім бажаючим «канали», які виглядають як певні групи, де адміністратори можуть публікувати пости у вигляді повідомлень в канали або чати. Телеграм канали створюються для різних цілей, наприклад для об’єднання людей в групи за спільною тематикою : інтернет магазини, новинні видавництва, середовище авторів аудіо та відеоконтенту тощо.

Важливою частиною зворотної реакції користувачів які підписані на телеграм канал є емоційні реакції та коментарі. Зазвичай саме коментарі є найпопулярнішим способом вираження своєї думки в відведеному місці під певним повідомленням. Аналогом Телеграм каналів в якості платформи для публічних обговорень є соціальна мережа Twitter. На даний час вона має набагато більше функціоналу в порівнянні з звичайними телеграм каналами як для користувачів, так і для власників акаунтів.

В більшості випадків великі телеграм канали отримують певний прибуток за пости або продаж товарів. Рекламодавцям важливо отримувати зворотній

зв’язок про якість наприклад товару, послуги, або якість власне реклами, але в Телеграмі відсутня статистика коментарів яка дозволяла б аналізувати суспільну думку щодо певних постів.

В даній кваліфікаційній роботі розроблено програмне забезпечення, яке може бути вбудоване як модуль в Телеграм збору статистики коментарів **для** власник**ів** телеграм каналів. Це збільшило б функціонал Телеграму як маркетплейсу, шляхом автоматизації роботи спеціалістів з аналізу коментарів. Для цього варто використовувати алгоритми text mining для майбутнього навчання нейронних мереж і наступної категоризації майбутніх коментарів.

Об’єктом дослідження **є** процес класифікації коментарів, отриманих з різних публічних каналів мессенджеру Telegram**.**

Предметом дослідження є алгоритми text mining.

Метою дослідження є розширення функціоналу Telegram для стейкхолдерів, які зацікавлені у використанні статистики, отриманої шляхом аналізу суспільної думки.

Завданням кваліфікаційної роботи є створення програмного забезпечення для класифікації текстових даних за настроєм з найточнішим алгоритмом серед наявних.

Апробація результатів КРМ: Результат роботи було представлено на Всеукраїнській науково-практичній конференції молодих вчених, аспірантів і студентів «Інформаційні технології та інженерія» (7-10 лютого 2023 р., ЧНУ ім. Петра Могили).

Публікації: Результати роботи представлено тезами доповіді: Раленко В. С., Антіпова К. О., Програмне забезпечення класифікації Telegram-коментарів для аналізу настроїв та визначення суспільної думки. Інформаційні технології та інженерія : тези доп. Всеукр. наук.-практ. конф. молодих вчених, аспірантів і студентів. Миколаїв, 7-10 лютого 2023 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2023. С. 104 – 106.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ

1.1 Соціальні мережі як інструмент маркетингу

Однією з основних цілей використання соціальних мереж у маркетингу є ефективна комунікація, популяризація та реклама продуктів компанії. Ці компанії використовують соціальні мережі щоб залучити більше клієнтів. Це єдина форма маркетингу, яка може демонструвати кожний етап покупки. Маркетинг через соціальні медіа також має інші переваги. З перших 10 факторів, які співвідносяться з сильним пошуком Google Organic, 7 є залежними від соціальних мереж. Це означає, що якщо бренди менш активні або зовсім неактивні у соціальних мережах, вони, як правило, мають меншу кількість пошуків в Google Search. У той час, як платформи, такі як Twitter, Facebook, та Google+, мають більшу кількість щомісячних користувачів, на основі мобільних платформ для обміну інформацією на основі візуальних медіа, однак, Garner більш високий рівень взаємодії та зареєстрував найшвидший ріст і змінили способи, якими користувалися споживачам з вмістом бренду.[1] Instagram має швидкість взаємодії 1,46 %, що щомісяця на 130 мільйонів користувачів, на відміну від Twitter, який має швидкість взаємодії 0,03 % з середнім 210 мільйонами щомісячних користувачів.[2] На відміну від традиційних засобів масової інформації, які часто є недоступним для багатьох компаній, стратегія соціальних медіа (SMM) не вимагає астрономічного бюджету.

Набір засобів SMM досить великий, адже він покликаний зацікавити не «пошукових роботів», а живих людей.[3] Головне завдання — вписатися в систему тієї соціальної мережі, в якій проводиться рекламна кампанія.

Необхідно провести таку кампанію, яка підвищить інтерес до сайту з боку максимальної кількості учасників соціальної мережі, одночасно не викликаючи нарікань з боку адміністрації ресурсу. Головним чином використовується публікація матеріалів, в яких будуть зацікавлені користувачі інтернету.

Для проведення компанії необхідно мати високий рівень майстерності для того щоб не бути включеним в число спамерів. Для зменшення ймовірності бути в списку спамерів потрібно дотримуватись таких правил:

- потрібно стати «своїм» у співтоваристві;
- необхідно розповсюджувати корисну інформацію і не тільки зі своїх сайтів;
- якщо не можете стати своїм у чужому співтоваристві — **створіть новий контент за обраною тематикою;**
- контент має бути цікавим навіть тоді, коли це лише рекламна інформація;
- якщо ви опублікуєте суперечливу інформацію, то пам'ятайте, що у вас з'являться як прихильники, так і противники;
- важливим фактором є авторитетність аккаунту.

Просування в соціальних мережах дозволяє точково впливати на цільову аудиторію, вибирати майданчики, де ця аудиторія більшою мірою представлена, і найбільш відповідні способи комунікації з нею, при цьому в найменшій мірі зачіпаючи незацікавлених в цій рекламі людей.

Переваги SMM:

- впізнаваність бренду;
- робота з цільовою аудиторією;
- прямий зворотній зв'язок;
- покриття цільової аудиторії яка не реагує на класичні види реклами;
- поєднання маркетингу та піару.

Недоліки SMM:

- необхідність бути комунікабельним і підлаштовуватися під аудиторію;
- неправильний підхід викличе негатив у споживача;
- висока конкурентність.

Існує дві основні стратегії залучення соціальних мереж як інструменти маркетингу:

Пасивний підхід. Соціальні мережі можуть стати корисним джерелом ринкової інформації та дізнатися про думку клієнтів. Блоги, контент-спільноти та форуми – це платформи, на яких люди діляться своїми відгуками та рекомендаціями щодо брендів, продуктів та послуг. Підприємства можуть використовувати та аналізувати голоси та відгуки клієнтів у соціальних мережах для маркетингових цілей; у цьому сенсі соціальні мережі є відносно недорогим джерелом інформації про ринок, яку маркетологи та менеджери можуть використовувати для відстеження та реагування на них. Наприклад, інтернет вибухнув відео та фотографіями iPhone 6 "тест на вигин", який показав, що бажаний телефон можна зігнути руками. Це створило плутанину серед клієнтів, які місяцями чекали на запуск останньої версії iPhone. Проте Apple негайно виступила із заявою, в якій говорилося, що проблема зустрічається вкрай рідко, і що компанія зробила кілька кроків, щоб зробити корпус мобільного пристрою сильнішим і надійнішим. На відміну від традиційних методів дослідження ринку, таких як опитування, фокус-групи та збір даних, які забирають багато часу та вимагають великих витрат і займають тижні[4] або навіть місяці для аналізу, маркетологи можуть використовувати соціальні мережі для отримання «живої» чи «реальної» інформації про поведінку споживачів та поглядів на бренд чи продукцію компанії.

Активний підхід. Соціальні мережі можуть використовуватися не тільки як інструменти зв'язків з громадськістю та прямого маркетингу, а й як канали зв'язку, призначені для дуже специфічної аудиторії з впливовими особами в соціальних мережах та особистостями соціальних мереж, а також як ефективні інструменти залучення клієнтів. Технології, що передують соціальним мережам, такі як широкомовне телебачення та газети, також можуть надати рекламодавцям досить точно цільову аудиторію, враховуючи, що реклама, що розміщується під час трансляції спортивних ігор або в спортивній секції газети, швидше за все, буде прочитана любителями спорту. Проте сайти соціальних мереж можуть більш точно орієнтуватися на спеціалізовані ринки.

Використовуючи цифрові інструменти, такі як Google Adsense, рекламодавці можуть орієнтувати свої оголошення на дуже специфічні демографічні дані, такі як люди, які зацікавлені у соціальному підприємництві, політичній активності, пов'язаній із певною політичною партією, або відеоіграми. Google Adsense робить це, знаходячи ключові слова в постах та коментарях користувачів соціальних мереж. Телеканалу або паперовій газеті було б важко надавати рекламу, спрямовану на цю мету.

Соціальні мережі у часто розглядаються як чудовий інструмент, що дозволяє уникнути дорогих досліджень ринку. Вони відомі тим, що вони надають короткий, швидкий і прямий спосіб досягти аудиторії через людину, яка широко відома. Наприклад, спортсмен, якого підтримує компанія, що займається спортивними товарами, також приносить свою підтримку мільйонам людей, які зацікавлені в тому, що вони роблять або як вони грають, і тепер вони хочуть стати частиною цього спортсмена завдяки своїм схваленням до цих конкретних компаній. У якийсь момент покупці відвідували магазини, щоб подивитися свою продукцію у знаменитих спортсменів, але тепер ви можете переглянути найновіший одяг знаменитостей, таких як Кріштіану Роналду, одним натисканням кнопки. Він рекламує їх вам безпосередньо через свої облікові записи в Twitter, Instagram і Facebook.

Facebook та LinkedIn є провідними соціальними мережами, де користувачі можуть націлювати свої оголошення на гіперпосилання. Гіпертаргетинг використовує як загальнодоступну інформацію профілю, а й інформацію, яку користувачі представляють, але приховують від других. Є кілька прикладів того, як фірми починають діалог із громадськістю у тій чи іншій формі, щоб покращити відносини з клієнтами. Фахівці у сфері маркетингу помітили, що керівники бізнесу, такі як Джонатан Шварц, президент і головний виконавчий директор Sun Microsystems та віце-президент McDonalds Боб Лангерт, регулярно публікують повідомлення у своїх блогах для керівників, заохочуючи клієнтів взаємодіяти та вільно висловлювати свої почуття, ідеї , пропозиції чи зауваження

щодо своїх повідомлень, компанії чи її продукции.[4] Використання впливових клієнтів (наприклад, популярних блогерів) може бути ефективним способом запуску нових продуктів або послуг. Серед політичних лідерів на своїй посаді прем'єр-міністр Індії Нарендра Моді має найбільше передплатників — 40 мільйонів, а президент США Дональд Трамп посідає друге місце з 25 мільйонами послідовників. Моді використовував соціальні медіа-платформи для обходу традиційних медіа-каналів, щоб охопити молоде та міське населення Індії, яке оцінюється в 200 мільйонів людей.

Розглянемо загальні показники, які дозволяють оцінити ефективність маркетингу соціальних мережах.

Органічний приріст передплатників (organic followers growth, OFG). OFG – це користувачі соціальних мереж, які підписалися на сторінку, паблік чи групу без рекламної комунікації з ними. Цей показник може свідчити про якість розміщеного контенту, про впізнаваність бренду/особистого бренду.

OFG може бути розрахований вручну, а в соціальній мережі Facebook його можна переглянути в розділі «Статистика» – «Підписувачі».

Органічний приріст лайків (organic likes, OL). OL — це показник позначок «подобається» до однієї публікації або публікацій, розміщених за певний проміжок часу, якщо публікація не просувається за допомогою інструментів таргетованої реклами або платного розміщення інформаційного повідомлення в спільнотах або у блогера. Цей показник може свідчити про якість контенту, що розміщується, а також про лояльність до бренду/особистого бренду. [5]

Показник органічного приросту лайків може бути розрахований вручну, з метою порівняти кількість позначок "подобається" на інший період часу (наприклад, попередній місяць) і зробити висновки про інтереси аудиторії, про найбільш популярний контент.

Отже, авторам контенту, власникам інтернет-ресурсів важливо мати статистику відгуків для подальшого планування стратегій створення продукту,

контенту, майбутнього просування контенту в рекомендації та подальшого маркетингу соціальних мереж.

Для розробки необхідного програмного забезпечення, яке буде використовувати text mining, потрібні певні попередньо класифіковані початкові дані, які будуть основою для майбутнього програмного аналізу нових даних.

Початковими даними буде виступати набір текстів, які вручну були проаналізовані та розподілені на категорії за загальним настроєм тексту. Категорії за настроєм є складовими аналізу суспільної думки. Категорії будуть відображати різноманітний характер висловлювань коментаторів. Настрої коментарів бувають позитивні, нейтральні та негативні.[6] Серед позитивних коментарів можна виділити коментарі, які заохочують до збільшення кількості створюваного контенту, створення мотивації до певних дій, збільшення натхнення від створення або переглядання поточного або майбутнього контенту, виділення позитивних особливостей побаченого споглядачами, підняття настрою від коментаторів іншим коментаторам або спостерігачам. В свою чергу негативні коментарі можуть мати прямо протилежний вплив як на оточуючих коментаторів, так і на автора контенту в вигляді демотивації до створення нового контенту, засудження певних дій, життєвих та професійних позицій, або створення цілеспрямованого хейту до певних об’єктів життєдіяльності.

1.2 Специфікація вимог до програмного забезпечення

1.2.1 ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

1.2.1.1 Призначення застосунку для якого розробляється програмне забезпечення

Призначення Telegram полягає в реалізації функціоналу спілкування між людьми, обміну різноманітною інформацією на різних платформах, таких як iOS та Android смартфони, а також персональні комп’ютери.

1.2.1.2 Межі проєкту ПЗ

Призначення програмного забезпечення в розширенні функціоналу Telegram для показу статистики суспільних настроїв власникам та адміністраторам Telegram груп.

1.2.2 ЗАГАЛЬНИЙ ОПИС

1.2.2.1 Сфера застосування

Сферою застосування програмного забезпечення, що розробляється, є один з видів складових системи швидкого обміну повідомленнями Telegram, які називаються коментарями до записів в Telegram-каналах.

1.2.2.2 Характеристики користувачів

Дане програмне забезпечення буде взаємодіяти з авторами та адміністраторами телеграм каналів. Вони мають бути зареєстрованими в Телеграм та мати необхідні права доступу до статистики. Розробники Телеграм в випадку додавання даного програмного забезпечення зможуть регулювати права доступу до даних

1.2.2.3 Загальна структура і склад системи

Система буде складатись з програмного коду, який буде обробляти вхідні текстові дані, розподіляти їх на слова та словосполучення, визначати настрій за допомогою завчасно навчених даних, виведення статистики за запитом клієнту.

1.2.2.4 Загальні обмеження

Система буде обмежуватись мовами якими написаний текст, оскільки під кожну мову якою може бути написаний коментар потрібно збирати свої навчальні дані або за допомогою перекладу початкових даних та повторного навчання так і за допомогою перекладу всіх повідомлень на англійську мову.

1.2.3 ФУНКЦІЇ СИСТЕМИ

1.2.3.1 Функція розділення даних на слова

Опис функції

Дана функція розподіляє введений рядок на слова за різними символами і перетворює рядок в масив слів.

Вхідна та вихідна інформація

Вхідна інформація – рядок.

Вихідна інформація – масив слів.

Функціональні вимоги

Рядок має для покращення роботи містити в собі правильно граматично написані слова без зайвих символів всередині них, особливо розділових знаків за якими можна розділити їх.

1.2.3.2 Функція нормалізації тексту

Опис функції

Дана функція перетворює текст в стандартизовану форму, яку він, можливо, не мав раніше. Нормалізація тексту перед зберіганням або обробкою допомагає уникнути багатьох проблем, оскільки формат введених даних гарантовано буде узгодженим. Нормалізація тексту вимагає розуміння, який тип тексту потрібно нормалізувати та як він буде оброблятися пізніше; не існує універсальної процедури нормалізації.

Вхідна та вихідна інформація

Вхідна інформація – масив слів.

Вихідна інформація – масив слів.

Функціональні вимоги

Особливих функціональних вимог немає.

1.2.3.3 Функція розділення даних на слова

Опис функції

Дана функція розподіляє введений рядок на слова за різними символами і перетворює рядок в масив слів.

Вхідна та вихідна інформація

Вхідна інформація – рядок.

Вихідна інформація – масив слів.

Функціональні вимоги

Рядок має для покращення роботи містити в собі правильно граматично написані слова без зайвих символів всередині них, особливо розділових знаків за якими можна розділити їх.

1.2.3.4 Функція видалення стоп слів з тексту

Опис функції

Тексти часто містять непотрібні слова, які використовуються лише для зв'язку інших слів у реченні. Ці слова, загалом відомі як «стоп-слова», не сприяють контексту чи змісту текстів. Вони постійно повторюються, що сприяє розумінню змісту документів. Наприклад, «і», «є» і «це» можуть бути стоп-словами. Вони непридатні для класифікації документів, тому їх потрібно видалити. Однак створення списку стоп-слів може бути складним завданням, оскільки залежить від конкретного текстового корпусу та мети обробки. Видалення стоп-слів відокремлює важливу інформацію та покращує ефективність системи.

Видалення стоп-слів із текстів допомагає визначити важливу інформацію, усуваючи присутність прийменників, сполучників та інших слів, які не мають семантичного значення.

Вхідна та вихідна інформація

Вхідна інформація – масив слів.

Вихідна інформація – масив слів.

Функціональні вимоги

Рядок має для покращення роботи містити в собі правильно граматично написані слова без зайвих символів всередині них, особливо розділових знаків за якими можна розділити їх.

1.2.3.5 Функція класифікації отриманого набору текстових даних

Опис функції

Дана функція означає, що програмне забезпечення має містити в собі класифікатор (може бути декілька), який за допомогою навчальних даних зможе

класифікувати подані масиви слів та текстові дані в необхідні класи які характеризуватимуть текстові дані з точки зору настрою, який передає текст.

Рисунок 1.1 – Процес побудови класифікатора

Для максимальної точності потрібно буде попередньо в тестовій версії застосунку зробити підбір параметрів класифікатору для збільшення точності класифікації на тестових даних, щоб мінімізувати відсоток похибки в майбутніх класифікаціях коментарів.

Вхідна та вихідна інформація

Вхідна інформація – масив слів

Вихідна інформація – масив слів

Функціональні вимоги

Масив слів який є вхідним має бути завчасно обробленим за допомогою попередньо вказаних функцій.

1.2.3.6 Функція виведення статистики

Опис функції

Дана функція виводить статистику завчасно оброблених класів за запитом.

Вхідна та вихідна інформація

Вхідна інформація – масив слів.

Вихідна інформація – масив слів, текстові або графічні дані.

Функціональні вимоги

На вхід має приходити прокласифікований масив тексту, за яким можна зібрати класову статистику. В випадку відсутності класів за текстовими даними функція не має працювати і повертати помилку, яка буде відображатись лише в debug-режимі.

1.2.4 ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

1.2.4.1 Джерела і зміст вхідної інформації

Джерелами інформації слугуватимуть бази даних с текстовими коментарями для певних каналів. Коментарі які містять в собі додатковий контент не будуть передавати його, а лише текстовий опис.

1.2.4.2 Нормативно-довідкова інформація

В даному програмному забезпечені будуть використовуватись алгоритми text mining для обробки тексту. Ключовими для роботи text mining будуть створення токенізатору. Процес побудови класифікатору текстової інформації складається з таких етапів:

а) попередня обробка тексту;

б) витяг ознак з тексту;

в) зменшення розмірності векторів ознак;

г) навчання класифікатора;

д) оцінка якості класифікатора;

е) тюнинг гіпер-параметрів алгоритму.

1.2.4.3 Вимоги до способів організації, збереження та ведення інформації

Особливих вимог до зберігання інформації немає, оскільки будуть оброблятись лише текстові дані, а спосіб їх передачі буде залежати від розробників Telegram.

Для тестової версії програмного забезпечення та для фінальної версії яка може працювати за бажанням незалежно від Telegram потрібно створити код, який буде передавати текстові дані в розроблене програмне забезпечення з певного сайту, локальної або віддаленої бази даних або з створеного всередині коду масиву з даними для класифікації. Важливо щоб всі дані які передаються в програмне забезпечення на вхід були текстовими.

1.2.5 ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Для роботи програмного забезпечення потрібні потужні сервери, потужність яких буде залежати від кількості даних які будуть поступати в програмне забезпечення та кількості користувачів яким потрібно вивести певні дані. В Telegram є в наявності різні сервери в багатьох країнах світу, та до даних серверів можна підключити програмне забезпечення.

Для тестової версії програмного забезпечення з невеликою кількістю даних достатньо звичайного стаціонарного ПК з можливістю запускати інтерпретатори та встановлювати бібліотеки до них. Для виведення зображень в тестовій версії потрібна наявність кольорового монітору який приєднаний до персонального комп’ютеру.

Зі збільшенням кількості оперативної пам’яті та при заміні процесора на більш потужний зростає продуктивність програмного забезпечення. Також з покращенням апаратного забезпечення програмне забезпечення зможе обробляти більші масиви даних які до нього входять.

1.2.6 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.2.6.1 Архітектура програмної системи

Програмна система є універсальною, вона може обробляти дані які їй передані з бази даних.

1.2.6.2 Системне програмне забезпечення

Системне програмне забезпечення повинно мати можливість передачі текстового контенту в розроблене програмне забезпечення та мати можливість коректно приймати оброблену інформацію

1.2.6.3 Мова і технологія розробки ПЗ

Для розробки програмного забезпечення буде використовуватись мова програмування Java з допомогою WEKA. WEKA дозволятиме застосовувати

різні алгоритми роботи з Big Data для завдань класифікації, кластеризації, фільтрування та ін.

1.2.7 ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

1.2.7.1 Інтерфейс користувача

Від інтерфейсу користувача потрібна можливість відображати текстові та графічні дані.

1.2.7.2 Програмний інтерфейс

Програмне забезпечення має взаємодіяти з серверами Telegram в release варіанті для подальшого виведення результатів програми в Telegram клієнти за допомогою власних протоколів клієнт-серверної взаємодії Telegram.

1.2.7.3 Комунікаційний протокол

Для розробників Telegram дана програма може працювати як сервіс, який отримує масив текстових даних та повертає вихідні дані в вигляді зображень та/або текстових даних. Також для тесту можна запускати програмне забезпечення з своїми масивами текстових даних.

1.2.8 ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.2.8.1 Доступність

В розробленому варіанті програмне забезпечення буде доступне всім користувачам персонального комп’ютера з можливістю встановлення програмного забезпечення.

1.2.8.2 Супроводжуваність

Програмне забезпечення має бути адаптованим до зміни джерел отримуваної інформації і незалежним від конкретного джерела інформації (лише формат тексту та текст).

1.2.8.3 Переносимість

Дана властивість програмного забезпечення означає, що програмне забезпечення має мінімально залежати від системи, на якій воно буде запущено.

Від системи на якій буде розгортатись програмне забезпечення потрібна лише можливість роботи з інтерпретатором Java.

1.2.8.4 Продуктивність

Програмне забезпечення має виконувати дії чітко до його завдання та тематики.

1.2.8.5 Надійність

Дана властивість програмного забезпечення вказує про необхідність стабільної роботи програмного забезпечення. Вона має бути реалізована за допомогою перевірки вхідних даних на відсутність слів.

1.2.8.6 Безпека

Програмне забезпечення не має передавати дані третім особам та не має самостійно збирати статистику його використання.

Висновки до розділу 1

Telegram набирає популярності серед користувачів інтернету. В цьому зацікавлені стейкхолдери, які можуть реалізовувати завдання месенджеру Telegram як місця для піар-компаній та розповсюдження реклами. Однією з основних цілей використання соціальних мереж у маркетингу є ефективна комунікація, популяризація та реклама продуктів компанії. Ці компанії використовують соціальні мережі щоб залучити більше клієнтів. Це єдина форма маркетингу, яка може демонструвати кожний етап покупки.

Для повноцінного перетворення Телеграм в соціальну мережу потрібно створити програмний застосунок, який буде аналізувати коментарі, та визначати суспільну думку користувачів месенджеру про певні речі та теми. Завдяки подальшому збору думок формуватиметься статистика, яка буде корисна під час ведення SMM діяльності.

2 ПРОЄКТНІ РІШЕННЯ

2.1 Моделі та підходи для обробки природної мови

Першим кроком у створенні класифікатора текстової інформації є попередня обробка текстових даних. Вона полягає в розбитті документа на лексеми – окремі слова – та видаленні стоп-слів – слів, які не несуть семантичного навантаження (наприклад, прийменників, сполучників тощо).

Усі слова, присутні в корпусі документа, утворюють словник корпусу. Таким чином, документ можна представити у вигляді двійкового вектора, де значення «1» вказує на те, що певне слово присутнє в тексті, а значення «0» означає, що його немає. Однак на практиці підхід побудови словника з усіх слів використовується рідко. Це тому, що його розмір може складатися із сотень тисяч або навіть мільйонів слів. Такі обсяги даних є дуже великими для обробки, і вони можуть негативно вплинути на точність алгоритму та сприяти його перенавчанню.

Bag-of-Words. Bag-of-Words (BoW) — це простий і гнучкий метод виділення ознак із тексту, який використовується в задачах обробки природної мови, таких як моделювання мови та класифікація документів.[7] Він представляє текст як опис появи слів у документі, відкидаючи інформацію про порядок або структуру слів. Ця модель названа так тому, що вона схожа на «мішок», де не важливий порядок слів та враховується лише те, що певні відомі слова присутні в документі.

Модель BoW дуже проста у розумінні та реалізації і пропонує універсальність, оскільки її можна адаптувати до конкретних завдань. Вона широко використовується і є успішною у задачах обробки природної мови. Тим не менш, вона має певні суттєві недоліки, такі як:

- словник: проєктування словника вимагає добре продуманого підходу, адже є деякі критичні аспекти, які впливають на функціональність

словника. Одним з них є його обсяг, який обмежує використання текстових даних в деяких випадках;

- швидкодія: моделювання розрідженого представлення є вузьким місцем за критерієм інформації та просторової складності, зокрема, коли завданням є використання мінімум інформації в великому просторі даних;
- семантика: відкидання порядку слів ігнорує контекст і, в свою чергу, значення слів в документі (семантику). Врахування контексту та сенсу можуть значно поліпшити ефективність моделі.

TF-IDF. Також для вибору ознак, а саме для побудови словника використовується підхід TF-IDF метрики. TF-IDF метрика є важливим показником важливості слова в контексті певного документа при виборі ознак для побудови словника.[8] Її дія основана на двох компонентах: розрахунок важливості слова в межах одного документа та інверсія частоти слова у всьому корпусі документів.

Перша складова обраховується за формулою:

$$F(t, d) = \frac{n_t}{\sum_k n_k} \tag{1}$$

де n_t – кількість входжень слова t в документ d;

$\sum_k$ n_k – загальна кількість слів у документі.

За допомогою другого показника зменшується кількість загальновживаних слів. Показник обраховується за формулою

$$I(t, D) = \frac{|D|}{|\{d \in D : t \in d\}|}, \tag{2}$$

де $|D|$ – кількість документів у корпусі;

$|\{d \in D : t \in d\}|$ – кількість документів з корпусу D, в яких зустрічається слово t.

Фінальна формула цього показника виглядає так:

$$T(t, d, D) = F(t, d) \times I(t, D), \quad (3)$$

де t - слово;

D - документ;

D - корпус текстів;

F(t, d) - показник частоти слова;

I(t, D) - показник оберненої частоти слова.

Отже, найбільшу вагу надають словам із високою частотою в одному документі та низькою частотою в інших. Значущість слів визначається їх здатністю характеризувати різницю між категоріями в корпусі документів. Ця метрика допомагає скласти словник значущих слів і представити документи в корпусі як числові вектори.

Word2Vec. Word2Vec - це модель компанії Google, яка представляє слова у вигляді 300-вимірних векторів, навчена на великих текстових даних. Близькі семантично слова знаходяться рядом.[9] Word2Vec відома своєю здатністю відображати близькість за семантикою слів у векторному просторі. Word2Vec - це «золотий стандарт» в обробці текстових даних з використанням глибокого навчання і використовує два алгоритми: «Continuous Bag-of-Words» та «Skip-Gram». Алгоритмічно ці моделі дуже схожі, за винятком того, що CBOW прогнозує цільові слова, використовуючи слова з контексту, тоді як skip-gram використовує зворотній алгоритм і прогнозує контекст, використовуючи цільове слово (рисунок 2.1). Модель може бути використана у 100-вимірному або 300-вимірному просторі, а також може бути дотренована під конкретну задачу на власних корпусах текстів.

Під час вирішення поставленої задачі модель word2vec у 300-вимірному просторі було дотреновано, використовуючи записи користувачів з соціальних мереж Facebook та Twitter на політичну тематику. Розмір корпусу - 3 Гб текстових даних. На рисунку 2.1 наведено сформований 300-вимірний векторний простір слів, близький за семантикою.

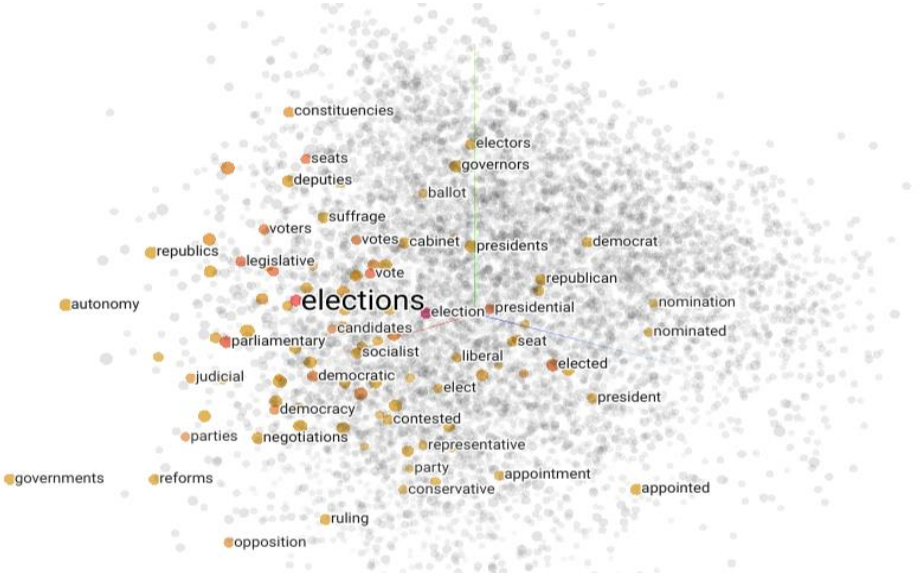

Рисунок 2.1 – Векторний простір слів

Дослідники з Стенфордського університету представили новий метод створення семантичних векторних просторів, заснований на агрегованій глобальній статистиці використання слів в документах. Модель, названа Global Vectors for Word Representation (GloVe), демонструє цікаві лінійні відносини в отриманому векторному просторі.

Далі вектори документів подаються на вхід алгоритмам машинного навчання.

Named-Entity Recognition (NER). Кожен день новини та видавництва генерують велику кількість онлайн-контенту, тому важливо мати ефективне управління ними, щоб максимально використовувати кожну статтю. Розпізнавання імен сутностей допомагає виявляти ключові особи, організації та місця які обговорюються в статтях за допомогою автоматичного сканування цілих статей. Правильне визначення тегів для кожної статті автоматично класифікує їх та забезпечує кращу якість виявлення змісту. Створені системи NER, які використовують лінгвістичні граматичні методи, а також статистичні моделі, такі як машинне навчання. Ручні системи, основані на граматиці, мають кращу точність, але вимагають довготривалої роботи досвідчених обчислювальних лінгвістів. Статистичні системи NER потребують великої кількості даних анотованих вручну.

WordNet. WordNet — це всеохоплююча та обширна лексична база даних англійської мови, яка може похвалитися великою колекцією іменників, дієслів, прикметників і прислівників, організованих у набори когнітивних синонімів, відомих як синсети. Кожен синсет втілює чітку та чітку концепцію, і ці синсети взаємопов’язані через концептуально-семантичні та лексичні зв’язки. Добре структурований склад WordNet робить його цінним інструментом для комп’ютерної лінгвістики та обробки природної мови.

WordNet — це набагато більше, ніж просто тезаурус, оскільки він групує слова на основі їх значень і пов’язує конкретні значення слів. На відміну від тезауруса, який просто групує слова зі схожими значеннями, WordNet пов’язує слова семантично, створюючи мережу слів, які семантично пов’язані одне з одним. Цей зв’язок проявляється через специфічні семантичні та лексичні зв’язки, що робить WordNet цінним інструментом для комп’ютерної лінгвістики та обробки природної мови. Більше того, WordNet не лише пов’язує форми слів, але й конкретні значення слів, дозволяючи глибше зрозуміти значення слів та їх зв’язок одне з одним. Така база корисна для вирішення завдань з опрацювання англійської мови.[10]

WordNet описує основне відносини між словами через синонімію, яка позначає одну і ту ж концепцію. Слова, які є синонімами, згруповані в невпорядкованих наборах (synsets), кожен з яких містить слова з однойменною концепцією та коротке визначення. Кожен з 117 000 наборів WordNet пов'язаний з іншими за допомогою невеликої кількості «концептуальних відносин». Крім того, synset містить коротке визначення і, в більшості випадків, одне або кілька коротких пропозицій, що ілюструють використання слів з наборів. Таким чином, кожна пара слів в WordNet унікальна. Приклад структури WordNet показано на рисунку 2.2.

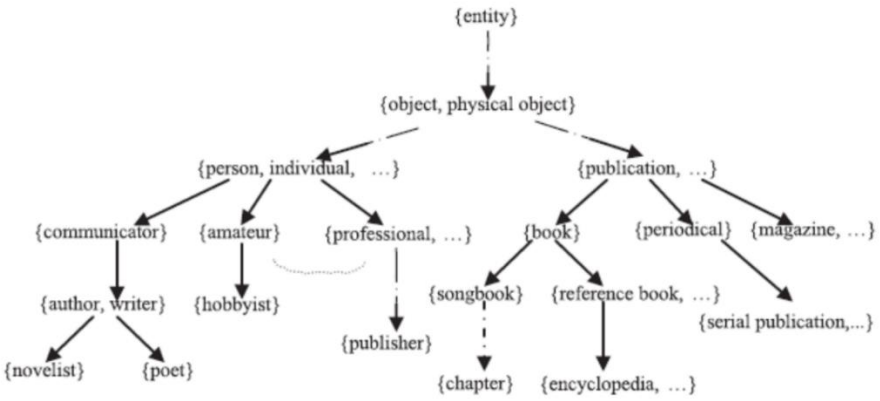

Рисунок 2.2 – Структура сегменту WordNet

Бібліотека NLTK дозволяє легко використовувати лексичну базу WordNet, враховуючи синонімію в процесі створення додаткових ознак для класифікації..

Part-Of-Speech Tagging. PoS тегер - це алгоритм, який відмічає слова у відповідності до його функції у реченні за допомогою включення до однієї з декількох категорій. В англійській мові слова можуть бути однією з восьми або дев'яти частин мови, включаючи іменники, дієслова, артиклі, прикметники, прийменники, займенники, прислівники, сполучники та вигуки.

PoS алгоритми застосовуються для ідентифікації ролей, які відіграють слова в текстовому корпусі. Вони роблять більш складні категорії, ніж ті які визначені як базові PoS, з такими тегами, як "однина-множина" або навіть більш складними мітками.[11]

PoS алгоритми визначають тип PoS слова в текстовому корпусі за його реляційною позицією у фразі, взаємодією з сусідніми словами та ідентифікацією слова. PoS алгоритми можуть бути базовані на статистичних методах, ймовірнісних моделях або правилах.

Одним з перших розроблених тегерів PoS був тегер Бріла. Він заснований на правилах та все ще широко використовується сьогодні. В даний час, існує ряд інших маркувальників PoS, таких як Stanford Log-linear Part-Of-Speech Tagger, Tree Tagger та Microsoft POS Tagger. Ці інструменти використовуються для класифікації термінів в контексті, враховуючи їх роль у фразі, співвідношення до близьких слів та за визначенням слова. Це може включати в себе визначення

частини мови, але також може бути вказано неоднозначність слова або граматичне маркування.

2.2 Методи побудови та навчання класифікаторів

Можна виділити наступні алгоритми класифікації текстових даних:

- класифікатор методом найближчих сусідів;
- байєсівський класифікатор;
- класифікатор на основі дерева рішень;
- метод опорних векторів;
- класифікатор на основі графових моделей;
- класифікація на основі асоціативних правил;
- класифікатор на основі штучних нейронних мереж.[12]

Логістична регресія – це метод побудови лінійного класифікатора, який дозволяє оцінити апостеріорні ймовірності приналежності об’єктів класам. Алгоритм мультиноміальної логістичної регресії для кожної категорії залежної змінної будує рівняння бінарної логістичної регресії.

Логістична регресія виражає модель зв’язку між залежною та незалежною змінними у вигляді формули:

$$P\{X_1, \dots, X_P\} = \frac{e^{\hat{Y}}}{1 - e^{\hat{Y}}}, \tag{4}$$

де Y – залежна змінна;

X – регресори;

$\hat{Y}$– логит (функція логістичного закону розподілення).

Для покращення узагальнюючої здібності моделі та запобіганню її перенавчання використовується регуляризація, яка штрафує модель за складність. У системі автоматичної класифікації текстової інформації використовується L1-регуляризація та L2-регуляризація.[9] З’ясовано, що на великорозмірних даних логістична регресія дає кращі результати ніж наївний

байесівський класифікатор.[13] Такі гіперпараметри як регуляризація та крок градієнтного спуску, використовуються для підбору оптимальних параметрів моделі.

Наївний байєсівський класифікатор є простим та швидким ймовірнісним методом, який оснований на припущеннях про незалежність виникнення слів у тексті. Його перевагою є невелика потреба даних для навчання, а також висока швидкість побудови моделі. Він використовує підхід апостеріорного максимуму для визначення найбільш вірогідного класу:

$$c_{map} = arg\ max \frac{P(c)P(c)}{P(d)}, \quad (5)$$

де c_{map} – найбільш вірогідний клас;

$P(c) -$ ймовірність зустріти документ d серед всіх документів класу c;

$P(c)$ –безумовна ймовірність зустріти документ класу c в корпусі документів;

$P(d) -$ безумовна ймовірність документа d в корпусі документів.

Наївний байєсівський алгоритм є ефективним методом для побудов класифікатору для даних невеликого розміру.

Ідея методу опорних векторів полягає у побудові гіперплощині, що розділяє класи з максимальним зазором. На початку будуються дві паралельні гіперплощини по обидві сторони від гіперплощини, яка розділяє класи, як зображено на рисунку 2.3. Алгоритм припускає, що чим більша відстань між паралельними площинами, там краще буде точність класифікатора.

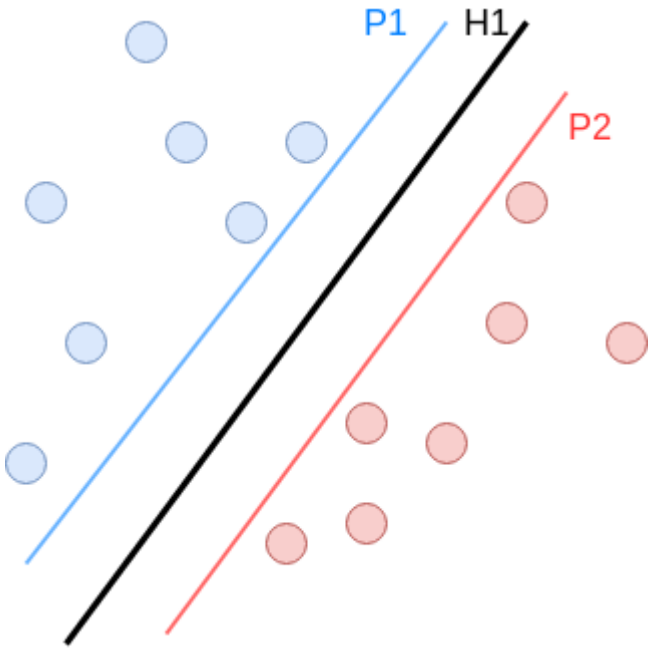

Рисунок 2.3 — Дві паралельні гіперплощини P_1 та P_2 та гіперплощина H_1,що розділяє класи

Гіперплощину, що розділяє класи можна записати у вигляді множини точок:

$$w * x - b = 0, \tag{6}$$

де w - нормальний вектор до гіперплощини, що розділяє класи;

x - вектор розмірності p;

b - допоміжний параметр.[14]

Параметр $\frac{b}{||w||}$ визначає зміщення гіперплощини від початку координат вздовж нормалі w. Для того, щоб відстань між гіпер площинами була максимальною, треба вирішити задачу оптимізації:

$$||w|| \rightarrow min \tag{7}$$
$$y \quad (w \cdot x \quad - b) \geq 1,$$

де w - нормальний вектор до гіперплощини, що розділяє класи;

x - вектор розмірності p;

y - приймає значення 1 або -1 в залежності від того, до якого класу належить x;

b - допоміжний параметр.

Класифікатори на основі методу опорних векторів (SVM) не вимагають зменшення розмірності простору ознак, оскільки вони добре масштабуються та мають хорошу стійкість до перенавчання.

Метод опорних векторів (SVM) є ефективним способом класифікації текстової інформації. Однак основним його недоліком є те, що під час пошуку гіперплощини, яка розділяє класи, розглядаються лише граничні точки. У багатьох випадках також неможливо знайти оптимальну гіперплощину, і в цьому випадку класифікатор будує гіперплощину, мінімізуючи кількість помилок і максимізуючи відстань між класами.

Алгоритм k-NN – метод найближчих сусідів є метричним алгоритмом класифікації. Він класифікує об`єкт до того класу шляхом голосування k найближчих сусідів. Кожний сусід $x_{i,u}$ голосує за то, щоб віднести об’єкт u до свого класу $y_{i,u}$. Алгоритм обирає клас, який набрав найбільшу кількість голосів.

Алгоритм має кілька недоліків, одним з яких є його нестійкість до викидів. Викид — це об’єкт, який розташований поблизу об’єктів інших класів і буде неправильно класифікований сам, а також неправильно класифіковує всі об’єкти, які будуть поруч із ним. Крім того, алгоритм не має параметрів, які можна налаштувати для конкретної вибірки.

Алгоритми ансамблів - це мета-алгоритми, які об'єднують кілька методів машинного навчання в одну прогностичну модель, щоб зменшити дисперсію (variance), зміщення (bias) або поліпшити прогнози (stacking).

Методи ансамблів можна розділити на дві групи:

- послідовні ансамблеві методи, в яких базові моделі генеруються послідовно (наприклад, AdaBoost);
- паралельні методи ансамблю, де базові моделі генеруються паралельно (наприклад, випадковий ліс).

Послідовний метод дозволяє додавати нові моделі в послідовність, якщо виявляється, що раніше використані моделі недостатньо точні. Це також

дозволяє додатково припустити в доробку відомості про дані, зокрема, використовуючи кореляцію між моделями та даними. В кінцевому етапі послідовний метод представляє собою гібридну модель, яка має високу точність та є здатною підтримувати баланс між перенавчанням та недонавчанням.

Унікальність паралельних методів в порівнянні з послідовними полягає в тому, щоб навпаки використовувати незалежність між базовими моделями, оскільки помилка може бути різко зменшена шляхом усереднення.

Параллельні методи призначені для агрегації бутстрапів. Один з способів зменшити дисперсію оцінок - агрегувати разом кілька оцінок. Bagging використовує вибірку бутстрапа для отримання підмножин даних для навчання базових моделей. Для агрегування їх результатів використовується голосування для класифікації та усереднення для регресії.

Бустингові алгоритми - це алгоритми, які дозволяють перетворювати слабкі моделі в сильні за допомогою послідовного будування послідовностей нескладних моделей, надання вагового коефіцієнту даним і об'єднання результатів за допомогою зваженої більшості голосів. Моделі будуються таким чином, щоб краще представляти дані, які помилково класифіковані в попередніх раундах.

Бустінгові алгоритми є найбільш ефективними для задач класифікації. Бустінг – це алгоритм послідовної побудови композиції алгоритмів машинного навчання, коли кожний наступний алгоритм намагається компенсувати недоліки композиції усіх попередніх композицій. Для вирішення задачі класифікації використовувався алгоритм градієнтного бустінгу, який будує модель у вигляді суми дерев:

$$f(x) = h_0 + v \sum_{j=1}^{M} h_j(x), \tag{8}$$

де h_0 – деяка константна модель;

$v \in (0,1]$ - параметр, що регулює швидкість навчання та вплив окремих дерев на всю модель;

$h_j(x)$ - регресійні дерева рішень.

Алгоритм градієнтного бустінгу реалізований у багатьох бібліотеках машинного навчання, зокрема Xgboost (для таких мов програмування, як C++, Java, Python, R та Julia), а також LightGBM (для таких мов програмування, як Python та R). Для алгоритмів бустингу важливо обмежувати глибину дерева або кількість листків на останньому рівня для запобігання перенавчанню алгоритму.

Розробники IBM Watson створили алгоритм класифікації тексту з використанням моделей seq2seq в рамках одного з найбільших проєктів обробки природної мови. Автори пропонують відійти від простих статистичних моделей і графових структур і замість цього застосувати глибоке навчання з використанням рекурентних нейронних мереж. Причиною використання рекурентних нейронних мереж замість прямих або згорткових нейронних мереж для текстових даних є унікальні характеристики вхідних даних. Повторювані мережі мають здатність «запам’ятовувати» контекст речення, змінювати його та вирішувати, який контекст більше не потрібний і його можна видалити, а також чи слід його розширити новими даними. Для вирішення задачі класифікації використовуються дві рекурентні мережі. Одна «закодує» вхідний текст документа в певний n-вимірний простір, а інша візьме закодований вхідний текст і визначить клас тексту.

Після створення класифікаторів, важливим кроком є вибір найефективнішого з них. Зазвичай використовується проста метрика - відсоток документів, для яких класифікатор відповідає вірно. Однак, такий підхід може бути некорректним у випадку нерівномірного розподілу документів у корпусі, в якому одні класи можуть бути переважно представлені. У такому випадку, класифікатор може працювати добре лише для цих класів, заперечуючи рішення для інших. Щоб запобігти такому результату, важливо використовувати

збалансовані дані. Однак, такий підхід може негативно вплинути на кількість інформації, доступної системі.

Особливістю цього методу є його підхід end-to-end навчання, передбачає конвертацію слів у word2vec вектори та відправлення усіх корпусів у систему для навчання. Немає необхідності створювати статистичні метрики або проводити експерименти з представленням слів. Все, що потрібно, - це "налаштувати" конфігурацію мережі, визначивши потрібну кількість прихованих слоїв та активаційну функцію. Це не вимагає великих зусиль, оскільки дві найбільш популярні активаційні функції - сигмоїдальна та ReLU (rectified linear unit) - вже доступні.

Створення текстового класифікатора починається з визначення кількості тренувальних даних. Однією з найпроблематичніших частин процесу тренування класифікатора в реальних застосуваннях є вирішення питання про достатність тренувальних даних. Велика кількість прикладів кожного класу, сотні або навіть тисячі, є необхідною для створення високопродуктивного класифікатора для багатьох проблем й алгоритмів. В багатьох реальних контекстах зустрічаються великі групи категорій. Тому, якщо є достатньо часу для реалізації класифікатора, велика частина його повинна бути витрачена на збір тренувальних даних.[15]

2.3 Методи оцінки ефективності класифікатору

Для оцінки ефективності класифікатору використовують метрики точності (precision) та повноти (recall) системи. Вони використовуються як базові метрики для F-метрики.

Точність системи у межах класу – це відсоток документів, які насправді належать даному класу, відносно усіх документів, які система віднесла до цього класу.

Повнота системи – це процент документів, які за результатами системного пошуку належать класу, відносно всіх документів цього класу в тестовій виборці.

Значення точності та повноти системи легко розрахувати на основі таблиці контингентності (таблиця 2.1), яка будується для кожного класу:

Таблиця 2.1 – Таблиця контингентності

		Експертна оцінка	
		Позитивна	Негативна
Оцінка системи	Позитивна	TP	FP
	Негативна	FN	TN

TP (true positive) – істинно-позитивне рішення, FP (false positive) – хибно-позитивне рішення, FN (false negative) – хибно-негативне рішення, TN (true negative) – істинно-негативне рішення.

Точність системи розраховується наступним чином:

$$p = \frac{P}{P + F}, \tag{9}$$

де p - точність системи;

P - кількість істинно-позитивних рішень;

F - кількість хибно-позитивних рішень.

Повнота системи розраховується наступним чином:

$$r = \frac{P}{P + N}, \tag{10}$$

де p - повнота системи;

P - кількість істинно-позитивних рішень;

F - кількість істинно-негативних рішень.

Чим більше точність та повнота системи, тим краща ефективність класифікатора. Створення текстового класифікатора в реальному світі вимагає балансу між кількістю тренувальних даних та точністю системи. Існує проблема, коли занадто мало тренувальних даних призводить до зниження точності класифікатора, але з іншого боку, занадто багато даних також може призвести до зменшення точності. Тому важливо шукати компроміс між кількістю

тренувальних даних та точністю системи, щоб досягти найкращого відповідного результату. Найчастіше використовується F-міра , яка є гармонійним середнім між точністю та повнотою:

$$F = 2\frac{p \times r}{p + r}, \tag{11}$$

де p - точність системи;

r - повнота системи.

Якщо розробник хоче надавати перевагу або точності, або повноті системи, то метрика модифікується наступним чином:

$$F = (\beta^2 + 1)\frac{p \times r}{\beta^2 p + r}, \tag{12}$$

де β^2 - додатковий параметр;

p - точність системи;

r - повнота системи.

Якщо β знаходиться у межах $0 < \beta < 1$, то системи надає перевагу точності системи, якщо $\beta > 1 -$ то повноті системи.

Для оцінки якості класифікатора у системі автоматичної класифікації текстової інформації використовується F-міра.

Налаштування гіпер-параметрів алгоритму необхідно для запобігання перенавчанню та недонавчанню. Відомо, що похибка класифікатору складається з таких складових:

$$e = b^2 + v + \delta^2, \tag{13}$$

де $b^2 -$ квадрат зміщення (середня похибка по всім наборам даних);

$v -$ дисперсія (варіативність помилки);

σ^2 - помилки шуму.

Помилку шуму усунути неможливо, але можна зменшити зміщення або дисперсію. На практиці відомо, що неможливо зменшити зміщення та дисперсію

одночасно, так як при збільшенні складності моделі збільшується дисперсія, але зменшується зміщення і навпаки. Важко знайти баланс між ними, тому розробники намагаються знайти компроміс між цими значеннями. Графік залежності між зміщенням та дисперсією можна побачити на рисунку 2.4:

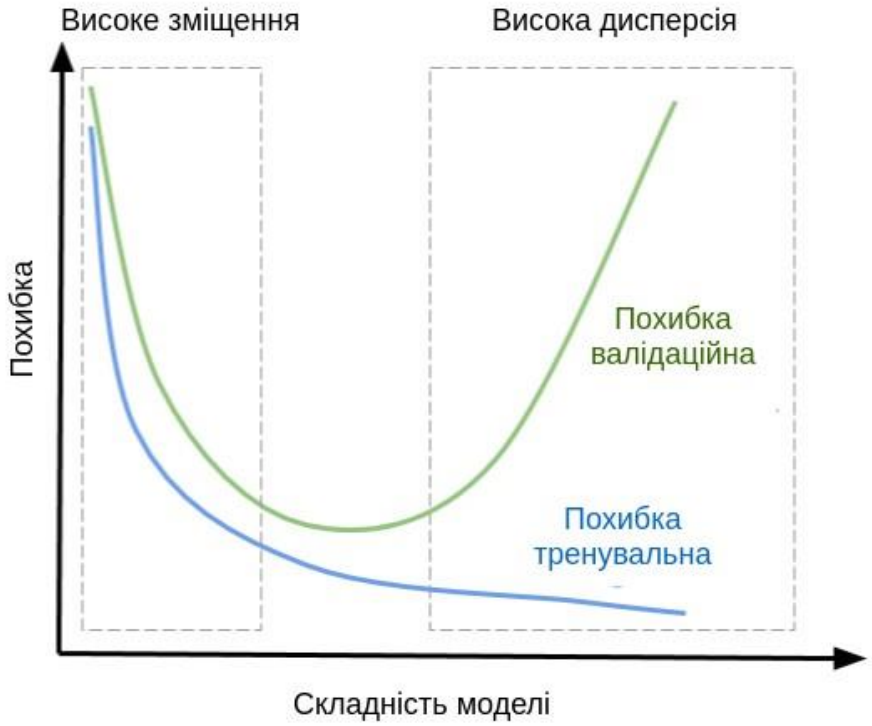

Рисунок 2.4 – Залежність зміщення та дисперсії від складності системи та похибки

При створенні класифікатора необхідно правильно розділити початковий набір даних на три частини. Тренувальна частина здійснює побудову моделі, крос-валідаційна частина використовується для оптимізації параметрів моделі, а тестова частина - для оцінки ефективності класифікатора. Кожна з цих частин відіграє важливу роль в процесі створення точного класифікатора.[12]

На практиці часто зустрічається метод кросс-валідації k-fold validation, метою якого є перевірка на те як побудована модель класифікує дані, які нові для неї і як саме вона буде працювати на практиці.

Модель використовує набір тренувальних даних для отримання якомога найкращого опису даних. Однак, існує небезпека перенавчання моделі, коли вона

відібралася на прикладах тренування та не може коректно робити прогнози для нових даних. Для запобігання перенавчанню, крос-валідація використовується для перевірки моделі на нових даних і визначення наявності перенавчання.

Крос-валідація методом k-fold розділює набір даних на k частин. На k-1 частинах проводиться навчання моделі, а її точність перевіряється на даних, що залишилися. Для того щоб отримати найбільш узагальнюючу оцінку ефективності алгоритму процедура повторюється k разів, змінюючи дані, на яких проводиться перевірка.

Принцип алгоритму k-fold зображения на рисунку 2.5:

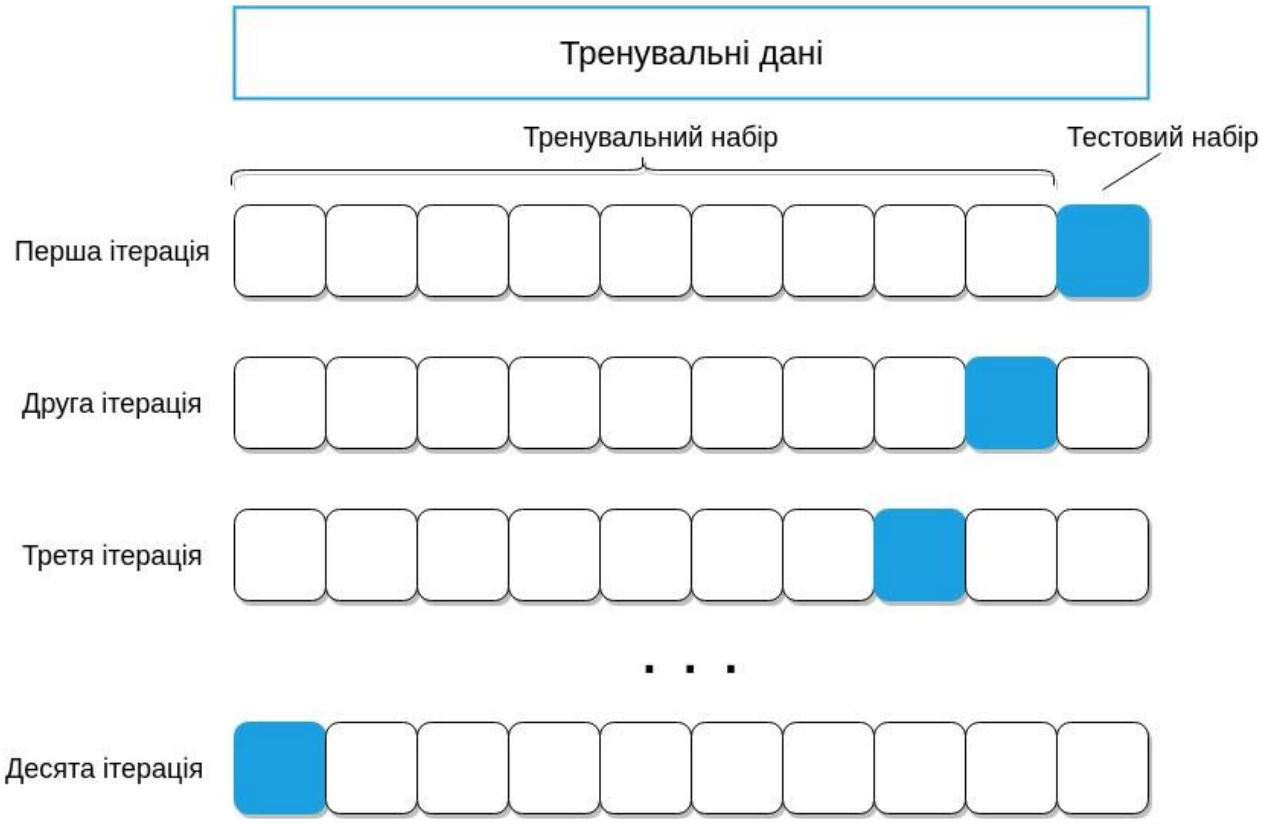

Рисунок 2.5 – Принцип алгоритму k-fold

У системі автоматичної класифікації текстової інформації використовується алгоритм крос-валідації k-fold з k = 5.

Висновки до розділу 2

Для реалізації поставленого завдання кваліфікаційної роботи потрібно застосовувати алгоритми класифікації. Перед тим як використовувати їх

потрібно обробити природню мову для подальшої обробки класифікатором. Досліджено, що існують такі моделі як WordNet, WER, Bag-of-words, TD-IDF, NER. Спільна суть даних моделей полягає в фільтруванні інформації для покращеної роботи алгоритмів класифікації за допомогою видалення зайвих символів, таких як знаки пунктуації, пробіли, табуляції, а також моделі, кожна по-своєму, видаляють непотрібні слова з тексту, які можуть заважати суті та зменшувати точність алгоритмів класифікації.

3 АРХІТЕКТУРА, МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технології та мови програмування

В першу чергу, для майбутньої розробки програмного забезпечення потрібно визначитись з технологіями які будуть застосовані в програмному забезпеченні та з мовою програмування.

Java — це об’єктно-орієнтована кросплатформна мова програмування, яка широко використовується для розробки великомасштабних програм. Мова добре підходить для програм видобутку тексту завдяки своїй надійності, стабільності та здатності обробляти великі обсяги даних. Віртуальна машина Java (JVM) гарантує, що код Java може працювати на будь-якій платформі, що робить її ідеальним вибором для розробки програм, які потрібно розгортати на кількох платформах.

Java є популярною та широко використовуваною мовою програмування, яка підходить для широкого спектру програм, включаючи видобуток тексту. Мова має низку особливостей, які роблять її хорошим вибором для розробки систем аналізу тексту, зокрема:

– Об’єктно-орієнтоване програмування: Java є об’єктно-орієнтованою мовою програмування, що означає, що вона базується на концепції об’єктів, класів і успадкування. Це спрощує моделювання реальних об’єктів і зв’язків, що важливо для аналізу тексту, де потрібно представляти такі поняття, як документи, терміни та теми.

– Портативність: Java призначена для переносимості на різні платформи та операційні системи, що означає, що систему видобутку тексту на основі Java можна запускати на різноманітних комп’ютерах, від настільних комп’ютерів до серверів. Це полегшує розгортання системи на різних машинах, а також її інтеграцію з іншими системами та інструментами.

– Велика екосистема: Java має велике та активне співтовариство розробників, які створили велику кількість бібліотек та інструментів для різних завдань, включаючи видобуток тексту. Це полегшує пошук уже існуючих рішень проблем і створення результатів уже наявної роботи.

– Продуктивність: Java — це швидка та ефективна мова, що робить її чудовою для завдань видобутку тексту, які вимагають великої потужності процесора. Наприклад, система видобутку тексту на основі Java може легко обробляти великі обсяги текстових даних і виконувати над ними складні операції, такі як токенізація, стемінг і векторизація.

– Масштабованість: Java створена для масштабування, що означає, що вона може обробляти великі обсяги даних і обробку без шкоди для продуктивності. Це важливо для аналізу тексту, коли розмір даних може швидко стати великим, а кількість необхідних обчислень може стати складною.

Іншим важливим фактором, який робить Java хорошим вибором для видобутку тексту, є наявність великої бібліотеки інструментів і фреймворків для машинного навчання,[16] обробки природної мови та аналізу даних. Екосистема Java надає розробникам широкий спектр бібліотек, включаючи Weka, RapidMiner і Apache Mahout, які можна використовувати для реалізації різноманітних алгоритмів машинного навчання для додатків видобутку тексту.

Weka, наприклад, є популярним інструментом інтелектуального аналізу даних, який надає повну колекцію алгоритмів машинного навчання та утиліт для попередньої обробки даних, класифікації, кластеризації та аналізу правил асоціації. Він надає простий у використанні інтерфейс для розробників, щоб експериментувати з різними алгоритмами та розробляти власні моделі для своїх додатків для видобутку тексту.[17]

RapidMiner — ще один інструмент, який широко використовується для додатків видобутку тексту. Це потужний, але простий у використанні інструмент інтелектуального аналізу даних, який надає широкий спектр алгоритмів машинного навчання, включаючи дерева рішень, k-найближчих сусідів і опорні

векторні машини.[18] RapidMiner також надає розширені функції, такі як вибір моделі, вибір функцій і оптимізація моделі, які необхідні для розробки точних і ефективних систем класифікації текстового аналізу.

Apache Mahout — ще одна популярна бібліотека машинного навчання з відкритим кодом, яка надає розробникам колекцію масштабованих алгоритмів для систем класифікації, кластеризації та рекомендацій. Він оптимізований для великих даних і може бути використаний для розробки великомасштабних програм інтелектуального аналізу тексту, які потребують ефективної обробки великих обсягів даних.

Порівнюючи мову програмування Java з мовою програмування Python можна виділити наступні переваги в даній розробці:

Продуктивність і масштабованість: однією з головних переваг Java над Python є її продуктивність і масштабованість. Java відома своєю швидкою та ефективною продуктивністю, що робить її хорошим вибором для розробки великомасштабних програм для видобутку тексту. Це пояснюється тим, що Java є мовою зі статичними типами, що означає, що змінні оголошуються певного типу, а перевірка типу виконується під час компіляції. Це дозволяє віртуальній машині Java оптимізувати код і зробити його швидшим. З іншого боку, Python є динамічно типізованою мовою, що означає, що тип змінної визначається під час виконання. Це може призвести до нижчої продуктивності порівняно з Java.

Надійні бібліотеки та інструменти: Java має багатий набір бібліотек та інструментів, які спеціально розроблені для аналізу тексту та обробки природної мови. Наприклад, Java Natural Language Toolkit (JNLTK) надає широкий спектр інструментів і бібліотек для видобутку тексту, включаючи токенізацію, тегування частин мови та аналіз настроїв. Крім того, Java також має велику спільноту розробників, які постійно роблять свій внесок у розвиток цих бібліотек, роблячи їх ще надійнішими та потужнішими. Для порівняння, Python має менше бібліотек і інструментів для аналізу тексту, хоча спільнота розробників, які працюють над цими бібліотеками, зростає.

Безпека: безпека є критично важливим аспектом будь-якої програми видобутку тексту, особливо коли ви маєте справу з конфіденційними даними. Java відома своїми надійними функціями безпеки, що робить її гарним вибором для розробки безпечних програм для видобутку тексту. Наприклад, Java має вбудовану модель безпеки, яка включає Java Security Manager, який може застосовувати політики безпеки, і Java Class Loader, який може запобігати виконанню шкідливого коду. Для порівняння, Python має обмежені функції безпеки та більш сприйнятливий до атак на безпеку.

Сумісність між платформами: Java є кросплатформною мовою, що означає, що вона може працювати на будь-якій платформі, включаючи Windows, Mac і Linux. Це робить його гарним вибором для розробки програм для видобутку тексту, які повинні працювати на кількох платформах. З іншого боку, Python не так сумісний з різними платформами, як Java, і може мати проблеми з роботою на деяких платформах.

Об’єктно-орієнтоване програмування: Java є об’єктно-орієнтованою мовою програмування, що означає, що вона підтримує інкапсуляцію, успадкування та поліморфізм. Ці функції спрощують розробку великомасштабних і складних програм видобутку тексту. Крім того, Java також має велику кількість шаблонів проєктування, які можна використовувати для розробки зручних і масштабованих програм. Python, з іншого боку, є мультипарадигмальною мовою, що означає, що він підтримує як об’єктно-орієнтоване, так і функціональне програмування.

Корпоративне впровадження: Java широко використовується в корпоративному світі та є переважною мовою для розробки корпоративних програм. Це означає, що існує велика група досвідчених Java-розробників, які можуть працювати над програмами видобутку тексту. Крім того, багато корпорацій інвестували в інструменти розробки Java, що полегшує розробку та підтримку великомасштабних програм видобутку тексту. Для порівняння, Python не так широко використовується в корпоративному світі, а це означає, що може

бути важче знайти досвідчених розробників Python для проєктів видобутку тексту.

У порівнянні ще з однією мовою програмування R, яка також може бути використана в завданнях text mining, окрім переваг які вказані вище над мовою програмування Python, Java має наступні переваги:

Портативність і масштабованість: Java є дуже портативною мовою, що означає, що код, написаний на Java, може працювати на будь-якій платформі без змін. Це робить Java чудовим вибором для проєктів видобутку тексту, які потрібно масштабувати до більших наборів даних або розгортати в різних середовищах. R, з іншого боку, обмежений середовищем, у якому він працює, що робить його менш портативним і більш складним для масштабування.

Продуктивність: Java — це мова зі статичними типами, що означає, що вона швидша та ефективніша за мови з динамічними типами, такі як R. Java також має компілятор Just-In-Time (JIT), який динамічно компілює код під час виконання, що призводить до покращена продуктивність порівняно з інтерпретованими мовами, такими як R. Крім того, Java має велику кількість бібліотек і інструментів для видобутку тексту, багато з яких оптимізовані для продуктивності.

Підсумовуючи, Java є хорошим вибором для розробки систем класифікації текстового аналізу завдяки її надійності, стабільності та наявності широкого спектру інструментів і бібліотек для машинного навчання та аналізу даних. Незалежно від того, розробляєте ви невелику програму чи великомасштабне рішення корпоративного рівня, Java надає функції та функції, необхідні для створення надійної та точної системи класифікації інтелектуального аналізу тексту.

3.2 Вибір компонентів програмного забезпечення

В якості компонентів програмного забезпечення були обрані Maven плагін для збірки та WEKA в якості бібліотеки для поліпшення роботи з класифікаторами text mining.

Maven — це потужний інструмент для створення та керування проєктами програмного забезпечення на основі Java. Коли справа доходить до розробки системи класифікації інтелектуального аналізу тексту, Maven може надати численні переваги, які роблять його ідеальним вибором для керування процесом розробки програмного забезпечення.

Перш за все, Maven пропонує надійну та масштабовану систему збірки. Maven може автоматично компілювати, тестувати та пакувати код, що полегшує розробникам зосередження на написанні та тестуванні коду. Maven також може керувати залежностями, полегшуючи використання сторонніх бібліотек у проєкті. Це дозволяє розробникам зосередитися на написанні власного коду, а не витрачати час на керування залежностями.

Maven також надає стандартизовану структуру проєкту, яка полегшує розробникам навігацію проєктом і розуміння того, як організовано код. Це сприяє повторному використанню коду та узгодженості, що може полегшити розробникам співпрацю над проєктом.

Крім того, Maven може автоматизувати процес розгортання, полегшуючи розповсюдження програмного забезпечення серед кінцевих користувачів. Maven можна налаштувати для автоматичного розгортання програмного забезпечення в різних середовищах, таких як розробка, тестування та виробництво. Це допомагає зменшити ризик людської помилки та забезпечує послідовне розгортання програмного забезпечення в усіх середовищах.[19]

Ще одна перевага використання Maven полягає в тому, що він підтримує використання плагінів, які можна використовувати для додавання нових функціональних можливостей у процес збірки. Наприклад, плагіни Maven можна

використовувати для автоматизації процесу виконання тестів, створення документації та розгортання програмного забезпечення.

Нарешті, Maven широко поширений і має велику спільноту користувачів і учасників. Це означає, що легко знайти ресурси та підтримку для Maven, включаючи форуми, навчальні посібники та плагіни.

Підсумовуючи, Maven є найкращим вибором в якості плагіну керування розробкою системи класифікації тексту, оскільки він пропонує надійну та масштабовану систему побудови, стандартизовану структуру проєкту, автоматичне розгортання, можливість використовувати плагіни та велику спільноту користувачів і учасників. Його здатність керувати залежностями, стандартизувати процес збирання та надавати архітектуру плагінів робить його ідеальним інструментом для створення програм на основі Java. Стандартизований процес збирання та багате сховище плагінів полегшують автоматизацію завдань збирання та забезпечують повторюваність процесу збирання, що полегшує тестування та підтримку програми з часом.[18]

В якості бібліотеки для роботи з частиною алгоритмів text mining в мові програмування Java була використана WEKA. WEKA — це популярне програмне забезпечення для аналізу даних із відкритим кодом, розроблене на Java, яке надає різні алгоритми машинного навчання для аналізу та прогнозування даних. Це потужний інструмент для розробки систем класифікації аналізу тексту завдяки його багатому набору функцій і зручному інтерфейсу. У цій статті ми обговоримо переваги використання WEKA для аналізу тексту та чому це чудовий вибір для розробки систем класифікації.

Перш за все, WEKA надає широкий спектр алгоритмів для інтелектуального аналізу тексту, включаючи дерева рішень, наивний Байєс, опорні векторні машини та багато іншого. Це дає можливість вибрати найбільш підходящий алгоритм для конкретного завдання класифікації та налаштувати його для підвищення продуктивності. Крім того, WEKA надає інструменти візуалізації, такі як Explorer і Experimenter, які дозволяють користувачам легко

порівнювати результати різних алгоритмів і знаходити найкращий для їх конкретної проблеми.

Ще однією перевагою використання WEKA є його інтеграція з Java. Java є широко використовуваною мовою програмування в промисловості та академічних колах, і WEKA надає Java API, який дозволяє легко інтегрувати системи класифікації текстового аналізу в існуючі програми Java. Це означає, що користувачі можуть легко створювати спеціальні програми, які використовують потужність WEKA, без необхідності вивчати нову мову програмування.

Крім того, WEKA надає низку інструментів попередньої обробки для очищення та підготовки даних для завдань інтелектуального аналізу тексту. Ці інструменти містять можливість фільтрувати атрибути, нормалізувати та стандартизувати дані та обробляти відсутні значення. Це спрощує для користувачів підготовку даних для завдань інтелектуального аналізу тексту та гарантує, що дані мають відповідний формат для аналізу.

Нарешті, WEKA має велику й активну спільноту користувачів, розробників і учасників. Це означає, що користувачі мають доступ до багатих знань і підтримки, а також до зростаючої бібліотеки плагінів і розширень, які можна використовувати для розширення функціональності WEKA.

Weka, RapidMiner і Apache Mahout є трьома популярними інструментами з відкритим кодом для видобутку тексту. Хоча всі три інструменти мають свої сильні та слабкі сторони, є кілька ключових причин, чому WEKA часто є кращим вибором для проєктів інтелектуального аналізу тексту.[20]

Однією з головних переваг WEKA є її простота. WEKA проста у використанні навіть для користувачів без досвіду програмування. Його графічний інтерфейс користувача дозволяє користувачам взаємодіяти з інструментом і виконувати такі завдання, як попередня обробка, вибір функцій і класифікація без необхідності писати код. Це робить WEKA ідеальним для людей, яким потрібно швидко навчитися працювати з видобутком тексту, або

тим, хто не має часу чи ресурсів, щоб вивчити більш складний інструмент, як-от RapidMiner або Apache Mahout.

Ще однією перевагою WEKA є його продуктивність. Незважаючи на свою простоту, WEKA є потужним інструментом, який може працювати з великими наборами даних і виконувати складні завдання інтелектуального аналізу тексту. Він часто швидший за інші інструменти видобутку тексту, такі як RapidMiner і Apache Mahout, а його алгоритми оптимізовані для текстових даних, що робить його ідеальним вибором для проєктів видобутку тексту.

WEKA також пропонує велику кількість алгоритмів для інтелектуального аналізу тексту, включаючи класичні алгоритми машинного навчання, такі як Naive Bayes, дерева рішень і опорні векторні машини, а також новіші алгоритми, такі як глибоке навчання та штучні нейронні мережі. Ця різноманітність алгоритмів робить WEKA ідеальним для проєктів, які вимагають більш спеціалізованого підходу до аналізу тексту.

Ще однією перевагою WEKA перед RapidMiner і Apache Mahout відкритість коду. Оскільки WEKA має відкритий код, користувачі мають доступ до вихідного коду, що означає, що вони можуть змінювати інструмент відповідно до своїх потреб. Це не стосується RapidMiner і Apache Mahout, які є комерційними інструментами.

Нарешті, WEKA широко використовується та добре задокументована. Це означає, що користувачі мають доступ до великої спільноти користувачів, які можуть запропонувати підтримку та керівництво, а також велику кількість онлайн-ресурсів і навчальних посібників. Це важливо, оскільки видобуток тексту може бути складним і складним завданням, а наявність доступу до спільноти підтримки може значно полегшити процес.

Підсумовуючи, WEKA — це потужний і гнучкий інструмент для розробки систем класифікації інтелектуального аналізу тексту. Його багатий набір функцій, зручний інтерфейс і інтеграція з Java роблять його ідеальним вибором

для спеціалістів з обробки даних, інженерів машинного навчання та дослідників, які хочуть легко виконувати завдання інтелектуального аналізу тексту.

В випадку використання сервісу як повноцінного окремого вебзастосунку буде застосовуватись Java Spring Boot Framework. Java Spring Boot — потужний інструмент для розробки веб-додатків на мові програмування Java. Він забезпечує надійну структуру для створення програм корпоративного рівня, які є масштабованими, модульними та простими в обслуговуванні. Коли мова йде про розробку програми класифікації тексту, Java Spring Boot може стати відмінним вибором завдяки таким особливостям:

– Швидкий розвиток Java Spring Boot розроблено для забезпечення швидкої розробки додатків, надаючи різноманітні функції та інструменти з коробки. Це дозволяє розробникам зосередитися на розробці основної функціональності своєї програми, не турбуючись про базову інфраструктуру. За допомогою Spring Boot розробники можуть швидко та легко створити веб-програму, яка може обробляти класифікацію тексту.

– Спрощена конфігурація Java Spring Boot спрощує процес конфігурації, дозволяючи розробникам витрачати менше часу на налаштування своєї програми та більше часу на написання коду. Він забезпечує низку автоматичних конфігурацій, які можна легко налаштувати відповідно до конкретних вимог програми. Це означає, що розробники можуть витрачати більше часу на написання бізнес-логіки та менше часу на налаштування.

– Архітектура мікросервісів Java Spring Boot розроблено для підтримки розробки мікросервісів, які є невеликими, незалежними та слабко пов’язаними сервісами. Ця архітектура дозволяє розробляти масштабовані та стійкі додатки, які можна легко розгортати та керувати ними. Для програми класифікації тексту ця архітектура може дозволити розробникам швидко розширювати програму в міру того, як зростає обсяг даних, які потрібно класифікувати.

– Безпека Безпека є важливим аспектом будь-якої програми, і Java Spring Boot надає надійні функції безпеки з коробки. Spring Security надає функції автентифікації, авторизації та контролю доступу, що полегшує захист програми класифікації тексту.

– Тестування Java Spring Boot забезпечує підтримку модульного та інтеграційного тестування, що дозволяє легко переконатися, що програма працює належним чином. Spring Boot також надає різноманітні інструменти тестування, які можуть допомогти розробникам писати ефективні тести та виявляти помилки на ранніх стадіях процесу розробки.[19]

– Розширюваність Java Spring Boot надає гнучку та розширювану платформу, яку можна легко налаштувати відповідно до конкретних потреб програми. Це дозволяє розробникам легко інтегрувати бібліотеки та інструменти сторонніх розробників, що полегшує додавання нових функцій і функцій до програми класифікації тексту.

– Підтримка спільноти Java Spring Boot має велику й активну спільноту, яка надає чудову підтримку, ресурси та документацію. Ця спільнота надає доступ до величезної бази знань, яка може допомогти розробникам швидко вирішувати проблеми та знаходити рішення поширених проблем.

Таким чином, Java Spring Boot є чудовим вибором для розробки програми класифікації тексту. Він забезпечує швидку розробку, спрощену конфігурацію, архітектуру мікросервісів, безпеку, тестування, розширюваність і підтримку спільноти. Завдяки цим функціям розробники можуть легко створити надійну та масштабовану програму класифікації тексту, яка відповідає конкретним вимогам їхнього бізнесу.

В якості API який буде застосовуватись в випадку використання сервісу з back-end частиною інформаційної системи буде використано REST API. У сфері класифікації тексту REST API є поширеним інструментом, який використовується для доступу та використання попередньо створеного

класифікатора. Використання REST API має багато переваг, нижче розглянемо деякі з них.

По-перше, використання REST API для класифікації тексту забезпечує більшу гнучкість і масштабованість системи. REST API розроблено таким чином, щоб бути незалежним від платформи, і до нього можна отримати доступ з будь-якої мови чи системи, яка може робити запити HTTP. Це означає, що систему класифікації тексту на основі REST API можна легко інтегрувати в широкий спектр інших систем і платформ, завдяки чому вона легко адаптується до мінливих потреб і вимог.

Ще однією ключовою перевагою використання REST API є можливість відокремити логіку класифікації тексту від решти системи. Інкапсулюючи функцію класифікації тексту в API, можна змінювати або оновлювати базовий класифікатор без необхідності модифікувати всю систему. Це особливо корисно в ситуаціях, коли компонент класифікації тексту може потребувати частого оновлення, щоб не відставати від даних і вимог, що змінюються.[21]

Крім того, використання REST API дозволяє легше тестувати та перевіряти компонент класифікації тексту. Оскільки до API можна отримати доступ незалежно від решти системи, можна перевірити логіку класифікації тексту окремо, не турбуючись про вплив інших компонентів системи. Це може допомогти переконатися, що компонент класифікації тексту працює належним чином і його можна легше оптимізувати для продуктивності.

На додаток до цих переваг, використання REST API також забезпечує кращий захист і контроль доступу для компонента класифікації тексту. Оскільки API можна захистити за допомогою механізмів автентифікації та авторизації, можна контролювати, хто має доступ до функцій класифікації тексту та як вона використовується. Це може допомогти захистити конфіденційні дані та забезпечити безпечне та відповідальне використання системи.

Нарешті, використання REST API дозволяє краще контролювати та реєструвати компонент класифікації тексту. Оскільки запити API можна легко відстежувати та реєструвати, можна контролювати використання та продуктивність компонента класифікації тексту в режимі реального часу. Це може допомогти виявити проблеми та потенційні вузькі місця, а також надати цінну інформацію про те, як використовується система та як її можна оптимізувати для кращої продуктивності.

Одним з головних складових REST API який буде застосований під час розробки класифікатору є метод GET. Метод GET є одним із найпоширеніших методів HTTP у веб-службах RESTful. Він використовується для отримання ресурсу або колекції ресурсів із сервера. Коли клієнт надсилає запит GET на сервер, сервер відповідає представленням запитуваного ресурсу, як правило, у формі JSON або XML.

Метод GET вважається безпечним і ідемпотентним, що означає, що кілька ідентичних запитів матимуть такий же ефект, як і один запит. Іншими словами, виконання кількох запитів GET до одного ресурсу не змінить ресурс або його стан.

В REST API метод GET зазвичай використовується для отримання інформації про конкретний ресурс або колекцію ресурсів. Наприклад, запит GET до кінцевої точки /users може повернути список користувачів, тоді як запит GET до кінцевої точки /users/{id} може повернути інформацію про конкретного користувача, ідентифікованого за його ID.[22]

На додаток до отримання ресурсів, метод GET також можна використовувати для виконання інших операцій, які не змінюють стан сервера, таких як пошук ресурсів або фільтрація колекцій ресурсів на основі певних критеріїв.

Загалом, метод GET є основним будівельним блоком веб-сервісів RESTful і необхідний для отримання даних і ресурсів із сервера безпечним і ідемпотентним способом.

Таким чином, використання REST API для класифікації тексту забезпечує багато переваг, включаючи гнучкість, масштабованість, адаптивність, а також легше тестування та перевірку. Інкапсулюючи функцію класифікації тексту в API, можна змінювати або оновлювати базовий класифікатор без необхідності модифікувати всю систему. Крім того, використання REST API забезпечує кращий захист і контроль доступу, а також спрощує моніторинг і журналювання компонента класифікації тексту. Усі ці переваги поєднуються, щоб зробити систему класифікації тексту на основі REST API високоефективним і ефективним рішенням для широкого спектру програм.

3.3 Розробка архітектури програмного забезпечення

Розробка архітектури для програми класифікації тексту вимагає системного підходу, який враховує різні аспекти, такі як масштабованість, продуктивність, зручність обслуговування та гнучкість. Нижче наведено архітектуру високого рівня для програми класифікації тексту.

Збір даних. Першим кроком у створенні програми класифікації тексту є збір великої кількості відповідних текстових даних. Ці дані мають включати різні типи тексту, як-от новинні статті, публікації в соціальних мережах і огляди продуктів. Дані також мають бути позначені відповідними категоріями або тегами, оскільки це буде використано для навчання класифікатора.

Попередня обробка даних: після того, як дані зібрано, їх потрібно попередньо обробити, щоб переконатися, що вони мають формат, придатний для використання алгоритмом класифікації тексту. Цей етап попередньої обробки може включати очищення даних від будь-яких нерелевантних символів або слів, перетворення тексту на малі літери та токенізацію тексту в окремі слова.

Витяг ознак: наступним кроком є вилучення відповідних ознак із текстових даних, які використовуватимуться для навчання класифікатора. Це може включати обчислення частоти слів, виділення n-грамів і використання

частотно-інверсного частотного зважування термінів (TF-IDF) для визначення найважливіших слів у кожному текстовому документі.

Вибір моделі: після виділення функцій наступним кроком буде вибір відповідної моделі для завдання класифікації тексту. Це може бути простий байєсівський класифікатор або більш складна модель глибокого навчання. Вибір моделі залежатиме від конкретних вимог програми класифікації тексту та розміру та складності даних.

Навчання моделі: після вибору моделі її можна навчити, використовуючи попередньо оброблені текстові дані та витягнуті функції. Модель використовуватиме навчальні дані для вивчення зв’язків між функціями та категоріями тексту, щоб вона могла точно передбачити категорію нових, невидимих текстових даних.

Перевірка моделі: після навчання моделі важливо перевірити її продуктивність на окремому наборі даних перевірки. Це дозволить нам виміряти точність класифікатора та внести необхідні коригування для покращення його продуктивності.

Розгортання: після перевірки моделі її можна розгортати у виробничому середовищі. Це може передбачати інтеграцію системи класифікації тексту в існуючу програму або створення автономної системи, до якої користувачі зможуть отримати доступ.

Обслуговування: нарешті, важливо регулярно контролювати продуктивність системи класифікації тексту та вносити будь-які необхідні оновлення чи вдосконалення, щоб підтримувати її точність і ефективність.

Підсумовуючи, розробка архітектури програми класифікації тексту вимагає ретельного розгляду різних факторів, щоб гарантувати, що система є масштабованою, продуктивною, зручною для обслуговування та гнучкою. Дотримуючись системного підходу та використовуючи правильні інструменти та методи, можна побудувати надійну та ефективну систему класифікації тексту,

яка може допомогти організаціям краще зрозуміти та класифікувати великі обсяги текстових даних.

Щоб розробити систему класифікації тексту за допомогою Java, потрібно почати зі створення класу для кожного класифікатора тексту. У цій архітектурі кожен клас класифікатора буде відповідати за виконання конкретного завдання класифікації, такого як аналіз почуття, класифікація за темами або ідентифікація мови. Діаграма класів, які мають бути створені під час розробки застосунку вказана на рисунку 3.1.

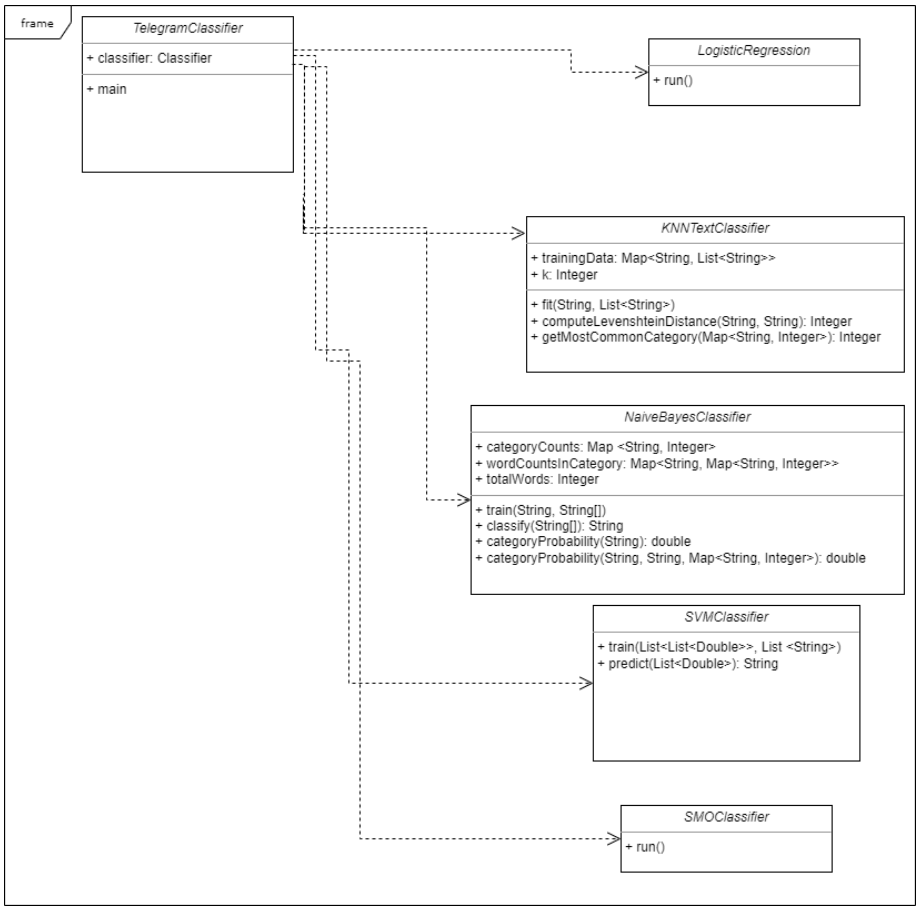

Рисунок 3.1 – Діаграма класів

Щоб реалізувати кожен клас класифікатора, потрібно попередньо обробити текстові дані, видаливши будь-яку нерелевантну інформацію, таку як стоп-слова або спеціальні символи. Цей крок важливий, оскільки він може допомогти підвищити точність і ефективність класифікатора.

Далі потрібно перетворити попередньо оброблений текст у числове представлення, наприклад, за допомогою пакета слів або матриці частотно-інверсної частоти термінів (TF-IDF). Це числове представлення дозволяє передати дані в алгоритм класифікатора.

Для кожного класу класифікатора буде реалізовано алгоритм класифікації. Є багато алгоритмів на вибір, включаючи дерева рішень, опорні векторні машини (SVM) і нейронні мережі.

Після того, як алгоритм класифікації буде реалізовано, ви будете використовувати його для прогнозування нових, невидимих текстових даних. Щоб оцінити продуктивність класифікатора, ви повинні порівняти передбачення з фактичними мітками для текстових даних.

Щоб запустити класифікатор на текстовому масиві рядків, потрібно буде створити екземпляр класу класифікатора для кожного рядка та викликати його метод передбачення. Метод передбачення повертав би мітку або клас для кожного рядка, вказуючи на його класифікацію.

Всередині класу кожного класифікатору можуть бути створені додаткові методи які необхідні для правильної роботи з даними під час класифікації. У кожного класифікатору можуть бути свої методи через особливості роботи алгоритму які потрібно реалізовувати в Java. Клас з класифікатором Naive Bayes потрібно реалізувати наступним чином:

Створюючи клас класифікатора тексту Naive Bayes у Java, ви реалізуєте алгоритм у спосіб, який робить його придатним для повторного використання та гнучкістю. Цей клас можна використовувати в різних проєктах класифікації тексту, і ви можете легко змінити або розширити його відповідно до ваших конкретних потреб.

Метод навчання в класі використовується для навчання класифікатора на заданому наборі текстових документів. Цей метод приймає як вхідні дані список текстових документів і відповідних їм категорій, і використовує цю інформацію для побудови ймовірнісних моделей для кожної категорії. Це включає

обчислення ймовірності появи кожного слова в кожній категорії та загальної ймовірності кожної категорії.

Метод classify приймає як вхідні дані новий текстовий документ і повертає категорію, до якої він, швидше за все, належить. Цей метод обчислює ймовірність кожної категорії для нового документа на основі моделей, побудованих на етапі навчання, і повертає категорію з найвищою ймовірністю.

Метод categoryProbability обчислює ймовірність заданої категорії для заданого текстового документа. Цей метод використовує ймовірнісні моделі, створені на етапі навчання, щоб обчислити ймовірність кожного слова в документі з урахуванням категорії, а потім множить ці ймовірності, щоб отримати загальну ймовірність категорії.

Метод wordProbability обчислює ймовірність того, що певне слово входить до певної категорії. Цей метод використовує ймовірнісні моделі, побудовані на етапі навчання, щоб оцінити вірогідність слова з урахуванням категорії.

У випадку з SVMClassifier методи мають бути побудовані, через те що вони засновані на ідеї пошуку гіперплощини, яка максимально розділяє різні класи даних таким чином:

Під час створення класу текстового класифікатора SVM у проєкті Java необхідно включити методи для навчання класифікатора, створення прогнозів і вилучення ознак із текстових даних. Метод «train» має використовуватись для побудови моделі з використанням набору позначених навчальних даних. Метод «predict» повинен використовувати навчену модель для прогнозування нових, невидимих текстових даних. Метод "extractFeatures" повинен використовуватись для перетворення текстових даних у числове представлення, яке може використовуватися моделлю SVM.

Щоб використовувати ці методи, клас класифікатора SVM спочатку має бути створений, а потім навчений на наборі позначених текстових даних. Коли модель навчена, її можна використовувати для прогнозування нових текстових даних, викликавши метод predict і передавши текстові дані як аргумент.

Таким чином, архітектура системи класифікації тексту складатиметься з кількох класів класифікаторів, кожен з яких відповідає за виконання конкретного завдання класифікації. Використовуючи модульний дизайн, ви можете легко додавати або видаляти класифікатори за потреби та повторно використовувати окремі класифікатори в кількох програмах.

Висновки до розділу 3

В якості мови програмування програмного забезпечення була обрана Java через те що вона є хорошим вибором для видобутку тексту, має велику бібліотеку інструментів і фреймворків для машинного навчання, обробки природної мови та аналізу даних. Також сувора типізація даних в мові програмування Java дозволяє спростити фільтрацію коментарів та вилучення з них нетекстових даних.

Серед бібліотек які допомагатимуть працювати з певними алгоритмами класифікації тексту в Java було обрано WEKA, через кращу продуктивність в порівнянні з аналогами, та через open-source код, що дозволяє користуватись бібліотекою без порушень авторського права. В масштабах компанії це дуже важливо, так як не витрачаються великі кошти на можливе ліцензування програмного забезпечення.

4 КОДУВАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, КЕРІВНИЦТВО КОРИСТУВАЧА

4.1 Інструменти для реалізації програмного забезпечення

Для розробки програмного забезпечення було використано InteliJ IDEA Ultimate. IntelIJ IDEA — це потужне інтегроване середовище розробки (IDE) для розробки програм Java. Його вважають одним із найкращих варіантів для розробки систем класифікації тексту завдяки його розширеним функціям і простоті використання. Нижче наведено деякі з причин, чому IntelIJ IDEA є ідеальним вибором для розробки системи класифікації тексту:

– Зручний інтерфейс: IntelIJ IDEA має зручний інтерфейс, який полегшує навігацію та використання. IDE надає візуальний інтерфейс для кодування та налагодження, що спрощує процес розробки та тестування систем класифікації тексту.

– Розширений рефакторинг коду: IntelIJ IDEA надає розширені можливості рефакторингу коду, які спрощують підтримку та покращення якості коду. IDE має набір інструментів рефакторингу коду, які допомагають у реорганізації та вдосконаленні структур коду, що допомагає зменшити ризик помилок у коді.

– Вбудований налагоджувач: IntelIJ IDEA має вбудований налагоджувач, який забезпечує простий і ефективний спосіб тестування та налагодження систем класифікації тексту. Налагоджувач дозволяє користувачам легко проходити код і перевіряти змінні, що дає змогу швидко виявляти та виправляти помилки.

– Вбудоване модульне тестування: IntelliJ IDEA має вбудовану структуру модульного тестування, яка полегшує написання та виконання тестів. Ця функція може допомогти забезпечити ретельне тестування коду перед випуском.

– Плагіни IntelliJ: IntelIJ IDEA надає широкий вибір плагінів, які можна використовувати для розширення функціональності IDE. Ці плагіни можна використовувати для додавання нових функцій, таких як підтримка певних мов програмування або інструментів, що робить можливим створення систем класифікації тексту з більш широким набором інструментів і технологій.

– Завершення коду: IntelIJ IDEA надає функції інтелектуального завершення коду, які можуть заощадити час і підвищити продуктивність кодування. Функції завершення коду дозволяють автоматично завершувати фрагменти коду, зменшуючи потребу вручну вводити код.

– Перевірка коду: IntelIJ IDEA надає можливості перевірки коду, які допомагають виявити проблеми з кодом і запропонувати вдосконалення. Функції перевірки коду допомагають виявляти потенційні проблеми з кодом і рекомендувати рішення, які можуть допомогти покращити якість коду.

– Вбудований профайлер: IntelIJ IDEA надає вбудований профайлер, який можна використовувати для оптимізації та покращення продуктивності систем класифікації тексту. Профайлер дає змогу вимірювати продуктивність коду та визначати області для вдосконалення, що може допомогти покращити загальну продуктивність системи.

Підсумовуючи, IntelIJ IDEA є ідеальним вибором для розробки систем класифікації тексту завдяки зручному інтерфейсу, розширеним можливостям рефакторингу коду, вбудованому налагоджувачу, великому вибору плагінів, функціям інтелектуального завершення коду, можливостям перевірки коду та вбудованим профайлер. Ці функції спрощують розробку, тестування та вдосконалення систем класифікації тексту, що може допомогти в досягненні високоякісних результатів і ефективних систем.

4.2 Реалізація та тестування програмного забезпечення

Результатом розробки програмного забезпечення для класифікації коментарів стало створення універсального сервісу, який може працювати як

самостійно, так і в якості складової інформаційної системи месенджеру Telegram.

Сервіс реалізований за допомогою класів, які потрібно було розробити для проведення тестування різних алгоритмів класифікації тексту, та за допомогою класу Classifier, завдяки якому відбувається тестування та робота з кращим алгоритмом класифікатору. З цими класами йде взаємодія як і в якості незалежного сервісу, так і в якості back-end частини інформаційної системи Telegram(приклад розгортання як одного з сервісів Telegram наведено на рисунку 4.1.).

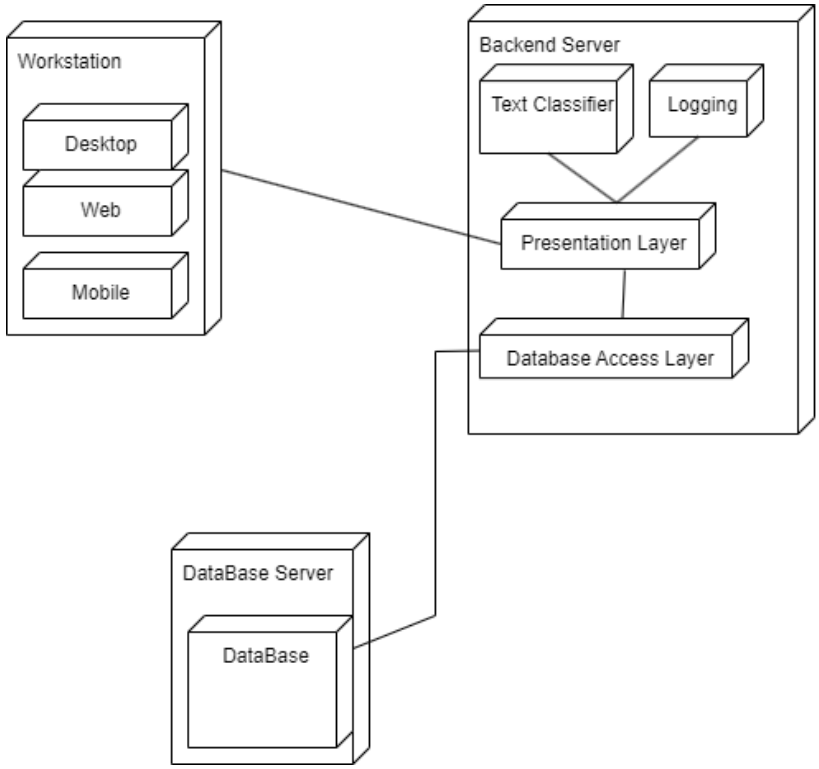

Рисунок 4.1 – Діаграма розгортання

Для тестування програмного забезпечення можна його використовувати поза межами Telegram. В такому випадку навчання та тестування буде відбуватись в один і той самий час під час запуску програми. Якщо в такому режимі використовувати застосунок багаторазово, то під час навчання класифікатору будуть повторно використовуватись на однотипні завдання ресурси персонального комп’ютера.

Для визначення найбільш точного алгоритму класифікації тексту потрібно запустити кожен класифікатор на навчальних даних. Перед запуском навчальні дані мають бути оброблені, відфільтровані та векторизовані на окремі слова.

StringToWordVector — це техніка попередньої обробки, яка використовується для перетворення необроблених текстових даних у набір числових ознак, які можна використовувати для класифікації тексту. Це робиться шляхом виділення окремих слів із тексту та зіставлення їх із набором функцій. Потім ці функції можна використовувати для представлення тексту в числовому форматі, який можна обробити алгоритмами машинного навчання.[19]

Існує кілька причин, чому StringToWordVector є ефективною технікою для підготовки тексту для класифікації тексту:

Узгоджене представлення функцій. Однією з ключових переваг StringToWordVector є те, що він забезпечує узгоджене представлення функцій у всіх документах у наборі даних. Це означає, що кожне входження слова обробляється однаково, незалежно від його розташування в документі. Це полегшує ідентифікацію шаблонів і схожості між документами, що може призвести до кращої ефективності класифікації.

Усунення шуму: у необроблених текстових даних часто є слова, які не містять жодної значущої інформації для класифікації. Приклади включають звичайні стоп-слова, такі як "the", "and" і "a", а також знаки пунктуації та спеціальні символи. StringToWordVector надає спосіб усунути цей шум шляхом фільтрації цих слів і символів, залишаючи лише найважливіші характеристики для класифікації.

Масштабованість: StringToWordVector — це масштабована техніка, яка може обробляти великі набори даних із мільйонами документів і сотнями тисяч функцій. Це важливо для завдань класифікації тексту, які часто включають великі обсяги даних.

Гнучкість: StringToWordVector забезпечує високий ступінь гнучкості щодо вибору функцій і зважування. Наприклад, це дозволяє користувачеві вибирати

мінімальний і максимальний пороги частоти для функцій, а також вибирати різні схеми зважування, такі як термін частотно-інверсної частоти документа (TF-IDF). Це дозволяє користувачеві точно налаштувати представлення ознак для оптимізації ефективності класифікації.

Інтеграція з WEKA: StringToWordVector — це вбудована техніка попередньої обробки в WEKA, популярній бібліотеці машинного навчання на Java. Це означає, що він простий у використанні та інтеграції з іншими класифікаторами WEKA, а також може бути включений у більший конвеєр машинного навчання.

Таким чином, StringToWordVector є ефективною технікою для підготовки тексту для класифікації тексту завдяки своїй здатності забезпечувати узгоджене представлення ознак, усувати шум, обробляти великі набори даних, надавати гнучкість у виборі ознак і зважування, а також її інтеграцію з WEKA. Це важливий етап попередньої обробки для будь-якого завдання класифікації тексту та широко використовується в академічних і промислових умовах.

Для навчальних даних використаний датасет, який зібраний з коментарів в Telegram з різних публічних джерел. Вони мають такі види суспільної думки: empty,sadness,enthusiasm,neutral,worry,surprise,love,fun,hate,happiness,boredom,relief,anger. Розглянемо інформацію про кожну з них:

– empty: емоція порожнечі відноситься до відчуття внутрішньої порожнечі, безглуздості або безцільності. Це часто асоціюється з почуттям самотності, апатії або відчуженості від себе та світу. Люди можуть відчувати порожнечу у відповідь на різні обставини, такі як втрата, відмова або невдача. Порожнечу часто описують як негативну емоцію, і вона може бути ознакою психологічного стресу або депресії.

– sadness: смуток — одна з основних людських емоцій, і вона часто асоціюється з втратою, розчаруванням або горем. Це природна реакція на негативні події чи обставини, і вона часто характеризується почуттям поганого настрою, млявістю та незацікавленістю. Смуток є універсальною емоцією, і її

відчувають люди різного віку, статі та культури. Це може бути нормальною частиною людського досвіду, але постійний або сильний смуток може бути ознакою депресії або інших психічних захворювань.

– enthusiasm: ентузіазм відноситься до сильного почуття хвилювання, інтересу або бажання з приводу чогось. Це часто асоціюється з позитивними подіями, такими як досягнення, успіх або значущий досвід. Ентузіазм — це мотивуюча емоція, яка може спонукати людей досягати своїх цілей і прагнень. Це часто виражається через велику енергію, оптимізм і взаємодію з іншими.

– neutral: нейтральність зазвичай не вважаються емоціями в психологічних дослідженнях, але їх можна використовувати для вказівки на відсутність емоційної валентності в тексті чи ситуації. Нейтральний текст може не викликати сильної позитивної чи негативної реакції у читача чи слухача. Його можна використовувати для опису фактичної або описової інформації, яка не є за своєю суттю емоційною.

– worry: занепокоєння - це поширена емоція, яка часто пов'язана з тривогою або страхом. Зазвичай це включає занепокоєння або побоювання щодо можливих майбутніх подій або результатів. Люди можуть турбуватися про низку тем, наприклад, про здоров’я, фінанси, стосунки чи роботу. Занепокоєння може бути нормальною реакцією на стрес або невизначеність, але надмірне або постійне занепокоєння може бути ознакою занепокоєння або інших психічних захворювань.

– surprise: подив - це емоція, яка викликається несподіваними або дивовижними подіями чи стимулами. Це часто характеризується коротким моментом шоку або розгубленості, після чого слід оцінка або оцінка ситуації. Здивування може бути позитивним або негативним, залежно від контексту та індивідуальної інтерпретації події. Це часто виражається через вираз обличчя або словесні сигнали, такі як «вау» або «о боже мій».

– love: кохання – це складна та багатогранна емоція, яка пов’язана з сильними почуттями прихильності, турботи та прихильності до іншої людини чи речі. Це часто виражається через дії, такі як фізичний дотик, слова підтвердження або послуги. Кохання може приймати різні форми, наприклад, романтичне кохання, батьківська любов або любов до друзів чи домашніх тварин. Це позитивна емоція, яка часто асоціюється зі щастям і благополуччям.

– fun: Емоція веселощів зазвичай асоціюється з легким і приємним досвідом. Тексти, які виражають емоцію веселощів, імовірно, містять слова або фрази, пов’язані з розвагою, сміхом і радістю.

– hate: Емоція ненависті часто асоціюється з сильним почуттям відрази до чогось або когось. Тексти, які виражають емоцію ненависті, ймовірно, містять негативні висловлювання, і почуття можуть бути спрямовані на людину, групу, ідею чи об’єкт.

– happiness: емоція щастя пов’язана з відчуттям задоволення та задоволення. Тексти, які виражають емоцію щастя, ймовірно, містять слова чи фрази, пов’язані з позитивом, радістю та задоволенням.

– boredom: емоція нудьги пов'язана з відчуттям нудності або відсутності інтересу. Тексти, які виражають емоцію нудьги, можуть містити фрази, пов’язані з монотонністю або повторенням, а настрої можуть вказувати на відсутність залучення до змісту.

– relief: відчуття полегшення часто асоціюється з відчуттям звільнення або задоволення від того, що щось не приємне або напружене закінчилося. Тексти, які виражають емоцію полегшення, можуть містити слова або фрази, пов’язані з розслабленням, легкістю та комфортом.

– anger: емоція гніву зазвичай пов'язана з почуттям ворожості або незадоволення. Тексти, які виражають емоцію гніву, ймовірно, містять негативну мову, і почуття можуть бути спрямовані на людину, групу, ідею чи об’єкт.

Набір даних містить дані з кількістю за типами, яка вказана в таблиці 4.1

Таблиця 4.1. – Кількість елементів в наборі даних за типами

Вид настрою	*Кількість елементів*
Empty	827
Sadness	5161
Enthusiasm	759
Neutral	8638
Worry	8456
Surprise	2187
Love	3842
Fun	1776
Hate	1323
Happiness	5209
Boredom	179
Relief	1526
Anger	110

Загальна кількість елементів становить 39993, серед яких найбільше елементів з настроєм “worry” та нейтральним. Для визначення найкращого класифікатору для коментарів створено окремі класи для кожного з видів класифікаторів. Класифікатори розміщені в окремому пакеті classifiers для покращеного знаходження класів.

Незважаючи на різноманіття алгоритмів для класифікації тексту, всі класи які створені на мові програмування Java мають спільні функції, такі як:

– функція попередньої обробки: текст очищається та попередньо обробляється для видалення нерелевантної інформації, такої як стоп-слова, спеціальні символи, знаки пунктуації та цифри.

– функція вилучення ознак: попередньо оброблений текст потім перетворюється на числове представлення ознак, присутніх у тексті.

– функція навчання моделі: модель машинного навчання навчається за допомогою попередньо оброблених і вилучених функцій тексту.

– функція оцінки: модель оцінюється на окремому наборі даних, щоб оцінити її продуктивність.

– функція прогнозування: навчена модель потім використовується для прогнозування класу нових або невидимих текстових даних.

Переглянемо ключові методи класів класифікаторів, завдяки яким вони можуть передбачати настрій тексту. На рисунку 4.2 вказаний метод predict класифікатору KNNTextClassifier. З даною назвою створений класифікатор за допомогою методу найближчих сусідів.

```
public String predict(String text) {
    Map<String, Integer> scores = new HashMap<>();
    for (Map.Entry<String, List<String>> entry : trainingData.entrySet()) {
        String category = entry.getKey();
        List<String> texts = entry.getValue();
        for (String t : texts) {
            int distance = computeLevenshteinDistance(text, t);
            scores.put(category, scores.getOrDefault(category, defaultValue: 0) + distance);
        }
    }
    return getMostCommonCategory(scores);
}
```

Рисунок 4.2 – Метод predict класифікатору k-найближчих сусідів

В даному випадку застосовуються такі змінні:

– “text”: ця змінна містить вхідний текст, який потрібно класифікувати.

– “trainingData”: це пара «ключ» - «значення», яка містить навчальні дані, де кожен ключ є категорією, а значенням є список текстів, що належать до цієї категорії.

– “scores”: це пара «ключ» - «значення», яка відстежує кумулятивну відстань вхідного тексту від кожного тексту в навчальних даних для кожної категорії.

– “category”: ця змінна представляє поточну категорію, яка розглядається у зовнішньому циклі.

– “texts”: ця змінна є списком текстів, що належать до поточної категорії, яка розглядається.

– “t”: ця змінна представляє поточний текст, який порівнюється з вхідним текстом.

– “distance”: ця змінна представляє відстань Левенштейна між вхідним текстом і поточним навчальним текстом, який порівнюється.

– “getMostCommonCategory”: це допоміжна функція, яка повертає категорію з найменшою сукупною відстанню для вхідного тексту.

Завдяки останній функції отримується остаточне значення класу настрою тексту. В класифікаторі NaiveBayesClassifier є також метод classify, який вже приймає масив слів, які були попередньо розбиті з рядка, та має алгоритм дій який зображений на рисунку 4.3.

```
public String classify(String[] words) {
    String bestCategory = null;
    double maxProbability = -1;

    for (String category : categoryCounts.keySet()) {
        double categoryProbability = categoryProbability(category);
        double probability = categoryProbability;

        Map<String, Integer> wordCounts = wordCountsInCategory.get(category);
        for (String word : words) {
            probability *= wordProbability(word, category, wordCounts);
        }

        if (probability > maxProbability) {
            maxProbability = probability;
            bestCategory = category;
        }
    }

    return bestCategory;
}
```

Рисунок 4.3 - Метод predict класифікатору NaiveBayes

В даному методі застосовуються такі змінні:

– “words”: це вхідний текст, який ми хочемо класифікувати. Це набір окремих слів, які складають текст.

– “bestCategory”: Ця змінна відстежує категорію з найвищою ймовірністю правильної класифікації для вхідного тексту. Він починається як нуль і оновлюється, коли ми переглядаємо категорії.

– “maxProbability”: ця змінна відстежує найвищу ймовірність, яку ми бачили, коли ми переглядаємо категорії. Починається як -1 і оновлюється, коли ми переглядаємо категорії.

– “categoryCounts”: це карта, яка зберігає кількість документів у кожній категорії. Він зіставляє назви категорій із кількістю документів у цій категорії.

– “wordCountsInCategory”: це вкладена карта, яка зберігає кількість кожного слова в кожній категорії. Він зіставляє назви категорій з іншою картою, яка зіставляє рядки слів із кількістю разів, коли це слово з’являється в документах цієї категорії.

– “categoryProbability”: цей метод обчислює попередню ймовірність певної категорії на основі кількості документів у цій категорії. Він приймає назву категорії як вхідні дані та повертає ймовірність як подвійну величину.

– “wordProbability”: цей метод розраховує ймовірність того, що дане слово входить до певної категорії. Він приймає слово, назву категорії та карту кількості слів у цій категорії як вхідні дані та повертає ймовірність як подвійну величину.

Загалом, цей метод використовує кількість категорій і кількість слів у кожній категорії, щоб обчислити ймовірність належності введеного тексту до кожної категорії. Він повторює кожну категорію та обчислює ймовірність шляхом множення ймовірності категорії на добуток ймовірностей кожного слова

у вхідному тексті. Категорія з найвищою ймовірністю повертається як прогнозована класифікація для вхідного тексту.

Методи для навчання класифікатору також відрізняються за алгоритмом. Поглянемо на один з таких на рисунку 4.4.

```
public void train(String category, String[] words) {
    if (!categoryCounts.containsKey(category)) {
        categoryCounts.put(category, 1);
    } else {
        categoryCounts.put(category, categoryCounts.get(category) + 1);
    }

    if (!wordCountsInCategory.containsKey(category)) {
        wordCountsInCategory.put(category, new HashMap<>());
    }

    Map<String, Integer> wordCounts = wordCountsInCategory.get(category);
    for (String word : words) {
        totalWords++;
        if (!wordCounts.containsKey(word)) {
            wordCounts.put(word, 1);
        } else {
            wordCounts.put(word, wordCounts.get(word) + 1);
        }
    }
}
```

Рисунок 4.4 - Метод train класифікатору NaiveBayes

Пояснення елементів методу:

– public void train(String category, String[] words): це метод, який навчає класифікатор тексту на певній категорії тексту, представленій у вигляді масиву слів. Параметр категорії представляє категорію тексту, а параметр words — це масив рядків, що містять окремі слова в тексті.

– if (!categoryCounts.containsKey(category)) {...}: це умовний оператор, який перевіряє, чи карта categoryCounts уже містить указану категорію. Якщо ні, вона додає категорію на карту та ініціалізує її кількість на 1.

– categoryCounts.put(category, categoryCounts.get(category) + 1): якщо карта categoryCounts уже містить указану категорію, цей рядок збільшує кількість категорії на 1.

– if (!wordCountsInCategory.containsKey(category)) {...}: це умовний оператор, який перевіряє, чи карта wordCountsInCategory вже містить указану категорію. Якщо ні, вона додає категорію до карти та ініціалізує її значення новою порожньою картою HashMap.

– Map<String, Integer> wordCounts = wordCountsInCategory .get(category): цей рядок отримує карту кількості слів для вказаної категорії з карти wordCountsInCategory і призначає її новій змінній wordCounts.

– for (String word : words) {...}: це цикл, який повторює кожне слово в масиві слів.

– totalWords++: цей рядок збільшує загальну кількість слів, на яких було навчено.

– if (!wordCounts.containsKey(word)) {...}: це умовний оператор, який перевіряє, чи карта wordCounts для вказаної категорії вже містить вказане слово. Якщо ні, воно додає слово на карту та ініціалізує його рахунок до 1.

– wordCounts.put(word, wordCounts.get(word) + 1): якщо карта wordCounts для вказаної категорії вже містить вказане слово, цей рядок збільшує його кількість на 1.

Таким чином, цей код навчає класифікатор тексту, підраховуючи кількість входжень кожного слова в кожній категорії тексту. Карта categoryCounts відстежує загальну кількість текстів, які навчалися в кожній категорії, тоді як карта wordCountsInCategory відстежує кількість входжень кожного слова в кожній категорії. Змінна totalWords відстежує загальну кількість слів, які були навчені.

Тепер поглянемо на приклад коду з використанням бібліотеки WEKA, який вказаний на рисунку 4.5.

```
ArffLoader loader = new ArffLoader();
loader.setFile(new File( pathname: "text_classification.arff"));
Instances data = loader.getDataSet();

// Set the class attribute
data.setClassIndex(data.numAttributes() - 2);

// Build the classifier
SMO smo = new SMO();
smo.buildClassifier(data);

// Evaluate the classifier
Evaluation eval = new Evaluation(data);
eval.evaluateModel(smo, data);
```

Рисунок 4.5 - Класифікатор за методом опорних векторів

Код для створення класифікатору за методом опорних векторів має наступні рядки:

– loader.setFile(new File("text_classification.arff"));

– Instances data = loader.getDataSet();

Цей розділ ініціалізує об’єкт ArffLoader для завантаження файлу ARFF під назвою «text_classification.arff». Потім він завантажує файл в об’єкт екземплярів під назвою «data».

– data.setClassIndex(data.numAttributes() - 2);

Цей рядок встановлює атрибут класу як останній атрибут у наборі даних.

– SMO smo = new SMO();

– smo.buildClassifier(data);

Цей розділ ініціалізує об’єкт SMO для створення класифікатора з використанням завантаженого набору даних.

Evaluation eval = new Evaluation(data);

eval.evaluateModel(smo, data);

Цей розділ ініціалізує об’єкт Evaluation для оцінки продуктивності класифікатора на завантаженому наборі даних. Метод `evaluateModel()`

викликається з класифікатором SMO та завантаженим набором даних як параметрами для виконання оцінки.

Таким чином, код завантажує файл ARFF, встановлює атрибут класу, створює текстовий класифікатор за допомогою алгоритму SMO та оцінює продуктивність класифікатора на завантаженому наборі даних за допомогою перехресної перевірки.

Інші класифікатори, які реалізовані за допомогою бібліотеки WEKA мають ідентичну структуру (окрім зміну класу класифікатору), та не потребують кардинальних змін в коді. Завдяки використанню даної бібліотеки було значно спрощено розробку коду для деяких класифікаторів завдяки вбудованому в бібліотеку широкому спектру алгоритмів.

Для випадку використання класифікатору в режимі контролеру для Spring Boot застосунку передбачено такий код, який показаний на рисунку 4.6.

```
@RestController
public class ClassifierController {

    private final TelegramClassifierService telegramClassifierService;

    public ClassifierController(TelegramClassifierService telegramClassifierService) {
        this.telegramClassifierService = telegramClassifierService;
    }

    @GetMapping("/classify")
    public String classifyText(@RequestParam String text) { return telegramClassifierService.classify(text); }

}
```

Рисунок 4.6. – Spring Boot контролер для використання класифікатору

Ось коротке пояснення кожного компонента:

– @RestController: анотація, яка використовується для вказівки, що клас є контролером Spring MVC, який обробляє HTTP-запити та повертає відповідь.[23]

– ClassifierController: головний клас, який визначає кінцеві точки REST API.

– TelegramClassifierService: сервісний клас, який виконує класифікацію тексту за допомогою Telegram API.

– classifyText: метод, який обробляє запит GET і приймає текстовий параметр як вхідні дані.

– @RequestParam: анотація, яка використовується для позначення того, що текстовий параметр є параметром запиту.[24]

Для тестування класифікаторів були використані NaiveBayes, K-найближчих сусідів, SMOClassifier в якості методу опорних векторів, Apriori в якості класифікатора на основі асоціативних правил, J48 в якості класифікатора на основі дерева рішень та MultilayerPerceptron як класифікатор на основі штучних нейронних мереж. Точність класифікаторів в відсотках вказана в таблиці 4.2

Таблиця 4.2. – Точність класифікаторів

Алгоритм класифікатору	*Відсоток точності*
NaiveBayes	44,3%
K-найближчих сусідів	67,2%
SMOClassifier	53,1%
Apriori	72,1%
J48	53,6%
MultilayerPerceptron	91%

Найгірший показник в класифікатору NaiveBayes, найкращий в нейронній мережі MultilayerPerceptron. На продуктивність кожного алгоритму може впливати якість даних, розмір набору даних і вибір гіперпараметрів. Зміна гіперпараметрів на інші не дає покращення результату. Даний класифікатор на основі нейронної мережі, якщо не змінювати навчальні дані, буде використаний як основний для запуску в програмному забезпеченні.

4.3 Керівництво користувача

Розроблене програмне забезпечення має 2 варіанти використання. Розглянемо спочатку інструкцію як працювати з сервісом, попередньо інтегрувавши його в власний Java Spring Boot проєкт. Для цього потрібно відкрити клас ClassifierController, який показаний на рисунку 4.7.

```
1  import org.springframework.web.bind.annotation.GetMapping;
2  import org.springframework.web.bind.annotation.RequestParam;
3  import org.springframework.web.bind.annotation.RestController;
4
5  @RestController
6  public class ClassifierController {
7
8      private final TelegramClassifierService telegramClassifierService;
9
10     public ClassifierController(TelegramClassifierService telegramClassifierService) {
11         this.telegramClassifierService = telegramClassifierService;
12     }
13
14     @GetMapping("/classify")
15     public String classifyText(@RequestParam String text) {
16         try {
17             return telegramClassifierService.classify(text);
18         } catch (Exception e) {
19             e.printStackTrace();
20             return e.toString();
21         }
22     }
23
24 }
```

Рисунок 4.7 – Клас ClassifierController

Даний клас потрібно перенести до власного проєкту. Для зміни посилання за яким буде класифікатор приймати запити потрібно відредагувати “/classify” який знаходиться на рисунку 4.7 в рядку номер 14.

В випадку відсутності в архітектурі back-end частини Telegram компонентів на основі Java Spring Boot потрібно його встановити. Щоб інсталювати Java Spring Boot, потрібно виконати такі дії:

По-перше, вам потрібно встановити Java у вашій системі. Ви можете завантажити останню версію Java з офіційного сайту як показано на рисунку 4.8.

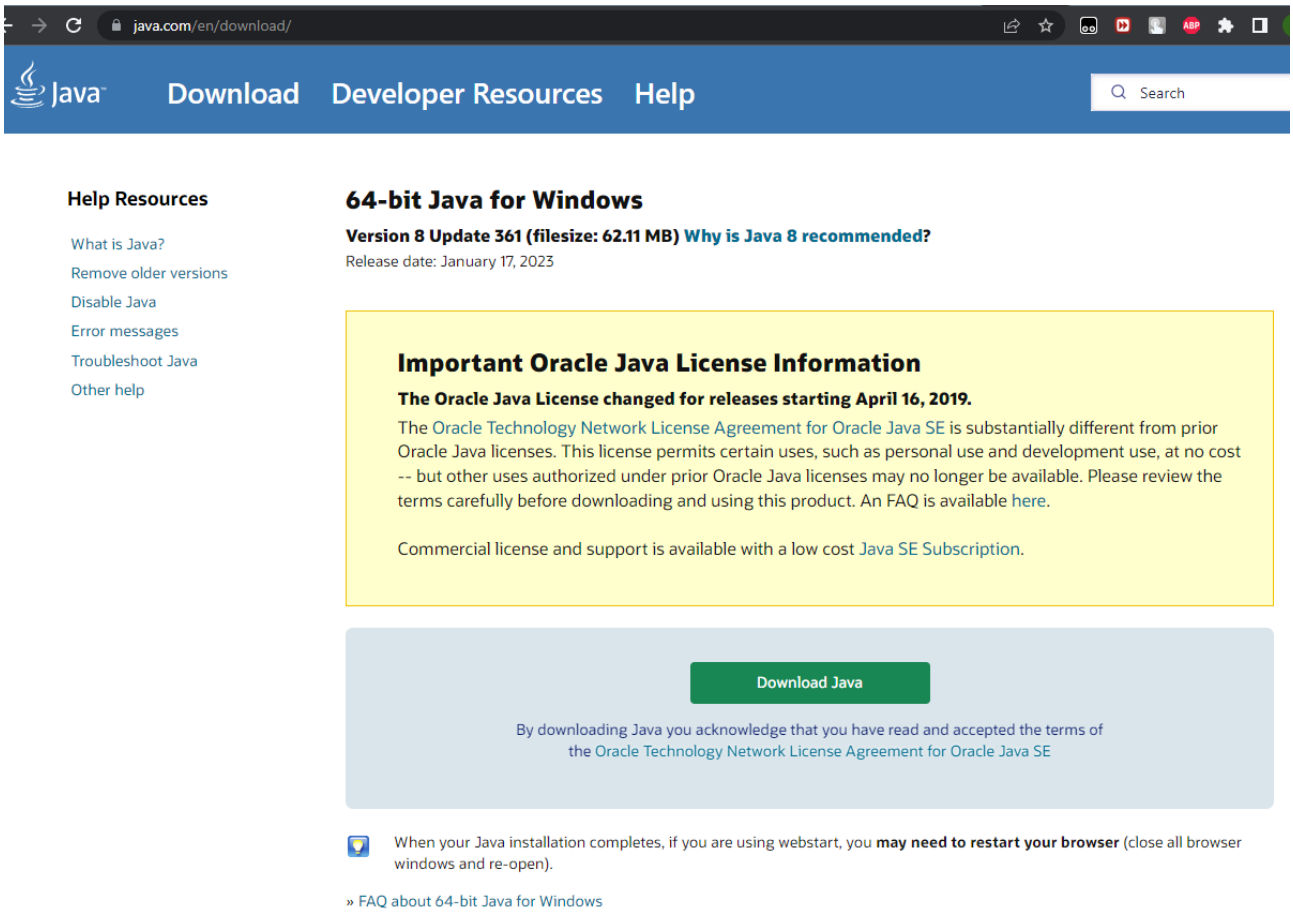

Рисунок 4.8 – Офіційний сайт Java

Після встановлення Java ви можете продовжити завантажувати Spring Boot CLI з офіційного веб-сайту як показано на рисунку 4.9.

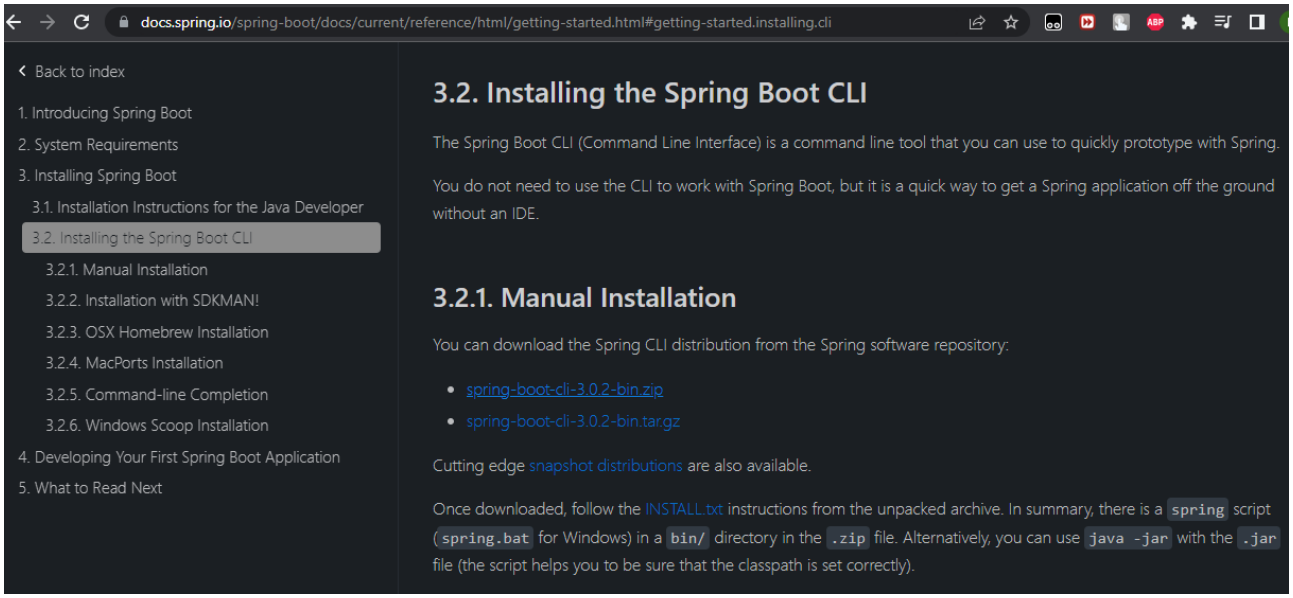

Рисунок 4.9 – Офіційний сайт Spring Boot CLI

Інструкція по встановленню Java наведена всередині встановлювача. Після завантаження Spring Boot CLI розпакуйте вміст zip-файлу в вільне місце у вашій системі.

Далі додайте каталог bin Spring Boot CLI до системної змінної середовища PATH. Це дозволить вам запускати інтерфейс командного рядка Spring Boot з будь-якого каталогу у вашій системі. Щоб зробити це в системі Windows, виконайте такі дії:

– Відкрийте Панель керування та перейдіть до «Система», далі до «Додаткові параметри системи» і до «Змінні середовища».

– У розділі «Системні змінні» виберіть змінну «Шлях» і натисніть «Редагувати».

– Натисніть «Новий» і додайте шлях до каталогу bin Spring Boot CLI, наприклад. C:\spring-2.5.0\bin (замініть «2.5.0» версією Spring Boot, яку ви встановили).

– Натисніть OK, щоб закрити всі вікна.

У системі на основі Unix додайте такий рядок до свого файлу .bashrc або .bash_profile:

export PATH=$PATH:/path/to/spring-boot-cli/bin

Щоб переконатися, що Spring Boot CLI встановлено правильно, відкрийте командний рядок або вікно терміналу та введіть таку команду:

spring --version

Це має відобразити версію Spring Boot, встановлену у вашій системі.Якщо вона відображена вірно, то тоді можна створити новий проект Spring Boot за допомогою Spring Boot CLI. Для цього потрібно відкрити командний рядок або вікно терміналу та перейдіть до каталогу, де ви хочете створити новий проект. Потім введіть таку команду:

spring init --dependencies=web my-project

Це створить новий проект Spring Boot із залежністю «web» (яка включає необхідні бібліотеки для створення веб-додатків) і назвою «my-project». Ви можете замінити "my-project" назвою на свій вибір.[25]

Після створення проекту перейдіть до каталогу проекту та виконайте таку команду, щоб запустити програму:

mvn spring-boot: run

Це запустить програму Spring Boot і зробить її доступною за адресою localhost:8080.

Для запуску сервісу без використання клієнт-серверної архітектури потрібно також завантажити Java з офіційного сайту та встановити. Після неї потрібно встановити IntelIJ IDEA з офіційного сайту як показано на рисунку 4.10.

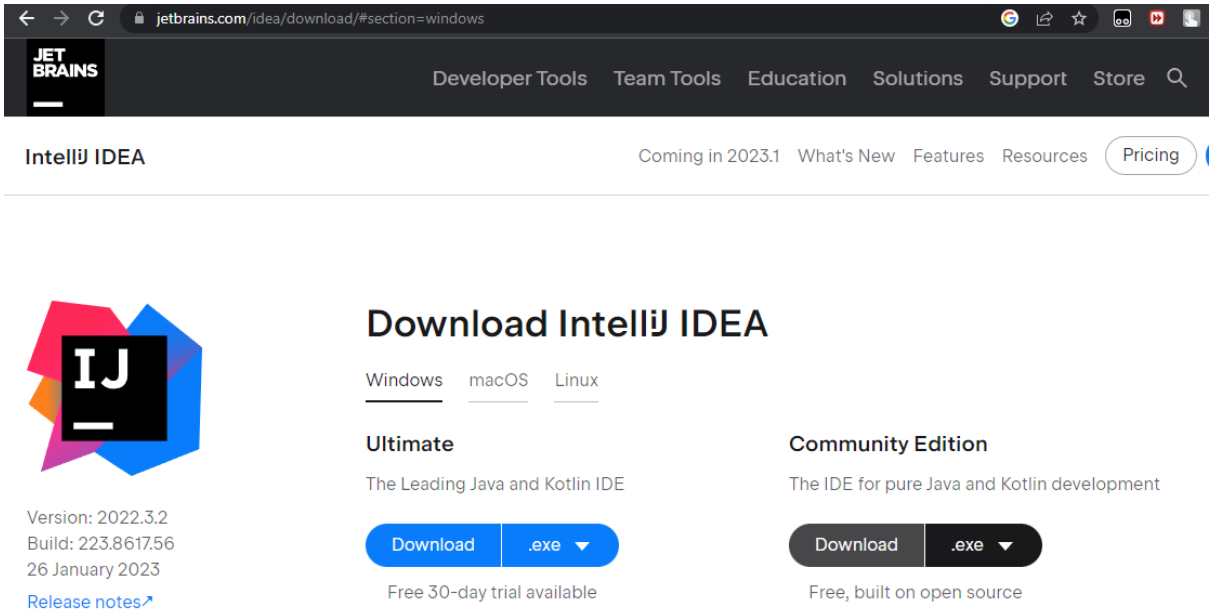

Рисунок 4.10 – Офіційний сайт IntelIJ IDEA

Після завантаження інсталятора потрібно встановити програму дотримуючись інструкцій всередині інсталятора. Далі за допомогою вікна привітання потрібно натиснути Open та відкрити теку з створеним програмним застосунком. Після цього, для проведення передбачення з файлу типу arff або csv потрібно додати сам файл в теку з застосунком та натиснути кнопку “Run” як відображено на рисунку 4.11.

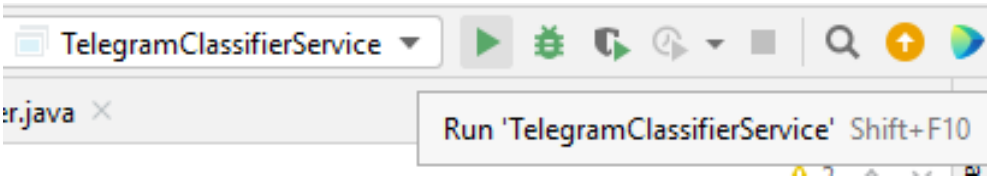

Рисунок 4.11 – Кнопка запуску застосунку

Після запуску застосунку в консоль буде виведена інформація про те як саме прокласифіковано введені дані, приклад зображений на рисунку 4.12.

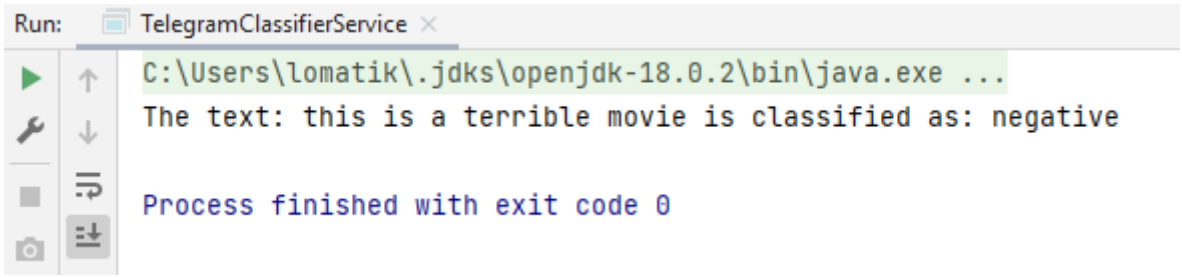

Рисунок 4.12 – Приклад роботи застосунку в консолі

Також в застосунку ведеться запис журналу, тому в випадку великої кількості даних для передбачення, результат передбачення можна буде побачити додатково в логах.

Висновки до розділу 4

Розроблено програмне забезпечення, на навчальних та тестових даних було порівнено класифікаторами які працюють за допомогою різних алгоритмів. Найбільшу точність показав класифікатор на основі штучних нейронних мереж. Більшість вказаних в розділі класифікаторів показали точність менше 70%. Це може бути пов’язане з нерівномірністю даних за класами. Відображені варіанти використання дороблено застосунку.

Для розробників Telegram найкращим варіантом застосування є створення окремого Spring Boot проєкту та розміщення його на виділених серверах.

ВИСНОВКИ

За час виконання кваліфікаційної роботи магістра було досліджено процес класифікації коментарів, отриманих з різних публічних каналів мессенджеру Telegram за допомогою алгоритмів text mining задля розширення функціоналу Telegram для стейкхолдерів, які зацікавлені у використанні статистики, отриманої шляхом аналізу суспільної думки.

Завдання кваліфікаційної роботи виконано, програмне забезпечення для класифікації текстових даних за настроєм з найточнішим алгоритмом серед наявних створено.

Для повноцінного перетворення Телеграм в соціальну мережу створено програмний застосунок, який аналізує коментарі, та визначає суспільну думку користувачів месенджеру про певні речі та теми. Завдяки подальшому збору думок формуватиметься статистика, яка буде корисна під час ведення SMM діяльності, Однією з основних цілей якої є ефективна комунікація, популяризація та реклама продуктів компанії. Ці компанії використовують соціальні мережі щоб залучити більше клієнтів. Це єдина форма маркетингу, яка може демонструвати кожний етап покупки.

Для реалізації поставленого завдання кваліфікаційної роботи застосовано алгоритми класифікації. Перед тим як використовувати їх потрібно обробити природню мову для подальшої обробки класифікатором. Досліджено, що існують такі моделі як WordNet, WER, Bag-of-words, TD-IDF, NER. Спільна суть даних моделей полягає в фільтруванні інформації для покращеної роботи алгоритмів класифікації за допомогою видалення зайвих символів, таких як знаки пунктуації, пробіли, табуляції, а також моделі, кожна по-своєму, видаляють непотрібні слова з тексту, які можуть заважати суті та зменшувати точність алгоритмів класифікації.

Для розробки програмного забезпечення було обрано мову програмування Java через її швидкодію, різноманіття бібліотек для класифікації тексту та сувору

типізацію. В якості набору бібліотек для text mining було обрано WEKA, через відкритість коду та велике різноманіття алгоритмів.

Розроблено програмне забезпечення, на навчальних та тестових даних порівняні створені класифікатори які працюють за допомогою різних алгоритмів. Найбільшу точність показав класифікатор на основі штучних нейронних мереж з показником який дорівнює 91%. Більшість інших класифікаторів мають точність менше 70%. Це може бути пов’язано з нерівномірністю даних за класами. Відображені варіанти використання дороблено застосунку.

Для розробників Telegram найкращим варіантом застосування є створення окремого Spring Boot проєкту та розміщення його на виділених серверах. Також в випадку відсутності можливості застосування окремих серверів, можна застосовувати сервера на яких також знаходяться інші елементи інформаційної системи Telegram, але за умови достатньої кількості оперативної пам’яті для всіх процесів.

Рекомендується під час використання програмного застосунку періодично оновлювати навчальні дані, для більшого уточнення можливих настроїв користувачів, оскільки в використаному наборі даних, настрої розподілені нерівномірно, так як більшість настроїв нейтральні або переживаючі. Періодичне оновлення буде збільшувати точність класифікації настрою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dave E. Social media marketing. Wiley & Sons, Incorporated, John, 2012.

2. Hart I. Social media marketing: guide to social media marketing, social media marketing strategies. Isabella Hart, 2020. 70 p.

3. Holland P. Social-Media-Marketing. Independently Published, 2022.

4. Solomon M. R., Tuten T. L. Social media marketing. SAGE Publications, Limited, 2017. 448 p.

5. Tuten T. L. Social media marketing. SAGE Publications, Limited, 2020. 480 p.

6. Девід Робінсон, Юлія Сільге Text Mining with R: A Tidy Approach : книга. London : 2012. 204 с.

7. Девід Робінсон Text Mining: Classification, Clustering : книга. London : Bookboon Learning, 2009. 164 с.

8. Якоб Коган Text Mining: Applications and Theory : книга. London : 2014. 235 с.

9. Ханько Г.В. Гавриленко О.А. Олійник Ю.В. Аналіз підходів до виявлення елементів агітації в текстових даних. Наукова конференція студентів, магістрантів та аспірантів «Інформатика та обчислювальна техніка» – 2018 Київ: С. 162-165

10. Ханько Г.В. Гавриленко О.А. Олійник Ю.В. Огляд та аналіз алгоритмів Text Mining: тези VII Міжнародна науково-технічна конференція “Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління”, 2018 - Полтава, с. 120-124

11. Hearst M. A. Text Data Mining. Oxford University Press, 2012. URL: https://doi.org/10.1093/oxfordhb/9780199276349.013.0034 (date of access: 08.12.2022).

12. Jo T. Text Mining. Cham : Springer International Publishing, 2019. URL: https://doi.org/10.1007/978-3-319-91815-0 (date of access: 08.12.2022).

13. Text Mining / ed. by A. N. Srivastava, M. Sahami. Chapman and Hall/CRC, 2009. URL: https://doi.org/10.1201/9781420059458 (date of access: 08.12.2022).

14. Text Mining / ed. by M. W. Berry, J. Kogan. Chichester, UK : John Wiley & Sons, Ltd, 2010. URL: https://doi.org/10.1002/9780470689646 (date of access: 20.01.2023).

15. Cleve J. Data mining. Müchen : De Gruyter Oldenbourg, 2014. 306 p.

16. Grunitzky G. Java programming: java. Independently Published, 2017.

17. Dharani R. Java basics ,java fundamentals and advanced java. Independently Published, 2018.

18. Hofmann M., Klinkenberg R. RapidMiner: data mining use cases and business analytics applications. Taylor & Francis Group, 2016. 525 c.

19. Lämmel U., Cleve J. Data mining. de Gruyter GmbH, Walter, 2020.

20. Deblauwe W. Practical guide to building an API back end with spring boot. Lulu Press, Inc., 2018.

21. Doglio F. Pro REST API development with node.js. Berkeley, CA : Apress, 2015. URL: https://doi.org/10.1007/978-1-4842-0917-2 (date of access: 21.01.2023).

22. Sushama Pawar Chetashri Bhusari Yogita Jore. Java programming. Independently Published, 2020.

23. Tudose C. Java persistence with spring data and hibernate. Manning, 2022. 625 p.

24. Introduction to java spring boot: learning by coding / A. Ankomah et al. Independently Published, 2019.

25. Gutierrez F. Spring boot fundamentals. Berkeley, CA : Apress, 2021. URL: https://doi.org/10.1007/978-1-4842-7066-0 (date of access: 16.02.2023).