КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

«Підсистема сентиментального аналізу коментарів про нові релізи кіно і телебачення на основі методів NLP»

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма «Технології штучного інтелекту»

Виконав:

студент 2 курсу магістратури, групи ТШІ-21 Підгурський В. О.
(ПІБ)

Науковий керівник: Тмєнова Н. П.
(ПІБ)

к.ф.-м.н., доцент
(науковий ступінь, вчене звання)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____
підпис

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № 12 від «11» травня 2023 р.

Зав. кафедри _____ ПІБ
підпис

Київ – 2023

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаної літератури із 20 джерел та 1 додатку. Загальний обсяг роботи 97 сторінок. Робота містить 9 таблиць та 16 рисунків.

Актуальність теми. Різноманітні рішення, які стосуються виробництва кіно- та телепродуктів приймаються аналогічно до інших бізнесів – шляхом аналізу маркетингових досліджень. Раніше такий аналіз мав би проводитися за допомогою опитувань або апостеріорного дослідження реакцій споживачів на вже випущений продукт для застосування отриманих знань у майбутньому. Проте розвиток інформаційних технологій презентує нові шляхи вирішення даної проблеми.

Соціальні мережі спонукають людей ділитися думками про все без винятку, в тому числі переглянуті та очікувані кінострічки та телесеріали. Це дає аналітикам у користування значно більший обсяг різноманітних даних для аналізу, а головне – безкоштовно, оскільки більше немає необхідності витрачати ресурси на традиційні маркетингові дослідження. Також це дає можливість оцінювати реакції аудиторії на прийняті рішення в реальному часі.

Мета роботи: Розробка програмного забезпечення для аналізу коментарів про нові релізи кіно та серіалів шляхом застосування сентиментального аналізу до релевантних текстів з мікро-блогів.

Об’єкт дослідження - застосування сентиментального аналізу до текстів із мікроблогів із використанням методів NLP.

Предмет дослідження - програмна система для сентиментального аналізу текстів із мікроблогів.

Результати роботи. програмна система, яка завантажує коментарі глядачів з таких ресурсів як Twitter та YouTube, проводить їх сентиментальний аналіз та підсумовує отриманні результати для використання у подальшому аналізі даних.

Ключові слова. Кіно, телебачення, сентиментальний аналіз, вибір ознак, класифікація.

ABSTRACT

The qualification work consists of an introduction, 3 chapters, conclusions, a list of references from 20 sources and 1 appendix. The total amount of work is 97 pages. The work contains 9 tables and 16 figures.

Relevance of the topic. Various decisions related to the production of film and television products are made similarly to other businesses – through the analysis of marketing research. Previously, such an analysis should have been carried out using surveys or a posteriori study of consumer reactions to an already released product to apply the gained knowledge in the future. However, the development of information technology presents new ways to solve this problem.

Social networks encourage people to share their thoughts about anything at all, including watched and expected films and television series. This gives analysts a much larger amount of various data for analysis, and most importantly – for free, since there is no longer a need to spend resources on traditional marketing research. It also makes it possible to evaluate the audience's reactions to decisions made in real time.

Objective: Development of software for analyzing comments on new releases of movies and series by applying sentimental analysis to relevant texts from micro-blogs

The object of research is the application of sentimental analysis to texts from microblogs using NLP methods

Subject of research - software system for sentimental analysis of texts from microblogs

Work results. a software system that downloads viewers' comments from resources such as Twitter and YouTube, conducts their sentimental analysis and summarizes the results for use in further data analysis.

Keywords. Cinema, television, sentimental analysis, choice of signs, classification.

ЗМІСТ

ВСТУП

Нині кінематограф та телебачення – це в першу чергу бізнес, і лише потім мистецьке вираження. Відповідно, різноманітні рішення, які стосуються виробництва кіно- та телепродуктів приймаються аналогічно до інших бізнесів – шляхом аналізу маркетингових досліджень. Раніше такий аналіз мав би проводитися за допомогою опитувань або апостеріорного дослідження реакцій споживачів на вже випущений продукт для застосування отриманих знань у майбутньому. Проте розвиток інформаційних технологій презентує нові шляхи вирішення даної проблеми.

Соціальні мережі спонукають людей ділитися думками про все без винятку, в тому числі про переглянуті та очікувані кінострічки та телесеріали. Це дає аналітикам у користування значно більший обсяг різноманітних даних для аналізу, а головне – безкоштовно, оскільки більше немає необхідності витрачати ресурси на традиційні маркетингові дослідження. Також це дає можливість оцінювати реакції аудиторії на прийняті рішення в реальному часі. Наприклад, після випуску першого трейлеру фільму «Їжак Сонік» (2019), дизайн титульного персонажа отримав чимало критики у соціальних мережах, після чого студія Paramount Pictures перенесла реліз стрічки та повністю змінила дизайн їжака. Проте, постає проблема аналізу цієї маси даних, оскільки людина не спроможна обробити таку їхню кількість.

Обробка природномовної інформації (NLP) – це міждисциплінарна галузь, яка поєднує лінгвістику, комп’ютерні науки та штучний інтелект для дослідження взаємодії комп’ютерів та природної мови. Методи NLP можуть допомогти з автоматизацією аналізу дописів у соціальних мережах.

Метою даної роботи є створення системи для автоматичного аналізу думок людей щодо нових та майбутніх кіно- та телерелізів.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРОБЛЕМИ СЕНТИМЕНТАЛЬНОГО АНАЛІЗУ ТЕКСТІВ З МІКРОБЛОГІВ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Область застосування розроблюваної системи

Для максимізації прибутків компанії-виробники, навіть ті, чий продукт мав би бути мистецьким, вираженням людської суб’єктивності, вдаються до об’єктивної математичної аналітики. Індиферентні комп’ютеризовані системи приймають рішення щодо оптимізації процесів виробництва і потребують для цього велику кількість вхідних даних. Це можуть бути як явні дані, такі як фінансові збори фільмів, а можуть бути і неявні, отримані внаслідок більш глибинного аналізу. Наприклад, знання про рішення, які приймають конкурентами, такі як ринки на які вони орієнтуються, особливості каталогів тощо, або знання про думки споживачів щодо попередніх, прийнятих компанією рішень.

Проте, все частіше, збором та аналізом даних займаються не самі компанії-виробники, а аналітичні компанії. Вони збирають дані з різноманітних джерел, аналізують та продають консолідовані результати досліджень компаніям-виробникам. Компанії-виробники звертаються до великої кількості аналітичних компаній, аби мати якомога більш повне уявлення про медіа-ринки. Іншими клієнтами аналітичних компаній можуть бути різноманітні, переважно економічні, журналістські видання, які висвітлюють становище цих ринків.

1.2 Опис профілів зацікавлених сторін

Проведемо аналіз профілів зацікавлених сторін (Таблиця 1.1)

Таблиця 1.1 Опис профілів зацікавлених сторін

Зацікавлена сторона	Здобута вигода	Основні інтереси	Обмеження
Компанії-виробники медіа-продуктів	1. Отримання більшої кількості прибутку за наявності результатів аналізу з великою кількістю даних 2. Усунення невизначеності та ризиків у процесі прийняття рішень	1. Отримання більшої кількості прибутку шляхом виробництва медіа-контенту, який привабить більшу кількість споживачів	1. Результати аналізу мають достовірно відображати медіа-ринок
Аналітичні компанії	1. Отримання більшої кількості диверсифікованих даних для проведення аналізу	1. Отримання прибутку шляхом продажу результатів аналізу компаніям-виробникам	1. Дані мають бути придатними для аналізу
Розробники	1. Можливість втілення ідеї системи для визначення емоційного контексту популярності релізів кіно та телебачення 2. Отримання досвіду роботи NLP.	1. Втілення ідеї системи для визначення емоційного контексту популярності релізів кіно та телебачення	1. Розробка системи має бути здійсненною.

1.3 Загальний огляд сентиментального аналізу

Сентиментальний аналіз – це галузь, що розвивається на стику лінгвістики та інформатики, яка намагається автоматично визначити думки та настрої, що містяться в тексті [1]. Сентиментальний аналіз тепер є поширеним інструментом у репертуарі аналізу соціальних мереж, який проводять компанії, маркетологи та політичні аналітики.

Об’єктом дослідження сентиментального аналізу є сентимент – суб’єктивна думка або відношення стосовно якогось об’єкту. Р. Квірк та інші [2] визначають суб’єктивність «приватними» станами – це дієслова стану, які не можуть бути об’єктивно верифікованими. Це інтелектуальні стани (знати, вірити, припускати, уявляти, розуміти), стани емоцій чи відношень (хотіти, бажати, любити, ненавидіти, жаліти), стани сприйняття (бачити, чути, відчувати, чути носом, куштувати) і стани тілесних відчуттів (відчувати біль, лоскіт, свербіж, холод).

Суб’єктивні елементи — це лінгвістичні вираження приватних станів у контексті [3]. Суб’єктивні елементи часто є лексичними. Це можуть бути окремі слова (скаржиться) або складені вирази (бути враженим). Чисто синтаксичні чи морфологічні прийоми також можуть бути суб’єктивними елементами (наприклад, зміна порядку слів у реченні, паралелізм тощо).

Суб’єктивний елемент виражає суб’єктивність джерела, яким може бути автор або хтось, згаданий у тексті. Крім того, суб'єктивний елемент зазвичай має ціль, тобто те, чого суб'єктивність стосується або на що спрямована. Проте, на практиці, зазвичай враховується лише полярність та ціль суб’єктивних елементів.

Таким чином, сентимент визначають як встановлення того, яке забарвлення має суб’єктивність тексту – позитивне чи негативне [4].

Сентимент також має кілька унікальних властивостей, які відрізняють його від інших якостей, які ми можемо відстежувати в тексті [5]. Часто ми хочемо класифікувати текст за темами, що передбачає роботу з цілими таксономіями тем. З іншого боку, сентиментальна класифікація зазвичай має справу з двома класами (позитивний та негативний), діапазоном полярності (наприклад, оцінка для фільмів) або навіть діапазоном сили переконання. Ці класи охоплюють багато тем, користувачів і типів документів. Хоча, коли ми маємо справу лише з кількома класами – це може здатися легшим завданням, ніж стандартний аналіз тексту, це, насправді, далеко від істини.

Як наукова галузь, сентиментальний аналіз тісно пов’язаний з комп’ютерною лінгвістикою, обробкою природної мови та аналізом тексту (або може розглядатися як його частина). Виходячи з дослідження афективного стану (психологія) і судження (теорія оцінки), ця галузь прагне відповісти на питання, які давно вивчаються в інших сферах дискурсу, використовуючи нові інструменти, надані аналізом даних і комп’ютерною лінгвістикою.

Сентиментальний аналіз має багато назв. Його часто називають аналізом суб’єктивності, дослідженням думок і вилученням оцінок. Він також пов’язаний з емоційним обчисленням (комп’ютерне розпізнавання та вираження емоцій). Сентиментальний аналіз зазвичай досліджує суб'єктивні елементи. Переважно це окремі слова, фрази чи речення. Іноді цілі документи вивчаються як сентиментальні одиниці настрою, але загальновизнано, що сентименти містяться в менших лінгвістичних одиницях.

Сентименти, які з’являються в тексті, мають два види: явні, коли суб’єктивне речення прямо виражає думку («Це чудовий день»), і неявні, коли текст лише натякає на думку («Навушник зламався за два дні»). Більшість роботи, виконаної досі, зосереджена на першому типі сентиментів, оскільки його легше аналізувати.

Полярність сентиментів є особливою ознакою тексту. Зазвичай її поділяють на два класи – позитивний і негативний, але полярність також можна розглядати як діапазон. Документ, що містить твердження з різними думками, матиме змішану полярність, яка відрізняється від відсутності полярності взагалі (об’єктивності). Крім того, слід розрізняти полярність сентименту та його силу. Хтось може повністю переконатися, що продукт є нормальним, не дуже хорошим чи поганим; або бути слабко переконаним в тому, що продукт дуже хороший (оскільки, можливо, хтось мав цей продукт у користуванні протягом надто короткого часу, щоб сформувати чітку думку).

Іншою важливою частиною сентименту є його ціль - об'єкт, концепція, людина, тощо. Найбільше роботи виконано над рецензіями на товари та фільми, де легко визначити тему тексту. Але часто варто звернути увагу на те, про яку особливість цього об’єкта говорить автор: дисплей камери чи час автономної роботи найбільше турбує споживачів? Згадки цих особливостей у тексті також можуть бути явними («Ємність батареї надто мала») або неявними («Камера завелика»).

На відміну від звичайного тематичного аналізу, авторство сентименту може бути невід’ємною частиною проблеми. Однією з головних проблем є цитування. Важливо знати, чи сентименти, висловлені в документі, відображають справжні думки автора. Політичні коментарі та новини переповнені цитатами і можуть мати заплутану структуру, яку важко розрізнити. Наприклад, у статті новин про політичні дебати буде поєднання цитат учасників дебатів, експертів, які коментують дебати, і, можливо, навіть позиція автора щодо обговорюваних проблем.

Через складність проблеми сентиментальний аналіз складається з ряду окремих завдань. Зазвичай вони поєднуються, щоб отримати певні знання про думки, які містяться в тексті.

Першим завданням є виявлення сентиментів або думок, що можна розглядати як класифікацію тексту як об’єктивного чи суб’єктивного. Зазвичай визначення сентиментів ґрунтується на перевірці прикметників у реченнях. Наприклад, полярність «це гарна картинка» можна легко визначити, подивившись на прикметник.

Друге завдання — класифікація полярності. Якщо ми маємо текст із думкою, то наша мета - класифікувати думку як таку, що підпадає під одну з двох протилежних полярностей сентиментів, або визначити її позицію в просторі між цими двома полярностями. Якщо розглядати її як бінарну ознаку, класифікація полярності — це завдання бінарної класифікації, яка полягає в позначенні документа як такого, що виражає загалом позитивну чи загалом негативну думку. Більшість досліджень було проведено на рецензіях продуктів, де визначення «позитивного» та «негативного» є чіткими. Інші завдання, такі як класифікація новин як «позитивні» чи «негативні», викликають певні труднощі. Стаття новин може містити «негативні» новини без фактичного використання будь-яких суб’єктивних термінів. Крім того, ці класи зазвичай змішуються, коли документ виражає як позитивні, так і негативні почуття. Тоді завдання може полягати в тому, щоб визначити головний сентимент документа.

Щоб розрізнити різні змішування двох протилежностей, у класифікації полярності використовується інтервальна шкала (наприклад, оцінка для рецензії фільму). Тут завдання перетворюється на проблему категоризації тексту з кількома класами. Але на відміну від задач класифікації на основі кількох класів, де словники відрізняються для кожного класу (або трохи накладаються), словники для позитивних, нейтральних і негативних класів можуть бути дуже схожими і відрізнятися лише кількома важливими словами. Оскільки багато документів мають «змішану» точку зору, цей клас насправді є поєднанням позитивного та негативного. Заперечення, які, як правило, ігноруються в аналізі

тексту як неважливі, відіграють важливу роль у сентиментальному аналізі, перетворюючи початково позитивний термін на негативний і навпаки.

Наведені вище два завдання можна виконувати на кількох рівнях: термін, фраза, речення або документ. Зазвичай вихідні дані одного рівня використовують як вхідні дані для вищих рівнів. Наприклад, можна застосувати сентиментальний аналіз до фраз, а потім використовувати цю інформацію для оцінки речень, потім абзаців тощо. Для різних рівнів підходять різні методи. Методи з використанням класифікаторів n-грам або лексиконів зазвичай працюють на рівні термінів, тоді як розмічування частин мови використовується для аналізу фраз і речень. Евристика часто використовується для узагальнення настроїв на рівні документа.

Третім завданням, яке доповнює сентиментальний аналіз, є виявлення цілі сентименту. Складність цього завдання значною мірою залежить від області аналізу. Як згадувалося раніше, зазвичай можна з упевненістю припустити, що відгуки про продукт переважно говорять про вказаний продукт. З іншого боку, загальні тексти, такі як веб-сторінки та блоги, не завжди мають заздалегідь визначену тему та часто згадують багато об’єктів.

Іноді сентименти мають кілька цілей, наприклад, в порівняльних реченнях. Суб’єктивне порівняльне речення впорядковує об’єкти в порядку переваг, наприклад, «цей фотоапарат кращий за мій старий». Ці речення можна ідентифікувати за допомогою порівняльних прикметників і прислівників (більше, менше, краще, довше), прикметників у найвищому ступені (найбільше, найменше, найкраще) та інших слів(однакові, відрізняються, виграють, віддають перевагу тощо).

Одна з особливостей сентиментів полягає в тому, що хоча поняття позитивної та негативної думки є загальними, вираження цих думок сильно відрізняється в спектрі тематичних сфер. Таким чином, однотематичний та міжтематичний аналіз настроїв вивчається з метою покращення продуктивності

в певній області. Важливим питанням тут є поєднання загальних знань про вираження почуттів і тих, що стосуються конкретної теми. У міжтематичному аналізі ідея полягає в тому, щоб використовувати знання, зібрані про одну область, в іншій.

1.4 Аналіз методів вибору ознак

Завданням сентиментального аналізу вважається класифікація сентиментів [6]. Першим кроком у задачі сентиментального налізу є вилучення та вибір текстових ознак. Деякі з них:

- **Наявність і частота термінів** - ці ознаки є окремими словами або n-грамами слів і їх частотними показниками. Вони або надають словам двійкову вагу (нуль, якщо слово з’являється, одиницю, якщо ні), або використовує частотні ваги термінів, щоб вказати відносну важливість ознак.
- **Частини мови** - пошук прикметників, оскільки вони є важливими показниками думок.
- **Слова та фрази, що висловлюють думку** - це слова, які зазвичай використовуються для вираження думок (хороший, поганий, подобатися, ненавидіти).
- **Заперечення** - поява негативних слів може змінити полярність думки, наприклад «не хороший» еквівалентно «поганому».

Методи вибору ознак можна розділити на методи на основі лексики, які вимагають ручної анотації від людини, та статистичні методи, які є автоматичними методами та використовуються частіше. Підходи, засновані на лексиконах, зазвичай починаються з невеликого набору «початкових» слів. Потім вони розширюють цей набір за допомогою виявлення синонімів або

онлайнових ресурсів, щоб отримати більший лексикон. Статистичні підходи, з іншого боку, є повністю автоматичними.

Методи вибору ознак обробляють документи або як групу слів (Bag of Words (BoW)), або як рядок, що зберігає послідовність слів у документі. BoW використовується частіше через його простоту в процесі класифікації. Найпоширенішим етапом вибору ознак є видалення стоп-слів і стемінг.

Одними із найпоширеніших статистичних методів, що використовуються при виборі ознак, є:

- Поточкова взаємна інформація (ПВІ);
- Хі в квадраті (χ^2);
- Латентне семантичне індексування (ЛСІ).

Міра взаємної інформації забезпечує формальний спосіб моделювання взаємної інформації між ознаками та класами. Поточкова взаємна інформація (ПВІ) $M_i(w)$ між словом w і класом i визначається на основі рівня спільної появи класу i і слова w. Нехай маємо частоту появи слова w - $F(w)$, ймовірність появи класу i - P_i, ймовірність спільної появи слова w і класу i - $p_i(w)$. Тоді, очікувана спільна поява класу i та слова w, на основі взаємної незалежності, визначається як $F(w) \cdot P_i$, а справжня спільна поява задається $F(w) \cdot p_i(w)$.

Взаємна інформація визначається співвідношенням між цими двома величинами та визначається наступним рівнянням:

$$M_i(w) = \log\left(\frac{F(w) \cdot p_i(w)}{F(w) \cdot P_i}\right) = \log\left(\frac{p_i(w)}{P_i}\right), \qquad (1.1)$$

Слово w позитивно корелює з класом i, якщо $M_i(w)$ більше 0. Слово w негативно корелює з класом i, коли $M_i(w)$ менше 0.

Міра χ^2 визначається наступним чином. Нехай n — загальна кількість документів у наборі, $p_i(w)$ — умовна ймовірність класу i для документів, які містять w, P_i — глобальна частка документів, що містять клас i, а $F(w)$ — глобальна частка документів, які містять слово w. Отже, міра χ^2 між словом w і класом i визначається як:

$$\chi_i^2 = \frac{n \cdot F(w)^2 \cdot (p_i(w) - P_i)^2}{F(w) \cdot (1 - F(w)) \cdot P_i \cdot (1 - P_i)} \tag{1.2}$$

χ^2 і ПВІ — це два різні способи вимірювання кореляції між термінами та класами. χ^2 краще, ніж ПВІ, оскільки це нормалізоване значення, а отже, ці значення краще порівнюються для термінів в одній категорії.

Методи вибору ознак призначені для того, щоб зменшити розмірність даних шляхом вибору меншого набору ознак з оригінального набору. Методи трансформації ознак створюють менший набір ознак як функції оригінального набору ознак. ЛСІ є одним із найбільш відомих методів перетворення ознак. Метод ЛСІ перетворює текстовий простір на систему осей, яка є лінійною комбінацією оригінальних ознак слова. Для досягнення цієї мети використовується метод головних компонент (МГК). Він визначає осьову систему, яка зберігає найбільший рівень інформації про варіації базових значень атрибутів. Основним недоліком ЛСІ є те, що це неконтрольована техніка, яка не опирається на існуючий класовий розподіл. Таким чином, функції, знайдені ЛСІ, не обов’язково є напрямками, за якими можна найкраще відокремити класовий розподіл базових документів.

1.5 Аналіз методів класифікації сентиментів

Методи класифікації сентиментів можна розділити на методи машинного навчання, методи на основі лексикону та гібридні методи [6]. Методи машинного навчання (МН) застосовують відомі алгоритми МН і використовує лінгвістичні ознаки. Методи на основі лексики, опираються на сентиментальну лексику, набір відомих і попередньо скомпільованих сентиментальних термінів. Вони поділяються на методи на основі словника та методи на основі корпусу, які використовують статистичні або семантичні методи для визначення полярності сентиментів. Гібридні методи поєднують в собі обидва підходи та дуже поширені, оскільки сентиментальні лексикони відіграють ключову роль у більшості методів.

1.5.1 Методи машинного навчання

Методи машинного навчання використовує відомі алгоритми МН для вирішення сентиментального аналізу як звичайної проблеми класифікації тексту з використанням синтаксичних та/або лінгвістичних особливостей тексту.

Методи класифікації тексту з використанням методів МН можна грубо розділити на методи навчання з учителем та без. Методи навчання з учителем використовують велику кількість розмічених навчальних документів. Методи навчання без учителя використовуються, коли важко знайти ці розмічені навчальні документи.

Методи навчання з учителем залежать від наявності розмічених навчальних документів. Існує багато видів класифікаторів, які застосовують навчання з учителем. Основні з них:

- ймовірнісні класифікатори;
- лінійні класифікатори;
- дерева рішень;
- класифікатори на основі правил.

Ймовірнісні класифікатори і, зокрема, архетиповий наївний класифікатор Баєса, є одними з найпопулярніших класифікаторів, які використовуються в машинному навчанні та все частіше в багатьох прикладних програмах [7]. Ці класифікатори походять від генеративних ймовірнісних моделей, які забезпечують структурований спосіб для дослідження статистичної класифікації в складних областях, таких як природна мова та візуальна обробка. Дослідження ймовірнісної класифікації — це дослідження наближення спільного розподілу з розподілом добутку. Правило Баєса використовується для оцінки умовної ймовірності віднесення об’єкту до певного класу, а потім робляться припущення щодо моделі, щоб розкласти цю ймовірність на добуток умовних ймовірностей.

Ймовірнісні класифікатори використовують сумішеві моделі для класифікації. Сумішева модель передбачає, що кожен клас є компонентом суміші. Кожен компонент суміші є генеративною моделлю, яка забезпечує ймовірність віднесення певного терміну до цього компонента. Тобто, ймовірнісні класифікатори виконують не жорстку класифікацію, коли об’єкт відноситься до певного єдиного класу, а м’яка – коли для кожного класу визначається ймовірність належності цього об’єкту.

Лінійні класифікатори є корисним інструментом у машинному навчанні та аналізі даних [8]. На відміну від нелінійних класифікаторів, таких як ядрові

методи, які відображають дані у просторі вищої розмірності, лінійні класифікатори працюють безпосередньо із вхідними даними у оригінальній розмірності. Лінійні класифікатори продемонстрували конкурентоспроможність у порівнянні з нелінійними класифікаторами. Важливою перевагою лінійних класифікаторів є те, що процедури навчання та тестування є значно ефективнішими. Тому використання лінійних класифікаторів може бути дуже корисним для деяких великомасштабних програм.

Лінійні класифікатори можна визначити наступним чином. Нехай маємо $\bar{X} = \{x_1, \dots, x_n\}$ — нормалізовані частоти слів документа, вектор $\bar{A} = \{a_1, \dots, a_n\}$— вектор лінійних коефіцієнтів з тією ж розмірністю, що й простір ознак, а b — скаляр. Тоді, виходом лінійного класифікатора є лінійний предикат, який визначається наступним чином:

$$p = \bar{A}.\bar{X} + b \tag{1.3}$$

Предикат p є роздільною гіперплощиною між різними класами. Існує багато видів лінійних класифікаторів, один з них – опорно-векторні машини (ОВМ), вид класифікаторів, які намагаються визначити хороші лінійні розділювачі між різними класами.

Дерева рішень забезпечують ієрархічну декомпозицію простору навчальних даних, у якому умова до значення атрибута використовується для розділення даних. Умова або предикат — це наявність або відсутність одного чи кількох слів. Поділ простору даних виконується рекурсивно, поки листові вузли не містять певну мінімальну кількість записів, які використовуються з метою класифікації.

Існують інші типи предикатів, які залежать від подібності документів для кореляції наборів термінів, які можуть використовуватися для подальшого поділу документів:

- Поділ за одним атрибутом, який використовує наявність або відсутність певних слів або фраз у певному вузлі дерева, щоб виконати поділ;
- Багатоатрибутний поділ на основі подібності, який використовує документи або кластери часто вживаних слів і подібність документів до цих кластерів слів для виконання розбиття;
- Багатоатрибутний поділ на основі дискримінантів використовує дискримінанти, такі як лінійний дискримінант Фішера, для виконання поділу.

У класифікаторах на основі правил простір даних моделюється за допомогою набору правил. Ліва сторона представляє умову до набору ознак, виражену в диз'юнктивній нормальній формі, тоді як права сторона є міткою класу. Умови встановлюються до присутності термінів. Відсутність термінів використовується рідко, оскільки вона не є інформативною у розріджених даних.

Існує кілька критеріїв для створення правил, на етапі навчання будуються всі правила залежно від цих критеріїв. Найбільш поширеними критеріями є підтримка та впевненість. Підтримка — це абсолютна кількість екземплярів у навчальному наборі даних, які відносяться до правила. Впевненість стосується умовної ймовірності того, що права частина правила виконується, якщо виконується ліва сторона.

Як дерева рішень, так і класифікатори на основі правил, зазвичай, кодують правила в просторі ознак, але дерева рішень досягають цієї мети за допомогою ієрархічного підходу. Основна відмінність між деревами рішень і класифікаторами на основі правил полягає в тому, що дерева рішень є суворим

ієрархічним поділом простору даних, тоді як класифікатори на основі правил допускають перекриття в просторі рішень.

У випадку навчання без учителя завдання полягає в тому, щоб виявити приховані зв’язки, структури, асоціації чи ієрархії на основі зразків даних, наданих системі [9]. Така інформація дає нам краще розуміння даних і основних процесів, які їх створюють. Кластери можуть бути побудовані на основі показників подібності або відстані, тому схожі зразки належать до одного кластера. Міру подібності також можна описати як міру відстані, оскільки подібність двох зразків можна інтерпретувати як відстань між двома зразками в просторі ознак.

Один із підходів для кластеризації шляхом навчання без учителя полягає у використанні цих мір подібності та побудові областей простору ознак, що відповідають різним кластерам, на основі обчислених відстаней між навчальними вибірками.

Інший підхід полягає у виділенні ознак з високою силою розрізнення або пошуку основних векторів простору ознак, уздовж яких дані можуть бути краще розділені. Ці функції можуть бути будь-якою підмножиною вхідних даних або вони також можуть бути створені мережевою архітектурою.

1.5.2 Методи на основі лексики

Методи на основі лексики полягають в пошуку сентиментальної лексики, яка використовується для аналізу тексту. Словниковий підхід залежить від визначення початкових слів, а потім пошуку в словнику їхніх синонімів і антонімів. Корпусний підхід, починаються з початкового списку слів, які

виражають особисту думку, а потім знаходить інші схожі слова у великому корпусі, щоб допомогти знайти певний контекст для цих слів. Це можна зробити за допомогою статистичних або семантичних методів.

Словниковий підхід має серйозний недолік, який полягає в неможливості знайти слова, орієнтовані на галузь і контекст. Корпусний підхід допомагає вирішити проблему пошуку вставних слів, які виражають особисте ставлення, із орієнтацією на контекст. Його методи опираються на синтаксичні шаблони або шаблони, які зустрічаються разом із початковим списком вставних слів, які виражають особисту думку, щоб знайти інші схожі слова у великому корпусі. Наприклад, почати зі списку початкових прикметників, які виражають особисту думку, і шукати їх разом із набором сполучників, щоб визначити додаткові прикметників, що виражають особисту думку, та їх орієнтації. Сполучник «і», наприклад, говорить про те, що сполучені прикметники зазвичай мають однакову орієнтацію. Цю ідею називають консистенцією настроїв, яка не завжди є практичною. Існують також несприятливі сполучники, такі як «але», «однак», які позначаються як зміна думки. Щоб визначити, чи є два сполучених прикметники однакової або різної орієнтації, навчання застосовується до великого корпусу. Потім зв’язки між прикметниками утворюють граф, і на цьому графові виконується кластеризація для отримання двох наборів слів: позитивного та негативного.

Пошук шаблонів спільної появи або початкових вставних слів, що виражають особисту думку, можна здійснити за допомогою статистичних методів. Можна використовувати весь набір індексованих документів у мережі як корпус для побудови словника. Це долає проблему недоступності деяких слів, якщо використовуваний корпус недостатньо великий. Полярність слова можна визначити, вивчивши частоту появи слова у великому розміченому корпусі текстів. Якщо слово частіше зустрічається серед позитивних текстів, то його

полярність позитивна. Якщо воно частіше зустрічається серед негативних текстів, то його полярність негативна. Якщо воно має рівні частоти, то це нейтральне слово.

Подібні вставні слова, що виражають особисту думку, часто з’являються разом у корпусі. Це головне спостереження, на якому базуються сучасні методи. Тому, якщо два слова часто з’являються разом в одному контексті, вони, ймовірно, мають однакову полярність. Тому полярність невідомого слова можна визначити шляхом обчислення відносної частоти спільної появи з іншим словом. Це можна зробити за допомогою ПВІ.

Семантичні методи дають значення настрою безпосередньо та опираються на різні принципи для обчислення подібності між словами. Цей принцип надає подібні сентиментальні значення семантично близьким словам. WordNet, наприклад, надає різні типи семантичних зв’язків між словами, які використовуються для обчислення полярності настроїв. WordNet також можна використовувати для отримання списку слів, що виражають настрої, ітеративно розширюючи початковий набір синонімами та антонімами, а потім визначаючи полярність настрою для невідомого слова за відносною кількістю позитивних і негативних синонімів цього слова. Семантичний підхід використовується в багатьох програмах для побудови лексиконної моделі для опису дієслів, іменників і прикметників, які будуть використовуватися в сентиментальному аналізі. Семантичні методи також можуть бути поєднані із статистичними методами для виконання завдань сентиментального аналізу.

1.6 Аналіз прикладних застосувань сентиментального аналізу

Одним із результатів багатьох десятиліть досліджень сентиментального аналізу є створення великої кількості лексиконів та розмічених даних. Однак основною рушійною силою досліджень є їх прикладні застосування. В роботі [10] наведені наступні сфери прикладного застосування сентиментального аналізу:

- Здоров’я
- Соціальна політика
- Електронна торгівля
- Цифрові гуманітарні науки
- Інші сфери досліджень

Текстові набори даних, пов’язані зі здоров’ям, можуть створюватися медичними працівниками у формі клінічних записок або окремими особами у формі публікацій у соціальних мережах. Застосування до них сентиментального аналізу може бути корисним багатьма способами:

- Зміна стану здоров'я: практикуючий лікар може відзначити, покращився чи погіршився стан пацієнта. Це можна зробити за допомогою сентиментального аналізу. Сентименти можуть виражати реакцію на фармакологічні втручання, які використовуються для лікування;
- Критичні події: про невідкладну медичну допомогу можна повідомляти через сильні негативні сентименти;
- Досвід пацієнта: практикуючий лікар може повідомляти про досвід пацієнта, коли пацієнт описує, як вони себе почувають під час прийому.

Крім того, через широке розповсюдження соціальних мереж люди часто повідомляють про стан свого здоров’я у формі дописів у соціальних мережах,

таких як твіти. Епідемічна розвідка на основі соціальних мереж — це використання публікацій у соціальних мережах для моніторингу початку та поширення епідемій у реальному світі.

Сентиментальний аналіз займає важливе місце в просторі соціальних обговорень у поєднанні з вилученням тексту з домену соціальних мереж. Це може бути дійсно потужним інструментом для збору громадської думки. Оскільки оцифрування громадської думки через такі форуми, як Reddit і Twitter, стає все більш популярним, і все більше людей отримують доступ до Інтернету, спостерігаються зміни в стратегіях, які використовують уряди світу з урахуванням цифрових медіа. Сентиментальний аналіз щодо політичного дискурсу також важливий, оскільки сентименти/емоції, виражені щодо політичного сценарію, є полярними за своєю природою. Люди висловлюють думки на користь або проти політичного рішення або навіть рішень судів у багатьох країнах. Таким чином, широко поширена думка, а також вже прийнятно використовувати інструменти та дослідження на основі сентиментального аналізу для визначення поглядів людей щодо соціальної політики.

Відгуки клієнтів на онлайн-платформах електронної торгівлі стали звичайним джерелом відгуків про продукт як для споживачів, так і для виробників. Такі сервіси, як Yelp (www.yelp.com), Zomato (www.zomato.com) і TripAdvisor (www.tripadvisor.com), зазвичай доступні для відгуків клієнтів про ресторани та туристичні напрямки. Соціальні медіа-платформи, такі як Facebook та Instagram, також є відомими хостами сторінок і каналів, які надають відгуки про ресторани та туристичні напрямки. Ці джерела відгуків також є важливою частиною «циклу зворотного зв’язку» для платформ електронної торгівлі та виробників для покращення загальної якості продукту на основі певних оглядів на рівні аспектів.

Під час покупки товарів у Інтернеті, клієнти часто покладаються на відгуки інших клієнтів, знайдені на платформах електронної торгівлі, більше, ніж на відгуки в спеціалізованих журналах. Ці огляди часто виявляються важливими рекомендаціями щодо книг, мобільних телефонів, ноутбуків, одягу тощо, і здаються надійними, оскільки вони написані користувачами продукту, у більшості випадків. Значне збільшення впливу цифрових технологій і проникнення Інтернету в сільську місцевість в останні роки призвели до величезної кількості висловлюваних думок щодо продуктів. Зокрема, клієнти висловлюють думки щодо багатьох аспектів продуктів, послуг, блогів і коментарів. Це відрізняється від традиційних опитувань і анкет, які учасники повинні були заповнювати без будь-якої особистої мотивації, що призводило до неоптимальної зібраної інформації.

Цифрові гуманітарні науки — це область досліджень, яка лежить на перетині обчислювальних методів (таких як обробка природномовної інформації) і сфери гуманітарних наук. Цифрові гуманітарні науки в контексті текстових наборів даних можуть аналізувати ці набори даних, щоб зрозуміти еволюцію мов, еволюцію упереджень у наборах даних. Такі твори, як книги, п’єси та фільми, можуть бути засобами розваги, але також відображають ширше суспільство. Аналіз літературних творів за допомогою обчислювальних методів може бути корисним для розуміння цих літературних творів, їх порівняння та аналізу закономірностей між літературними творами того самого періоду або одного автора. Наприклад, цікавим питанням може бути розуміння основних загальних закономірностей і схожості між героями та п’єсами Вільяма Шекспіра. Подібним чином, іншим цікавим додатком був би аналіз індійських епосів «Рамаяна» та «Махабхарата» та порівняння їхніх сюжетних ліній у контексті траєкторій емоцій персонажів.

Є також багато інших сфер дослідження, де сентиментальний аналіз може бути корисним, включно з некомп’ютерними галузями. Оскільки розгорнуті

моделі, засновані на настроях, стають легшими у використанні, дослідники в різних сферах, як-от психологія, вивчення усного перекладу, переклад тощо, використовували ці моделі для побудови своїх досліджень.

1.7 Аналіз застосування сентиментального аналізу до текстів з мікроблогів

Сфера досліджень сентиментального аналізу швидко розвивається завдяки багатим і різноманітним даним, які надають програми Web 2.0: блоги, сайти оглядів, форуми, сайти мікроблогів, вікі та соціальні мережі — усі вони надають різні дані, які використовуються для сентиментального аналізу [11].

Сайти оглядів – це веб-сайти, який дозволяють користувачам публікувати відгуки, що висловлюють критичну думку про людей, компанії, продукти чи послуги. Більшість аналізу настроїв було виконано на сайтах з оглядами фільмів і товарів. Метою рецензії є оцінка конкретного об’єкта, тому це однотематична проблема. Сентиментальний аналіз на сайтах оглядів корисний як виробникам, так і потенційним споживачам товару. За відгуками виробники можуть оцінити сприйняття продукту, з’ясувати, які елементи подобаються і не подобаються рецензентам.

Термін веб-журнал або блог означає просту веб-сторінку, що складається з коротких абзаців думок, інформації, записів до особистих щоденників або посилань, які називаються дописами, розташованих у хронологічному починаючи з останнього допису, у стилі онлайн-журналу. Блогери публікують щогодини, щодня або щотижня, що робить взаємодію швидшою та оперативнішою. Різні блоги мають різні стилі викладу, зміст матеріалу та техніки написання.

Форуми чи дошки оголошень дозволяють своїм учасникам вести обговорення, публікуючи повідомлення на сайті. Форуми, як правило,

присвячені певній темі, тому використання форумів як бази даних дозволяє нам аналізувати сентименти в одній темі.

Соціальні мережі — це онлайн-сервіси або сайти, які намагаються імітувати соціальні стосунки між людьми, що знають один одного або мають спільні інтереси. Сайти соціальних мереж дозволяють користувачам ділитися ідеями, діяльністю, подіями та інтересами в межах своїх індивідуальних мереж. Дописи в соціальних мережах можуть стосуватися чого завгодно, починаючи з купленого телефону, переглянутого фільму, політичних питань або душевного стану людини. Таким чином, публікації дають нам багатший і різноманітніший ресурс думок і сентиментів.

Для нашої задачі найкраще підходять дані з мікроблогів, таких як Twitter та коментарі YouTube. Тому доречно буде детальніше дослідити роботи присвячені застосуванню сентиментального аналізу до текстів з мікроблогів.

Робота [12] присвячена застосуванню сентиментального аналізу до текстів із Twitter. Twitter — це соціальна мережа та сервіс для мікроблогів, який дозволяє користувачам публікувати повідомлення в реальному часі, що називаються твітами. Твіти – це короткі повідомлення, довжина яких обмежена 280 символами. Через характер сервісу для мікроблогів (швидкі та короткі повідомлення) люди використовують абревіатури, допускають орфографічні помилки, використовують смайлики та інші символи, які мають особливі значення. Смайли: це вирази обличчя, зображені за допомогою розділових знаків і літер; вони виражають настрій користувача. Таргет: користувачі Twitter використовують символ «@» для позначення інших користувачів у мікроблозі. Звернення до інших користувачів у такий спосіб автоматично їх сповіщає. Хештеги: користувачі зазвичай використовують хештеги для позначення тем. Це в першу чергу робиться для збільшення видимості їхніх твітів. Це ж саме можна сказати і про тексти з коментарів YouTube. Хоч вони і не мають обмеження по

кількості символів, проте зазвичай теж є короткими текстами з аналогічними особливостями.

Автори роботи представили два ресурси для попередньої обробки даних Twitter: 1) словник смайлів і 2) словник абревіатур. Вони створили словник смайлів, позначаючи 170 смайлів, перелічених у Вікіпедії, їхнім емоційним станом. Наприклад, «:)» позначено як позитивний, а «:=(» — як негативний. Кожному смайлику вони призначили мітку з наступного набору міток: надзвичайно позитивний, надзвичайно негативний, позитивний, негативний і нейтральний. Вони також склали словник абревіатур за допомогою онлайн-ресурсу. Словник містить розшифрування для 5184 абревіатур, наприклад, «lol» розшифровується як «laughing out loud».

Вони попередньо обробляли всі твіти наступним чином: a) замінили всі смайли їхньою полярністю настроїв за допомогою словника смайлів, б) замінили всі URL-адреси тегом ||U||, в) замінили таргети (наприклад, «@John”) тегом ||T||, d) замінили всі заперечення (наприклад, not, no, never, n't, could) тегом “NOT”, і e) замінили послідовність повторюваних символів трьома символами, наприклад, перетворивши cooooooooool на coool. Вони не замінювали послідовність лише двома символами, оскільки хотіли розрізнити звичайне вживання слова від наголошених.

Деякі ознаки засновані на попередній полярності слів. Автори використали Словник афектів у мові (DAL) і розширили його за допомогою WordNet. Цей словник із приблизно 8000 слів англійської мови присвоює кожному слову оцінку приємності ($\in$ R) від 1 (негативний) до 3 (позитивний). Спершу вони нормалізували оцінки, понизивши кожну оцінку за шкалою (яка дорівнює 3). Вони вважають слова з полярністю менше 0,5 негативними, вище 0,8 позитивними, решту нейтральними. Якщо слово не знайдено безпосередньо в словнику, вони отримували всі синоніми з WordNet. Потім шукали кожен із синонімів у DAL. Якщо в DAL знайдено будь-який синонім, вони присвоювали

оригінальному слову той самий бал приємності, що й його синоніму. Якщо жоден із синонімів не присутній у DAL, слово не мало жодної попередньої полярності.

Автори проводили експерименти для двох задач класифікації: 1) Негативне проти Позитивного та 2) Негативне проти Позитивного проти Нейтрального. Для обох задач використовувались 3 моделі та 2 комбінації моделей:

- Модель уніграм (базова лінія)
- Модель дерева ядер
- Модель сенти-ознак
- Дерева ядер + сенти-ознаки
- Уніграми + сенти-ознаки

Для обох задач найкраще себе показала модель уніграм + сенти-ознак.

В роботі [13] для проведення сентиментального аналізу над текстами з мікро-блогів застосовується наївний баєсів класифікатор. Наївний баєсів класифікатор часто використовується в класифікації тексту через його швидкість і простоту. Він робить припущення, що слова (або k-грами) генеруються незалежно від позиції слова. Для даного набору класів він оцінює ймовірність класу c для заданого документа d з термінами, t, як:

$$P(c|d) = P(c) \prod_{1 \le k \le n_d} P(t_k|c) \tag{1.4}$$

Потім класифікатор повертає клас із найвищою ймовірністю в даному документі. На практиці зазвичай обраховується логарифмічна ймовірність за формулою:

$$\hat{c} = arg\max_{c} \log(\hat{P}(c)) + \sum_{1 \le k \le n_d} \log(\hat{P}(t_k|c)), \quad (1.5)$$

де $\hat{P}(c)$ – попередня ймовірність появи класу c, яка визначається частотою появи цього класу в навчальному наборі; $\hat{P}(t_k|c)$ – попередня ймовірність появи терміну t_k, за умови класу c, яка визначається частотою появи цього терміну за умови даного класу в навчальному наборі; n_d – кількість термінів у документі d.

Автор перевіряв дві моделі класифікатора – з уніграмами та біграмами. Для уніграмного наївного баєсового класифікатора ймовірність появи терміна за умови даного класу визначається емпіричним підрахунком частоти появи цього терміна в текстах з тим самим класом. Однак для цього існує два методи: один — мультиноміальна модель, а інший — модель Бернуллі. У мультиноміальній моделі ймовірність визначається як:

$$\hat{P}_{multi}(t_k|c) = \frac{T_{ct_k}}{|V_c|}, \quad (1.6)$$

де T_{ct_k}- загальна кількість появ терміна в текстах з цим класом, $|V_c|$ – загальна кількість термінів для цього класу. Це контрастує з моделлю Бернуллі, де ймовірність підраховується як кількість документів даного класу, що містять вказаний термін, поділена на загальну кількістю документів для цього класу. Також автор застосовує метод χ^2, описаний вище, для зменшення кількості ознак.

Біграмна модель відходить від моделі мішка слів і розраховує ймовірність того, що документ належить до класу, обчислюючи ймовірність того, що кожне слово йде після слова перед ним за умови даного класу на основі емпіричного підрахунку пар слів у повідомленнях заданого класу.

Однак, у цьому випадку, навчальний набір є занадто розрідженим для використання лише біграм для класифікації, тому використовується модель лінійної інтерполяції, а також модель відступу. Модель лінійної інтерполяції зважує ймовірності уніграми та біграми, щоб визначити загальну ймовірність належності документа до класу:

$$P(c|d) = \alpha P_{unigram}(c|d) + (1 - \alpha)P_{bigram}(c|d) \quad (1.7)$$

Модель відступу, з іншого боку, використовує ймовірність біграми для терміну, якщо біграма зустрічалась у текстах із цим класом, інакше вона відступає до ймовірності уніграми.

Тестування моделей проводилося для трьох задач класифікації: 1) Полярне проти Нейтрального, 2) Позитивне проти Негативного та 3) Позитивне проти Негативного проти Нейтрального. Для першої задачі класифікації найкраще себе показала мультиноміальна унірамна модель, тоді як для другої і третьої – біграмна модель з лінійною інтерполяцією.

1.8 Постановка задачі

Об’єкт дослідження: застосування сентиментального аналізу до текстів із мікроблогів із використанням методів NLP.

Предмет дослідження: програмна система для сентиментального аналізу текстів із мікроблогів.

Мета: Розробка програмного забезпечення для аналізу коментарів про нові релізи кіно та серіалів, шляхом застосування сентиментального аналізу до релевантних текстів з мікроблогів.

Функціональні вимоги:

- Пошук текстів мікроблогів, що стосуються введеного користувачем запиту
- Класифікація сентиментів коротких текстів з мікро-блогів

Нефункціональні вимоги:

- Система має давати корисні дані для подальшого аналізу медіа-ринку

Представимо систему у вигляді «чорної скрині» (рис. 1.1):

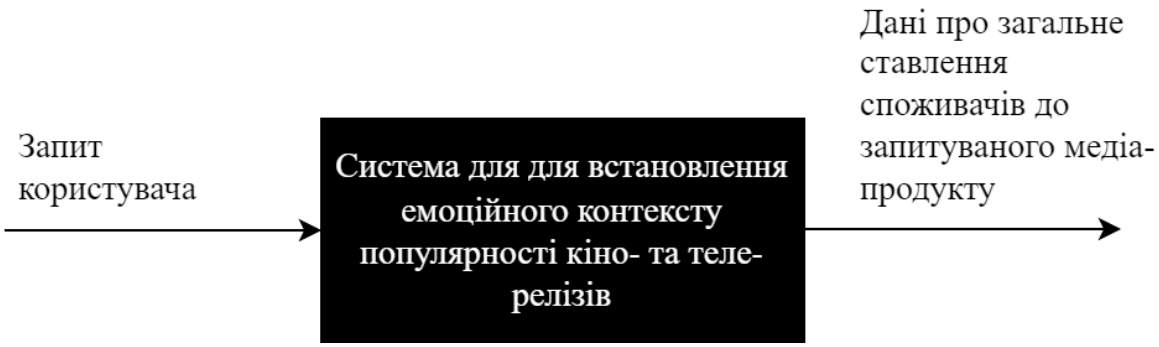

Рисунок 1.1 - Представлення системи у вигляді "чорної скрині"

1.9 Висновки з аналізу

В ході аналізу було розглянуто число наукових робіт присвячених сентиментальному аналізу. Спершу було проведено загальний аналіз сентиментального аналізу, в ході якого було охарактеризовано проблему сентиментального аналізу та визначено її особливості. Потім були розглянуті різні методи вибору ознак та безпосередньо класифікації сентиментів. Були досліджені різні сфери застосування сентиментального аналізу. Зрештою, з огляду на специфіку досліджуваних в даній роботі текстів, було проведено аналіз робіт присвячених застосуванню сентиментального аналізу до текстів з мікроблогів, таких як Twitter, та взято до уваги запропоновані авторами цих робіт способи вирішення проблеми.

РОЗДІЛ 2. ПРОЄКТНІ РІШЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ КОМЕНТАРІВ ПРО НОВІ РЕЛІЗИ КІНО І ТЕЛЕБАЧЕННЯ

2.1 Аналіз функцій системи

В цьому розділі наведено список, короткий опис та взаємозв’язок основних функцій програмної системи.

1. Функція завантаження коментарів з YouTube

Дана функція отримує рядок-запит (назва медіа-продукту) та посилання на відповідний YouTube-канал і завантажує коментарі до відео, що стосуються запиту користувача.

Вхідні дані: рядок-запит, посилання на YouTube-канал

Вихідні дані: список коментарів

2. Функція завантаження дописів з Twitter

Дана функція отримує рядок-запит (назва медіа-продукту) і завантажує дописи із Twitter, що стосуються запиту користувача.

Вхідні дані: рядок-запит

Вихідні дані: список дописів

3. Функція формування списку текстів

Дана функція формує єдиний список текстів із коментарів з YouTube та дописів з Twitter

Вхідні дані: список коментарів, список дописів

Вихідні дані: список текстів

4. Функція попередньої обробки текстів

Дана функція виконує попередню обробку текстів

Вхідні дані: список текстів

Вихідні дані: список текстів з виконаною попередньою обробкою

5. Функція вибору ознак

Дана функція виконує вибір ознак для текстів

Вхідні дані: список текстів з виконаною попередньою обробкою

Вихідні дані: список ознак для текстів

6. Функція класифікації сентиментів

Дана функція виконує класифікацію сентиментів для текстів, використовуючи його ознаки

Вхідні дані: список ознак для текстів

Вихідні дані: список маркерів класів для текстів

7. Функція консолідації вихідних даних

Дана функцію виконує консолідацію вихідних даних для користувача

Вхідні дані: список текстів, список маркерів для текстів

Вихідні дані: консолідовані вихідні дані

2.2 Узагальнена архітектура системи

Через API користувач відправляє серверу запит, в якому передає рядок-запит (назву медіа-продукту) та посилання на відповідний YouTube-канал. Модулі завантаження даних отримують дані із запиту користувача та завантажує відповідні дані, виконуючи запити до API YouTube та Twitter. Після цього модуль агрегації текстів збирає дані до купи, модуль класифікації сентиментів виконує попередню обробку, вибір ознак, та із використанням моделей класифікації афекту та класифікації полярності виконує сентиментальний аналіз та консолідує отримані дані. Зрештою, модуль API повертає користувачу відповідь на запит.

Технічна архітектура складається із веб-браузера клієнта, з якого він отримує доступ до API, робить запити і отримує результати, та сервера, на якому розгорнуте API, що виконує роботу.

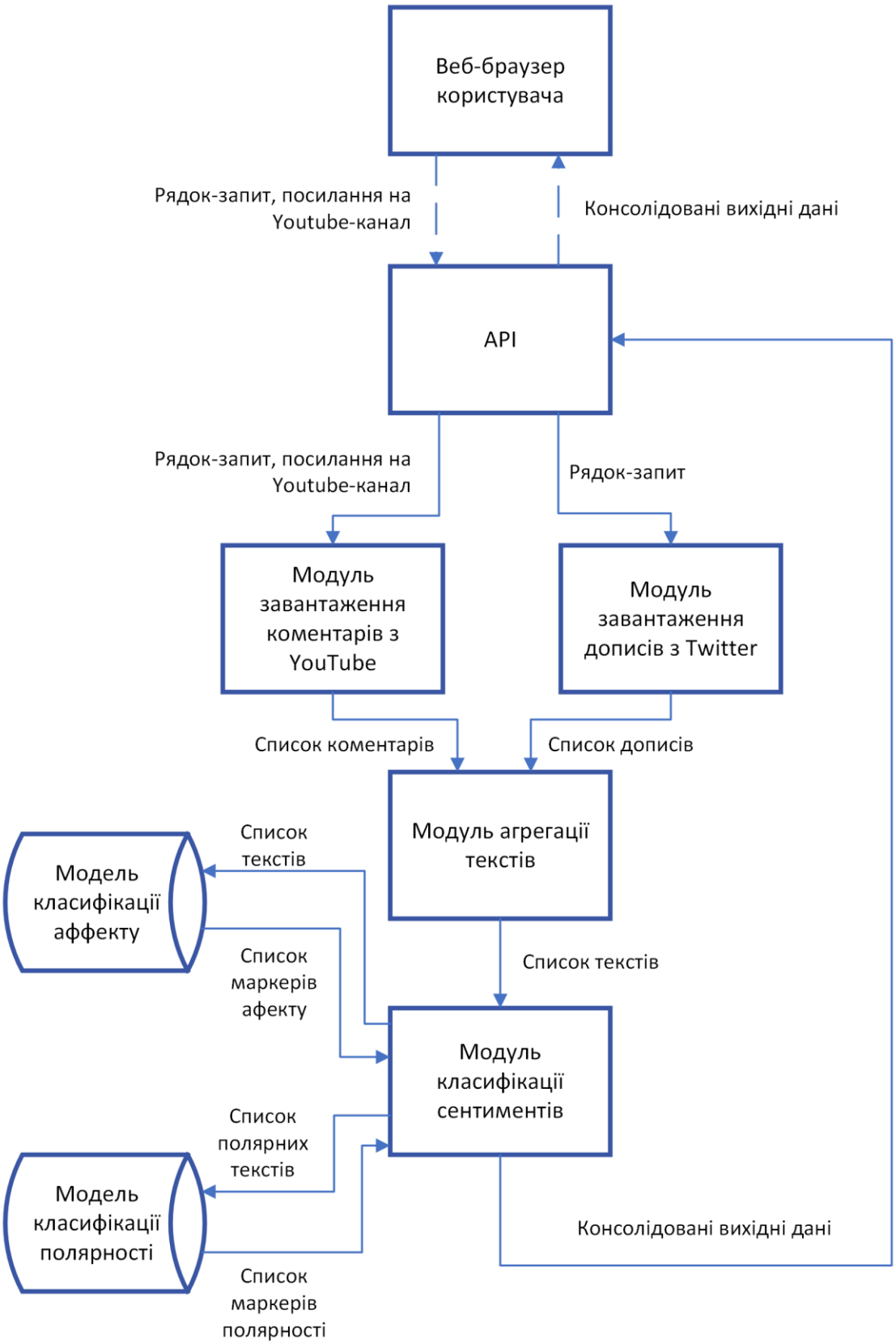

Рисунок 2.1 - Схема узагальненої архітектури системи

2.3 Дослідження можливостей використання існуючого математичного забезпечення

2.3.1 Метод вибору ознак χ^2

Міра χ^2 — це непараметричний статистичний метод аналізу, який часто використовується в експериментальній роботі, де дані складаються з частот або «підрахунків» — наприклад, кількість позитивних і негативних документів, у яких присутній певний термін — на відміну від кількісних даних, отриманих за допомогою вимірювання безперервних змінних, таких як температура, висота тощо [14]. Найпоширенішим використанням тесту є оцінка ймовірності кореляції або незалежності фактів.

Коли члени вибірки класифікуються подвійно (тобто класифікуються двома різними способами), результати можуть бути організовані в прямокутних таблицях. Така таблиця називається таблицею спряженості. Наприклад, у вибірці із 1048573 документів 248576 класифіковані як позитивні, а 799997 – як негативні. 1789 позитивних та 22254 документи містять слово “bad”, решта не містять. Ці дані можна організувати у таблиці розміром 2×2 (Таблиця 2.1):

Таблиця 2.1 Спостережені частоти появи слова “bad” серед позитивних та негативних документів

	Містить “bad”	Не містить “bad”	**Всього**
Позитивний	1789	246787	248576
Негативний	22254	777743	799997
Всього	24043	1024530	1048573

Документи в наведених даних класифіковані за двома атрибутами: полярність документу та наявність в ньому слова “bad”. Необхідно з’ясувати, чи полярність документу залежить від наявності слова “bad”.

Щоб виконати перевірку за допомогою χ^2, буде корисно перебудувати Таблицю 2.1, залишивши граничні частоти такими, як вони є, але замінивши частоти в клітинках основної частини таблиці літерами від E1 до E4. В результаті перебудови отримаємо Таблицю 2.2:

Таблиця 2.2 Таблиця спряженості із частотами заміненими на літери

	Містить “bad”	Не містить “bad”	**Всього**
Позитивний	E_1	E_2	248576
Негативний	E_3	E_4	799997
Всього	24043	1024530	1048573

Дивлячись на граничний стовпець підсумків у правій частині таблиці бачимо, що частка позитивних документів серед усіх документів має значення:

$$\frac{248576}{1048573} = 0.237 \qquad (2.1)$$

Нехай, маємо гіпотезу, що ці дві класифікації незалежні, тоді частка позитивних документів серед тих, які містять слово “bad”, має бути такою ж, як і частка позитивних документів серед тих, які не містять слово “bad”, і вони обидві мають бути рівними загальній частці позитивних документів серед усіх документів. Таким чином, маємо наступне рівняння:

$$\frac{E_1}{24043} = \frac{E_2}{1024530} = \frac{248576}{1048573} = 0.237 \quad (2.2)$$

З цього рівняння можна знайти E_1:

$$E_1 = 24043 * 0.237 = 5698.191 \quad (2.3)$$

Як тільки значення E_1 відоме, значення E_2, E_3 і E_4 можуть бути легко з'ясовані, оскільки правдиві наступні факти:

$$E_1 + E_2 = 248576 \quad (2.4)$$
$$E_1 + E_3 = 24043 \quad (2.5)$$
$$E_2 + E_4 = 1024530 \quad (2.6)$$
$$E_3 + E_4 = 799997 \quad (2.7)$$

З'ясувавши ці значення і підставивши їх у Таблицю 2.2, отримаємо Таблицю 2.3:

Таблиця 2.3 Очікувані частоти появи слова "bad" серед позитивних та негативних документів

	Містить "bad"	Не містить "bad"	**Всього**
Позитивний	5698.191	242877.809	248576
Негативний	18344.809	781652.191	799997
Всього	24043	1024530	1048573

Таблиця 2.3 містить очікувані значення, за умови, що дві наявні класифікації – незалежні. Видно, що очікувані значення помітно відрізняються від спостережених значень. Проте, необхідно чи різниця достатньо незначна, що вона могла бути спричинена випадковою помилкою вибірки, чи вона вказує на залежність полярності документу від наявності слова “bad”.

Перевірка χ^2 може дати цю відповідь. Перевірка базується на розподілі χ^2. Для того, щоб порівняти спостережені та очікувані значення частот, необхідно підрахувати значення χ^2 за наступною формулою:

$$\chi^2 = \sum_{1 \le i \le n} \frac{(O_i - E_i)^2}{E_i}, \tag{2.8}$$

де O_i – очікувані значення, E_i – спостережені значення, n – кількість комірок у таблиці спряженості. Підрахунки для прикладу зі словом “bad” наведені у Таблиці 2.4:

Таблиця 2.4 Підрахунки χ^2 для прикладу зі словом “bad”

O	E	$O - E$	$(O - E)^2$	$\frac{(O - E)^2}{E}$
1789	5698.191	-3909.191	15281774.2744	2681.8641
246787	242877.809	3909.191	15281774.2744	62.9195
22254	18344.809	3909.191	15281774.2744	833.0298
777743	781652.191	-3909.191	15281774.2744	19.5506
1048573	1048573			3597.394

Для того, щоб з’ясувати значимість отриманого значення необхідно звернутися до Таблиці 2.5:

Таблиця 2.5 Таблиця критичних значень χ^2

$\boldsymbol{df}\backslash\mathbf{P}$	99.9%	99.5%	99%	97.5%	95%	…
1	10.828	7.879	6.635	5.024	3.841	…
2	13.816	10.597	9.210	7.378	5.991	…
3	16.266	12.838	11.345	9.348	7.815	…
4	18.467	14.860	13.277	11.143	9.488	…
…	…	…	…	…	…	…

P у Таблиці 2.5 позначає рівень впевненості. df позначає ступінь незалежності атрибутів і обчислюється за формулою:

$$df = (r-1)(c-1), \tag{2.9}$$

де r – кількість рядків у таблиці спряженості, c – кількість стовпців у таблиці спряженості. В нашому випадку $df = 1$.

Таким чином, якщо значення χ^2 більше за 10.828, то із впевненістю 99.9% можна сказати, що атрибути класифікації є залежними. Оскільки отримане значення $\chi^2 = 3597.394$ є сильно більшим за 10.828, гіпотеза про незалежність атрибутів полярності документу та наявності слова “bad” – спростована.

Так, якщо ознака (наявність певного слова) проходить перевірку χ^2, то ми її обираємо для подальшого аналізу, інакше – відкидаємо як таку, що не має достатнього впливу на класифікацію.

2.3.2 Багаточленний наївний баєсовий класифікатор

Для класифікації сентиментів ми використовуємо многочленний наївний баєсовий класифікатор, описаний в роботі [15].

Текстовий документ представляється у форматі мішка слів, тобто невпорядкованого набору слів із вказаною частотою їхньої появи в документі, при цьому ігноруючи їхній порядок розташування в документі.

Наївний баєсовий класифікатор є ймовірнісним класифікатором. Це означає, що для документа d з усіх класів $c \in C$ класифікатор повертає клас $\hat{c}$, який має максимальну апостеріорну ймовірність, за умови заданого документа. Капелюх ˆ використовується для позначення «припущення правильного класу».

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) \tag{2.10}$$

Суть баєсової класифікації полягає у використанні правила Баєса для перетворення рівняння 2.1 в інші ймовірності, які мають деякі корисні властивості. Правило Баєса дає можливість розбити будь-яку умовну ймовірність P(x|y) на три інші ймовірності:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.11}$$

Таким чином,

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) = \underset{c \in C}{\operatorname{argmax}}\, \frac{P(d|c)P(c)}{P(d)} \tag{2.12}$$

Рівняння (2.3) можна спростити відкинувши знаменник $P(d)$. Це можливо, оскільки ми будемо обчислювати $\frac{P(d|c)P(c)}{P(d)}$ для кожного можливого класу. Але $P(d)$ не змінюється для кожного класу; ми шукаємо найімовірніший клас для одного і того ж документа d, який має однакову ймовірність $P(d)$. Таким чином, можна обрати клас, який максимізує простішу формулу:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) = \underset{c \in C}{\operatorname{argmax}}\, P(d|c)P(c) \tag{2.13}$$

Найімовірніший клас $\hat{c}$ знаходиться для заданого документа d вибором класу, який має найбільший добуток двох ймовірностей: попередньої ймовірності класу $P(c)$ і вірогідності документа $P(d|c)$:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(d|c)P(c) \tag{2.14}$$

Без втрати узагальнення можна представити документ d як набір ознак $f_1, f_2, \ldots, f_n$:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(f_1, f_2, \ldots, f_n|c)P(c) \tag{2.15}$$

На жаль, рівняння (2.6) все ще занадто важко обчислити безпосередньо: без деяких спрощуючих припущень оцінка ймовірності кожної можливої комбінації ознак (наприклад, кожного можливого набору слів і позицій) вимагала б величезної кількості параметрів і неймовірно великих наборів для навчання. Тому наївні баєсові класифікатори роблять два спрощуючих припущення.

Перше — це припущення про мішок слів, вказане вище: припускається, що позиція слова не має значення, і що слово «любити» однаково впливає на класифікацію незалежно від того, чи зустрічається воно як 1-е, 20-е чи останнє слово в документі. Таким чином, припускається, що ознаки $f_1, f_2, \ldots, f_n$ кодують лише ідентичність слова, а не позицію.

Друге, яке зазвичай називають наївним припущенням Баєса - це припущення умовної незалежності про те, що ймовірності $P(f_i|c)$ є незалежними за умови класу c, а, отже, можуть бути «наївно» помножені наступним чином:

$$P(f_1, f_2, \ldots, f_n|c) = P(f_1) \cdot P(f_2) \cdot \ldots \cdot P(f_n) \tag{2.16}$$

Таким чином, остаточне рівняння для класу, обраного наївним баєсовим класифікатором:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c) \prod_{f \in F} P(f|c) \tag{2.17}$$

Щоб застосувати наївний баєсовий класифікатор до тексту, потрібно розглянути слова, просто пройшовши індексом по кожному слові в документі:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c) \prod_{0 \le i \le |d|} P(w_i|c) \tag{2.18}$$

Наївні баєсові обчислення, як і обчислення для моделювання мови, виконуються в log-просторі, щоб уникнути зникнення порядку та збільшити швидкість. Таким чином, рівняння (2.9) виражається як:

$$\hat{c} = \underset{c \in C}{argmax} \log P(c) + \sum_{0 \le i \le |d|} \log P(w_i|c) \qquad (2.19)$$

Розглядаючи ознаки у log-просторі, рівняння (4.10) обчислює прогнозований клас як лінійну функцію вхідних ознак. Класифікатори, які використовують лінійну комбінацію вхідних даних для прийняття класифікаційного рішення, як наївний Баєс, а також логістична регресія, називаються лінійними класифікаторами.

Для оцінки $P(c)$ і $P(w_i|c)$ необхідно дати оцінку максимальної правдоподібності, в якості якої використовуються статистичні залежності в даних. Для попередньої ймовірності класу $P(c)$ необхідно з'ясувати, який відсоток документів у навчальному наборі належить класу c. Нехай N_c— кількість документів у навчальних даних, які належать класу c, а N_{doc} — загальна кількість документів. Тоді:

$$\hat{P}(c) = \frac{N_c}{N_{doc}} \qquad (2.20)$$

Для оцінки $P(w_i|c)$ обчислюється частота появи слова w_i серед усіх слів у всіх документах класу c. Спочатку об'єднуються всі документи класу c в один великий текст класу c. Потім обчислюється частота появи слова w_i в цьому об'єднаному документі, щоб дати оцінку максимальної правдоподібності вірогідності слова:

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}, \qquad (2.21)$$

Де $count(w, c)$ – кількість появ слова w в усіх документах класу c, V – об'єднання усіх слів з документів усіх класів.

Однак існує проблема з навчанням за оцінкою максимальної правдоподібності. Наприклад, при спробі оцінки вірогідності слова «фантастичний» для позитивного класу, навчальна вибірка не містить позитивних документів, які б містили слово «фантастичний». Можливо, слово «фантастичний» трапляється (саркастично) лише у негативному документі. У такому випадку вірогідність для цієї ознаки буде нульовою:

$$\hat{P}(\text{"фантастичний"}|\text{позитивний}) =$$

$$= \frac{count(\text{"фантастичний"}|\text{позитивний})}{\sum_{w \in V} count(w, \text{позитивний})} = 0, \qquad (2.22)$$

Але оскільки наївний Баєс наївно перемножує всі вірогідності ознак разом, нульові ймовірності для будь-якого класу призведуть до того, що ймовірність класу дорівнюватиме нулю, незалежно від інших ознак.

Найпростішим рішенням є Лапласове згладжування. Хоча згладжування за Лапласом зазвичай замінюють більш складними алгоритмами згладжування в мовному моделюванні, воно зазвичай використовується в наївній баєсовій категоризації тексту:

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|} \qquad (2.23)$$

2.3.3 Сенти-ознаки

Сенти-ознаки – це інший підхід до формалізованого представлення тексту, запропонований у роботі [12]. На відміну від «мішка слів» сенти-ознаки представлені у вигляді масиву числових значень різноманітних характеристик тексту.

Сенти-ознаки можна розділити на три категорії: ті, які є підрахунками різноманітних характеристик, і тому значення ознаки є натуральним числом $\in N$ (наприклад, кількість позитивних / негативних слів; кількість слів-заперечень; кількість слів, що належать до певної частини мови тощо); ті, значення яких є дійсним числом $\in R$ - це перш за все ознаки, які фіксують попередню оцінку полярності, отриману, наприклад, зі словника афекту в мові (DAL), але також й інші дійсні характеристики тексту (наприклад, відсоток тексту написаний великими літерами); і ті, значення яких є булевими $\in B$ (наприклад, наявність знаків оклику, слів, написаних великими літерами тощо).

Кожна з цих категорій поділяється на дві підкатегорії: полярні ознаки та неполярні ознаки. Ознака вважається полярною, якщо обчислена її попередня полярність, наприклад, за допомогою пошуку в DAL (розширеному через WordNet) або в словнику смайлів. Усі інші ознаки, які не пов’язані з жодною попередньою полярністю, належать до неполярної категорії. Кожна з полярних і неполярних функцій далі підрозділяється на дві категорії: POS та інші. POS відноситься до функцій, які збирають статистичні дані про частини мови, а Інше стосується всіх інших типів ознак.

Таблиця 2.6 Класифікація сенти-ознак

N	Полярні	POS	- Кількість позитивних / негативних слів, що належать до певної частини мови (ім., прикм., дієсл.)
		Інше	- Кількість слів-заперечень - Кількість позитивних / негативних слів - Кількість дуже позитивних / дуже негативних слів - Кількість позитивних / негативних смайлів - Кількість позитивних / негативних слів, написаних великими літерами
	Неполярні	POS	- Кількість слів, що належать до певної частини мови (ім., прикм., дієсл.)
		Інше	- Кількість слів, написаних великими літерами - Кількість сленгових слів
R	Полярні	POS	- Для кожної частини мови (ім., прикм., дієсл.) Σ попередньої полярності слів, що належать до цієї частини мови
		Інше	- Σ попередньої полярності усіх слів
	Неполярні	Інше	- Відсоток тексту, написаного великими літерами
B	Неполярні	Інше	- Наявність знаків оклику - Наявність тексту, написаного великими літерами

2.3.4 Багатошаровий перцептрон

Багатошаровий перцептрон є найвідомішим і найчастіше використовуваним типом нейронної мережі [16]. У більшості випадків сигнали передаються в мережі в одному напрямку: від входу до виходу. В мережі відсутні цикли, тобто вихід кожного нейрона не впливає на сам нейрон. Ця архітектура називається прямим поширенням.

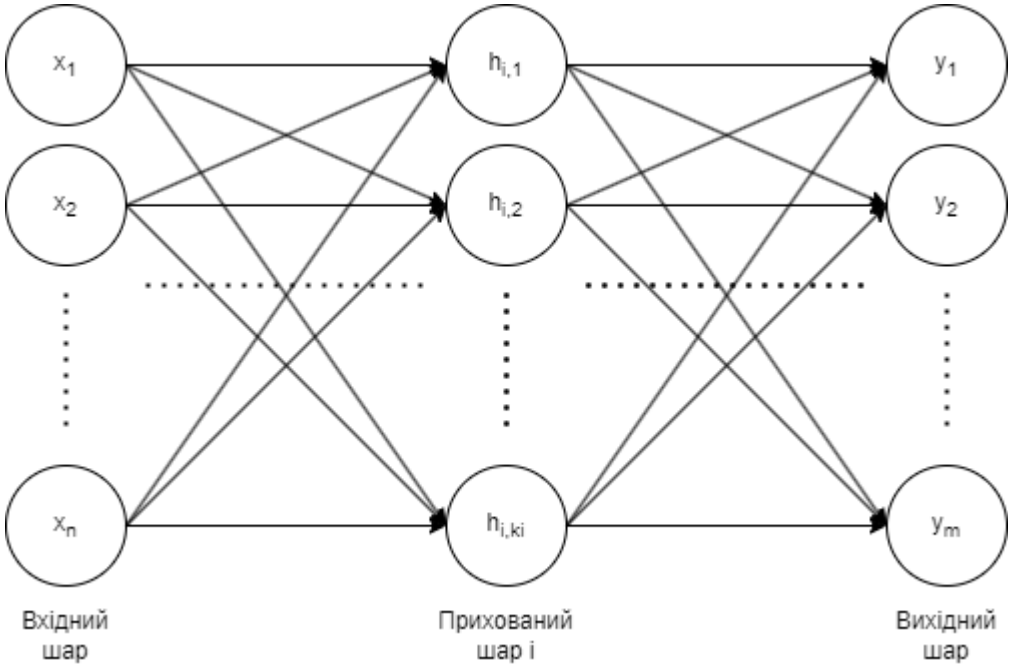

Рисунок 2.2 Схема багатошарового перцептрона прямого поширення

Вхідний шар багатошарового перцептрона матиме стільки нейронів, скільки є вхідних ознак [17]. Вихідний шар зазвичай має один нейрон для кожного класу виходів. Отже, структура багатошарових персептронів відрізняється лише кількістю прихованих шарів і кількістю нейронів у кожному з цих прихованих шарів. Приховані шари – це шари, які безпосередньо не пов'язані із зовнішнім середовищем. Перцептрон з одним лише вихідним шаром і одним вхідним генерує області прийняття рішень у вигляді півплощин. Додаючи один прихований шар, кожен нейрон діє як стандартний персептрон для виходів нейронів у передньому шарі, таким чином вихід мережі може оцінити опуклі області прийняття рішень, що є результатом перетину напівплощин, створених нейронами. У свою чергу, перцептрон із двома прихованими шарами може генерувати довільні області прийняття рішень.

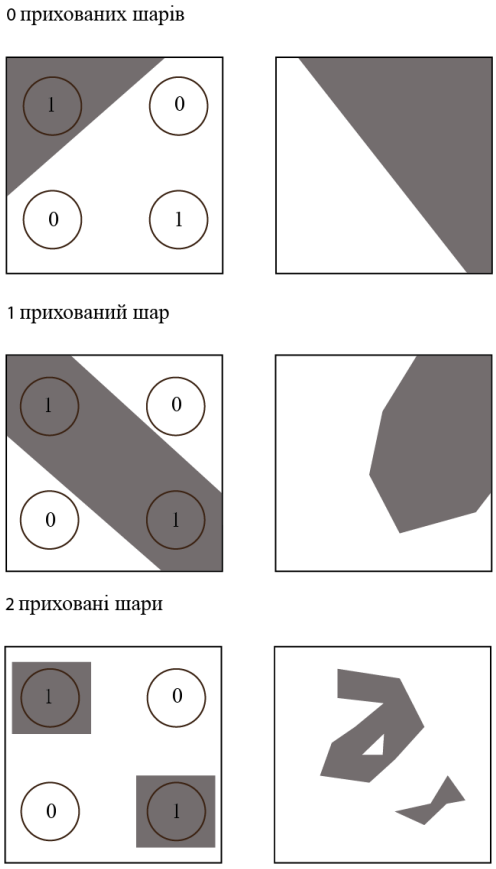

Рисунок 2.3 Форми областей прийняття рішень залежно від кількості прихованих шарів

Модель індивідуального нейрона перцептрона наведена на рисунку 2.3:

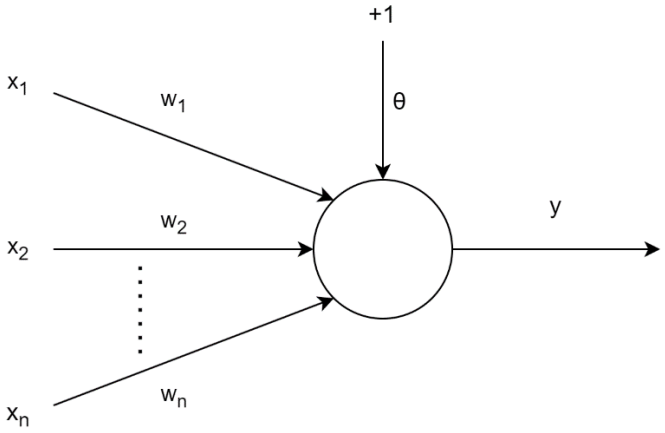

Рисунок 2.4 Модель індивідуального нейрона перцептрона

Зважені значення виходів нейронів попередніх шарів надходять на вхід нейрона. Він їх підсумовує та додає значення зміщення. На виході нейрона отримана сума трансформується функцією активації нейрона та передається до наступного шару перцептрону. Таким чином, функція активації має наступний вигляд:

$$y = f\left(\left(\sum_{i=1}^{n} x_i w_i\right) + \theta\right) \tag{2.24}$$

Зміщення або ініціалізація — це додатковий вузол, доданий до кожного шару багатошарового персептрона, вхід якого дорівнює одиниці. Таким чином, значення зміщення, помножене на вагу, що з'єднує порогове значення з наступним шаром, є константою.

Що стосується функції активації нейронів було виявлено, що багатошарові мережі не забезпечують збільшення обчислювальної потужності порівняно з мережами з одним шаром, якщо функції активації є лінійними, оскільки лінійна функція лінійних функцій також є лінійною функцією. Потужність багатошарового перцептрона походить саме від нелінійних функцій активації. Для цього можна використовувати практично будь-яку нелінійну функцію, за винятком поліноміальних функцій. Зараз найбільш часто використовуваними функціями є однополюсна (або логістична) сигмоїда (2.25) та біполярна сигмоїда (гіперболічний тангенс) (2.26)

$$\mathrm{f}(s) = \frac{1}{1 + e^{-s}} \tag{2.25}$$

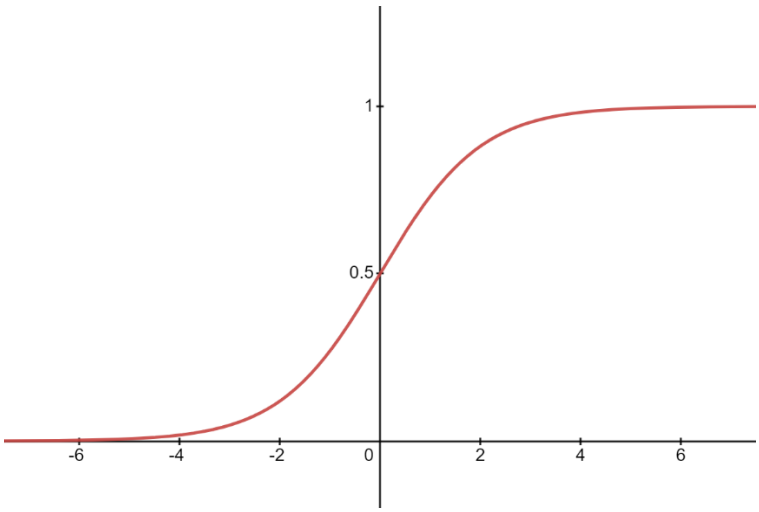

Рисунок 2.5 Графік логістичної сигмоїди

$$\mathrm{f}(s) = \frac{1 - e^{-a \cdot s}}{1 + e^{-a \cdot s}} \tag{2.26}$$

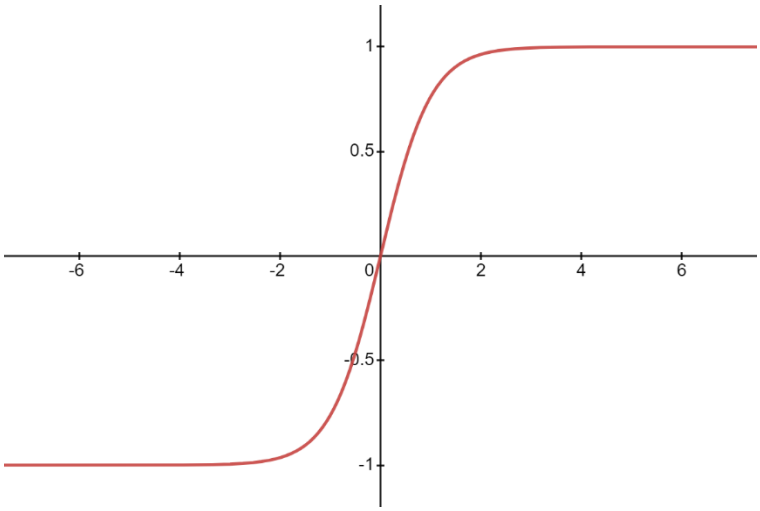

Рисунок 2.6 Графік гіперболічного тангенсу при $a = 2$

2.4 Алгоритм роботи програмної системи

Наведемо схему алгоритму роботи програмної системи (Рисунок 2.7)

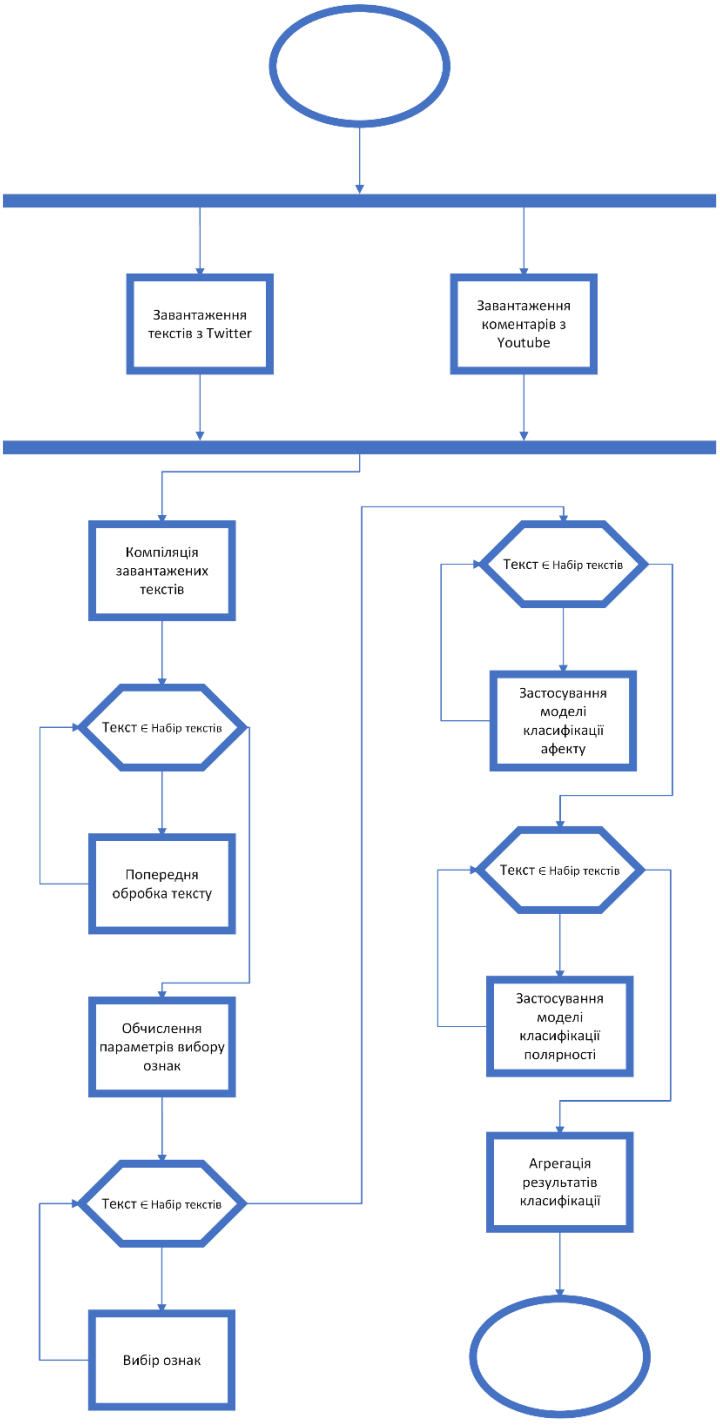

Рисунок 2.7 - Схема алгоритму роботи програмної системи

2.5 Опис моделей класифікації

У даній роботі розроблено дві моделі класифікації сентиментів – на основі наївного баєсового класифікатора та на основі сенти-ознак. Далі буде проведено їх порівняльний аналіз, та вибір кращої моделі для програмної системи. В цьому розділі наведено опис моделей.

2.5.1 Модель на основі наївного баєсового класифікатора

Спершу проводиться попередня обробка текстів. Вона включає в себе:

- Очищення текстів від непотрібних елементів: таргети (@John), посилання (http://example.com/), хештеги (#sentiment_analysis)
- Токенізація слів та смайлів
- Вилучення стоп-слів
- Лемматизація слів

В результаті попередньої обробки маємо список уніграм для кожного із текстів у навчальній вибірці.

Далі формується словник частот появи кожного слова у навчальній вибірці та відбувається вибір ознак. Спершу обираються 80% найчастіше вживаних слів. Потім до них застосовується метод χ^2 для вибору слів, які мають найменше значення невизначеності, тобто які впевненіше вказують на приналежність слова до одного із класів.

Після цього проводиться навчання уніграмного наївного баєсового класифікатора на обраних ознаках.

Потім зі списків уніграм формуються списки біграм для кожного із текстів навчальної вибірки. Для них аналогічним чином проводиться вибір ознак.

Навчання біграмного наївного баєсового класифікатора дещо відрізняється від уніграмного, оскільки визначає не ймовірність появи слова за умови певного класу, а ймовірність появи слова одразу після іншого слова, за умови певного класу.

Зрештою, формується класифікаційна модель із відступом. Тобто, спочатку відбувається спроба застосування біграмного наївного баєсового класифікатора до тексту, представленого у вигляді списку біграм, однак, якщо не вдається знайти певну біграму у словнику біграмного наївного баєсового класифікатора, то застосовується уніграмний наївний баєсовий класифікатор до уніграм, які формують дану біграму. Якщо ж не вдається знайти й уніграму, то вона ігнорується.

2.5.2 Модель на основі сенти-ознак

Спершу проводиться попередня обробка текстів, аналогічно до моделі на основі наївного баєсового класифікатора, за винятком того, що смайли не токенізуються, оскільки вони потім окремо аналізуються на етапі вибору ознак.

Ознаки в даній моделі представлені не у вигляді «мішка слів», як в моделі на основі наївного баєсового класифікатора, а у вигляді певних числових характеристик вхідного тексту. Також для вибору цих ознак використовується не лише оброблений токенізований текст, а й оригінальний необроблений.

Основні характеристику тексту містять наступні ознаки:

- Кількість слів у тексті
- Кількість словникових слів у тексту
- Кількість знаків оклику
- Відсоток тексту, написаний великими літерами
- Кількість слів-заперечень (не, ні тощо)
- Кількість таргетів (@John)
- Кількість посилань (http://example.com/)
- Кількість хештегів (#sentiment_analysis)

Емоційні характеристики – це кількість слів у тексті, які мають певне емоційне забарвлення, отримані за допомогою програмного модулю NRCLex:

- Кількість слів, що мають емоційне забарвлення Anticip (Очікування)
- Кількість слів, що мають емоційне забарвлення Trust (Довіра)
- Кількість слів, що мають емоційне забарвлення Surprise (Здивування)
- Кількість слів, що мають емоційне забарвлення Positive (Позитив)
- Кількість слів, що мають емоційне забарвлення Joy (Радість)
- Кількість слів, що мають емоційне забарвлення Anticipation (Очікування)
- Кількість слів, що мають емоційне забарвлення Fear (Страх)
- Кількість слів, що мають емоційне забарвлення Anger (Злість)
- Кількість слів, що мають емоційне забарвлення Negative (Негатив)
- Кількість слів, що мають емоційне забарвлення Sadness (Сум)
- Кількість слів, що мають емоційне забарвлення Disgust (Огида)

Характеристики частин мови – це кількість слів у тексті, що належать до певної частини мови:

- Кількість іменників
- Кількість дієслів
- Кількість прикметників
- Кількість прислівників

Характеристики смайлів – це кількість смайлів у тексті, що належать до певної категорії:

- Кількість позитивних смайлів (😀, ♡, :))
- Кількість негативних смайлів (☹, 💩, :()
- Кількість смайлів збудження (😮, 😧, :O)

Оцінки полярності – це попередні оцінки полярності тексту, отримані за допомогою програмного модулю NLTK за різними категоріями полярності:

- Оцінка позитивності
- Оцінка негативності
- Оцінка нейтральності
- Складена оцінка

Загалом маємо 30 ознак для кожного тексту, які використовуються для навчання багатошарового перцептрона для класифікації сентиментів.

2.6 Опис проєктних рішень

2.6.1 Технології, що використовуються

Дана програмна система написана мовою Python з використанням інтегрованого середовища розробки PyCharm. Модуль API створений за допомогою бібліотеки FastAPI та суміжних до неї бібліотек Gunicorn та Pydantic. Завантаження текстів з інтернету виконується за допомогою бібліотек asyncio та aiohttp. Обробка текстів виконується за допомогою біблотек re, json, NLTK, NumPy, NRCLex, SKLearn та joblib для збереження моделей.

2.6.2 Обмеження на вхідні та формалізація вихідних даних

На вхід програмна система отримує json-файл, який містить два текстових поля: title – назва медіа-продукту для якого проводиться аналіз; youtube_url – посилання на належний YouTube канал, тобто канал компанії, яка займається виробництвом відповідного медіа-продукту.

На виході програмна система видає json-файл, який містить підсумки сентиментального аналізу, а також всі знайдені програмною системою тексти та відповідні їм мітки класів (полярності).

2.6.3 Опис навчальних даних

Навчальна вибірка дописів у Twitter взята з [18]. Вона містить 1.6 мільйонів розмічених твітів, тобто кожен твіт має позначку класу (0 – негативний, 4 - позитивний). Навчальна вибірка коментарів YouTube надана компанією Ampere Analysis. Вона містить лише тексти коментарів, тому потребує ручної розмітки, аналогічної до розмітки навчальної вибірки дописів у Twitter.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

3.1 Опис структури інтерфейсу користувачів

API як таке не має інтерфейсу, оскільки це лише набір кінцевих точок (endpoints) до яких виконуються веб-запити. Проте розроблена програмна система має зручний користувацький інтерфейс, який містить перелік усіх кінцевих точок та дозволяє виконувати до них запити.

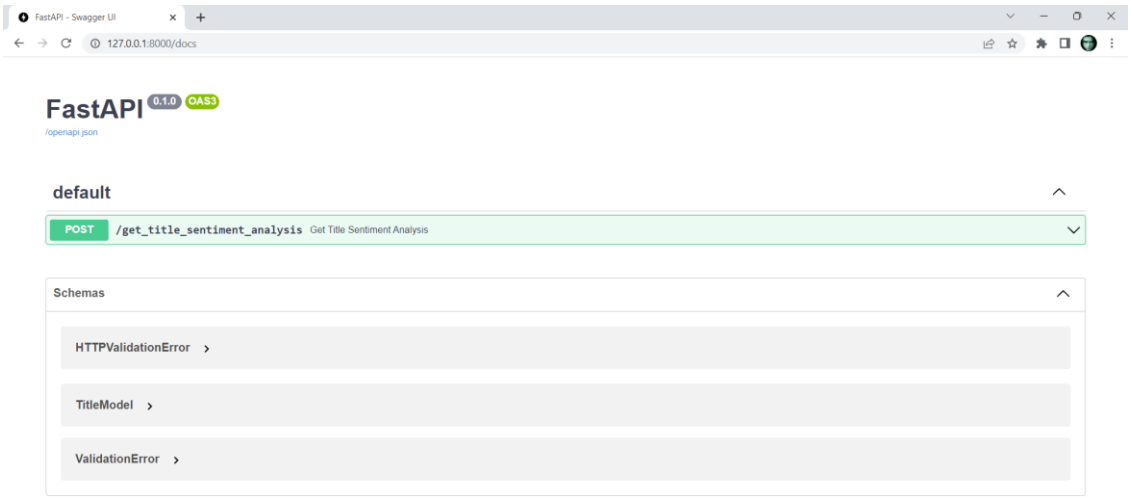

Рисунок 3.1 Сторінка інтерфейсу API з переліком кінцевих точок

У нашому випадку кінцева точка лише одна – POST запит /get_title_sentiment_analysis. За натиснення на неї розгорнеться форма для виконання запиту.

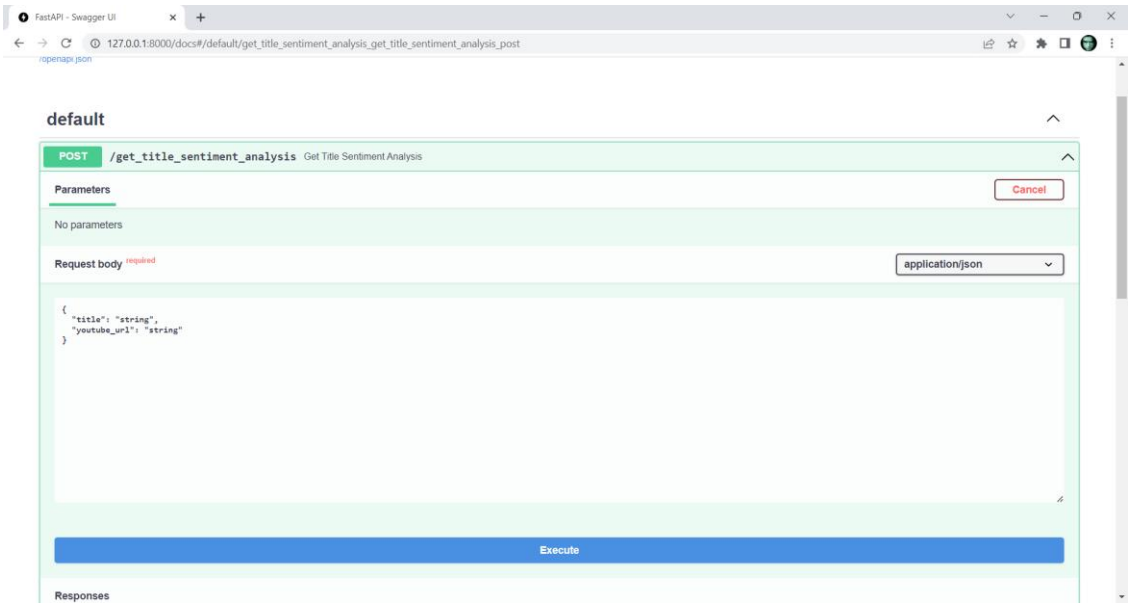

Рисунок 3.2 Форма виконання запиту до API

Після введення відповідних даних та натиснення кнопки «Execute» відбудеться виконання запиту і зрештою буде виведено повернений сервером результат.

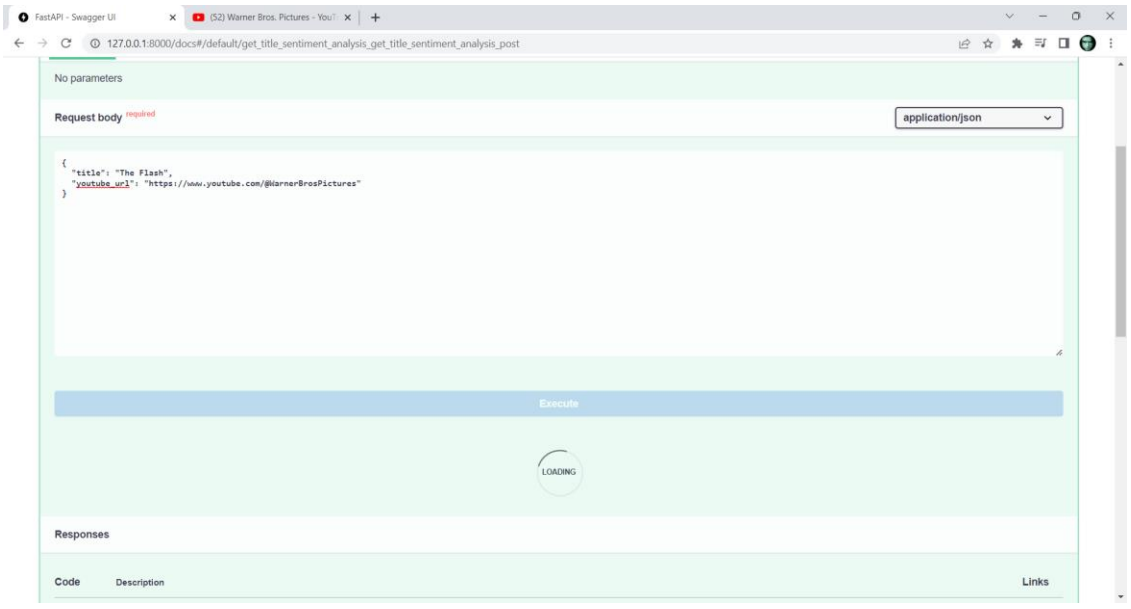

Рисунок 3.3 Очікування результату виконання запиту

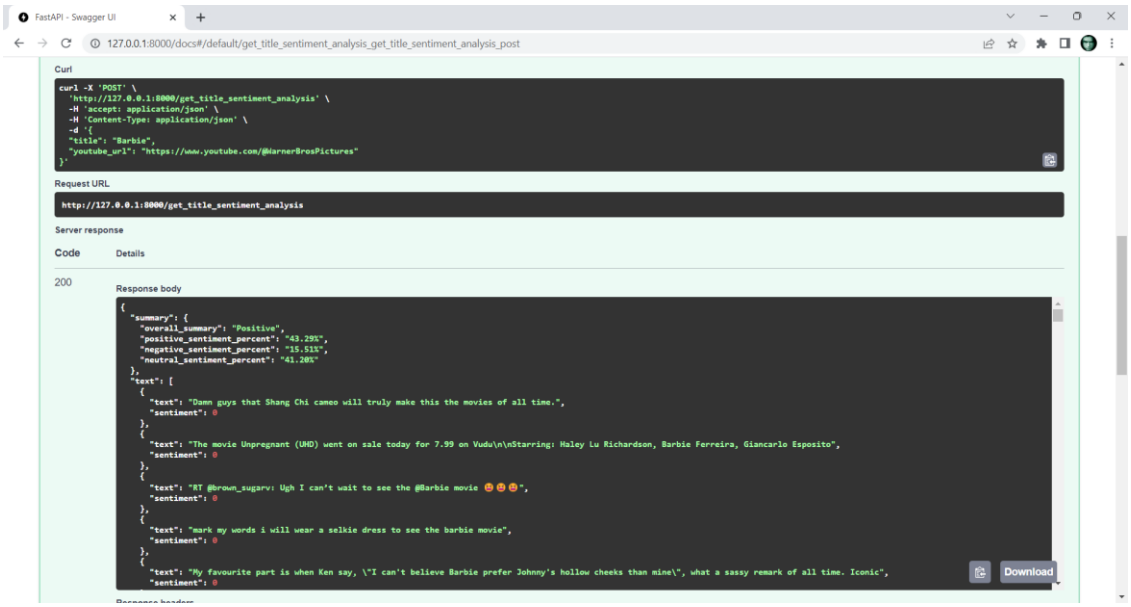

Рисунок 3.4 Результат виконання запиту

3.2 Опис модулів завантаження текстів

Модуль завантаження текстів з Twitter використовує офіційне Twitter API. Для нього було створено застосунок на платформі розробників Twitter та отримано API ключ. Модуль виконує запити до кінцевої точки https://api.twitter.com/2/tweets/search/recent?query= та переглядає результати і отримує токени для перегляду наступних сторінок твітів, знайдених за запитом, доки не досягне задовільної кількості твітів.

Модуль завантаження коментарів з YouTube виконує запити до API, яке використовує сам YouTube для завантаження даних на сторінку браузера. Таким чином, модуль спочатку завантажує певну кількість відео з відповідного YouTube каналу, обирає серед цих відео ті, назва яких відповідає запиту та потім паралельно завантажує коментарі до отриманих відео.

3.3 Опис методів порівняння результатів роботи моделей

Для порівняння результатів роботи моделей використовується F-міра та точність, як показано у роботі [19].

Для обчислення F-міри [20] підраховуються наступні значення: TP (True Positive) – кількість правильно віднесених до певного класу документів, FP (False Positive) – кількість хибно віднесених до певного класу документів, та FN (False Negative) – кількість хибно віднесених до іншого класу документів. Далі обчислюються значення влучності (P) та повноти (R) за наступними формулами:

$$P = \frac{TP}{TP + FP} \tag{3.1}$$

$$R = \frac{TP}{TP + FN} \tag{3.2}$$

Влучність представляє відношення частки документів, які система правильно відносить до певного класу, до загального числа документів, які система віднесла до цього класу. Вона винагороджує за ретельний відбір і карає за надто старанні системи, які повертають забагато результатів: щоб досягти високої точності, слід відкинути все, що може бути неправильним.

Повнота вказує, скільки з усіх документів, які мали бути віднесені до певного класу, дійсно були віднесені системою до цього класу. Ця метрика винагороджує вичерпність: щоб досягти високої повноти, краще включати об’єкти, щодо яких система не впевнена.

Ці міри контрастують одна з одною: якщо віднести до певного класу всі документи, то матимемо високу повноту та низьку влучність, якщо віднести лише один документ – високу влучність та низьку повноту. Тому загальноприйнятою практикою є балансувати їх за допомогою F-міри – середнє гармонійне зважене влучності та повноти:

$$F_\beta = (1 + \beta^2)\frac{P * R}{\beta^2 P + R} \tag{3.3}$$

У цьому рівнянні коефіцієнт β визначає баланс між влучністю та повнотою, при цьому високі значення β сприяють повноті. Зазвичай використовується значення $\beta = 1$:

$$F_1 = 2\frac{P * R}{P + R} \tag{3.4}$$

Значення P, R та F_1 обчислюються для кожного класу окремо.

Точність (A) позначає загальну частку правильно класифікованих документів та обчислюється спільно для всіх класів:

$$A = \frac{T}{N} \tag{3.5}$$

Де T – кількість правильно класифікованих документів, N – загальна кількість документів.

3.4 Порівняння результатів роботи моделей класифікації

Розроблені моделі використовуються для вирішення двох завдань – класифікація афекту (відділення нейтральних текстів від полярних) та класифікація полярності (відділення негативних текстів від позитивних). Результати роботи моделей наведені у Таблиці 3.1 і Таблиці 3.2.

Таблиця 3.1 Порівняння результатів роботи моделей для вирішення задачі класифікації афекту

	Полярні тексти			Нейтральні тексти			A
	P	R	F_1	P	R	F_1	
Уніграмна модель на основі наївного баєсового класифікатора	0.7715	0.8061	0.7884	0.7970	0.7612	0.7787	0.7838
Модель із відступом на основі наївного баєсового класифікатора	0.7700	0.7460	0.7578	0.7537	0.7771	0.7652	0.7617
Модель на основі сентиознак	0.9450	0.8184	0.8772	0.8398	0.9524	0.8926	0.8854

Таблиця 3.2 Порівняння результатів роботи моделей для вирішення задачі класифікації полярності

	Позитивні тексти			Негативні тексти			A
	P	R	F_1	P	R	F_1	
Уніграмна модель на основі наївного баєсового класифікатора	0.8197	0.7556	0.7864	0.7733	0.8338	0.8024	0.7948
Модель із відступом на основі наївного баєсового класифікатора	0.7948	0.6944	0.7412	0.7287	0.8207	0.7720	0.7412
Модель на основі сентиознак	0.7854	0.8262	0.8053	0.8167	0.7743	0.7949	0.8003

Як бачимо, для вирішення задачі класифікації афекту, модель на основі сенти-ознак показує себе значно краще. Тому вона і використовуватиметься у програмній системі.

Для вирішення задачі класифікації полярності, модель із відступом на основі наївного баєсового класифікатора сильно програє двом іншим. Проте уніграмна модель на основі наївного баєсового класифікатора та модель на основі сенти-ознак мають практично однакові показники. Оскільки для вирішення задачі класифікації афекту була обрана модель на основі сенти-ознак, то, заради спрощення обробки вхідних текстів, її ж і оберемо для вирішення задачі класифікації полярності у розроблюваній програмній системі.

3.5 Інструктивні матеріали користувача

Кінцевою точкою аналізу настроїв глядачів про певний продукт є POST кінцева точка /get_title_sentiment_analysis. Тобто, якщо API запущене на локальній машині користувача, то для виконання аналізу необхідно відправити POST-запит за посиланням http://localhost:8000/get_title_sentiment_analysis.

Тіло запиту має наступну структуру:

```
{
  "title": "string",
  "youtube_url": "string"
}
```

Де поле “title” позначає назву продукту, а “youtube_url” – посилання на YouTube-канал дистриб’ютора продукту.

Відповідь на запит має наступну структуру:

```
{
  "summary": {
    "overall_summary": "string",
    "positive_sentiment_percent": "string",
    "negative_sentiment_percent": "string",
    "neutral_sentiment_percent": "string"
  },
  "texts": [
    {
      "text": "string",
      "sentiment": int
    },
  ]
}
```

Де поле “summary” містить словник, в якому міститься консолідований результат аналізу: “overall_summary” – загальний сентимент стосовно продукту, тобто клас, до якого належить найбільша кількість текстів (Positive, Negative, Neutral), “positive_sentiment_percent” – відсоток текстів із позитивним сентиментом, “negative_sentiment_percent” – відсоток текстів із негативним сентиментом, “neutral_sentiment_percent” – відсоток текстів із нейтральним сентиментом. Поле “texts” містить список словників, які містять дані про знайдені коментарі: “text” – сам коментар, “sentiment” – клас, до якого належить даний коментар (1 – позитивний, 0 – нейтральний, -1 – негативний)

3.6 Тестування програмної системи

Далі наведено приклади роботи розробленої програмної системи для декількох фільмів.

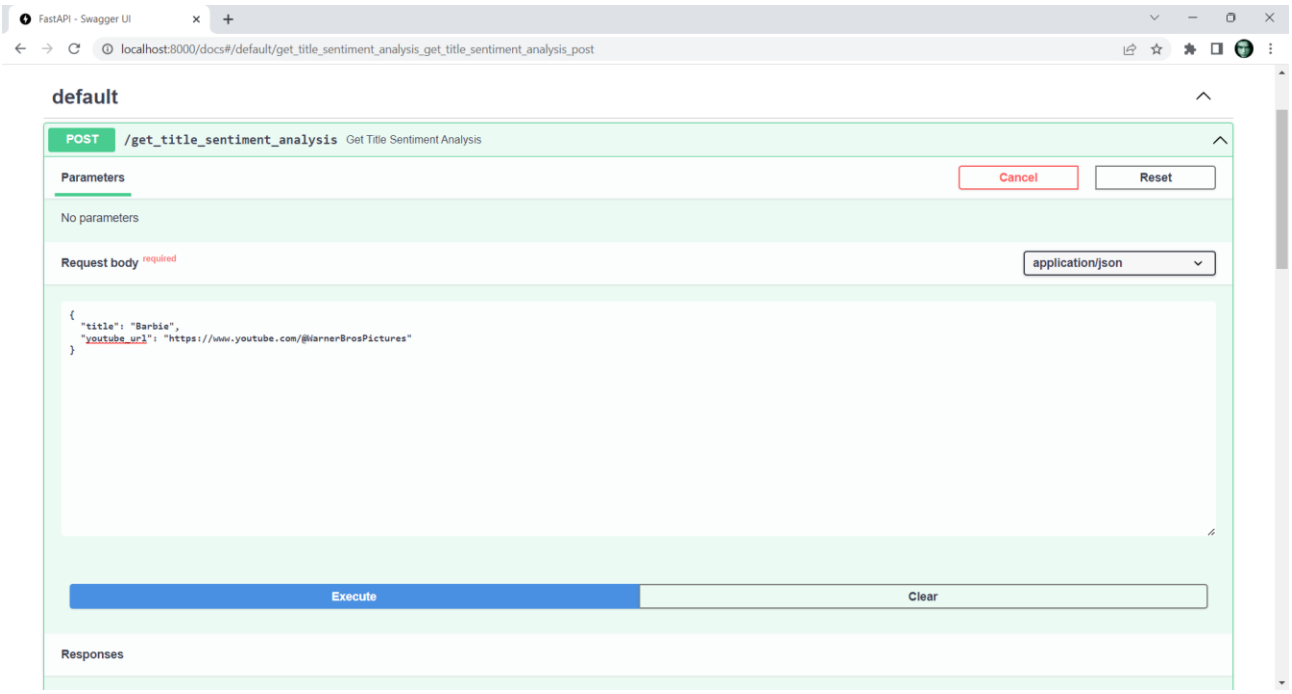

Рисунок 3.5 Запит для фільму Barbie (2023)

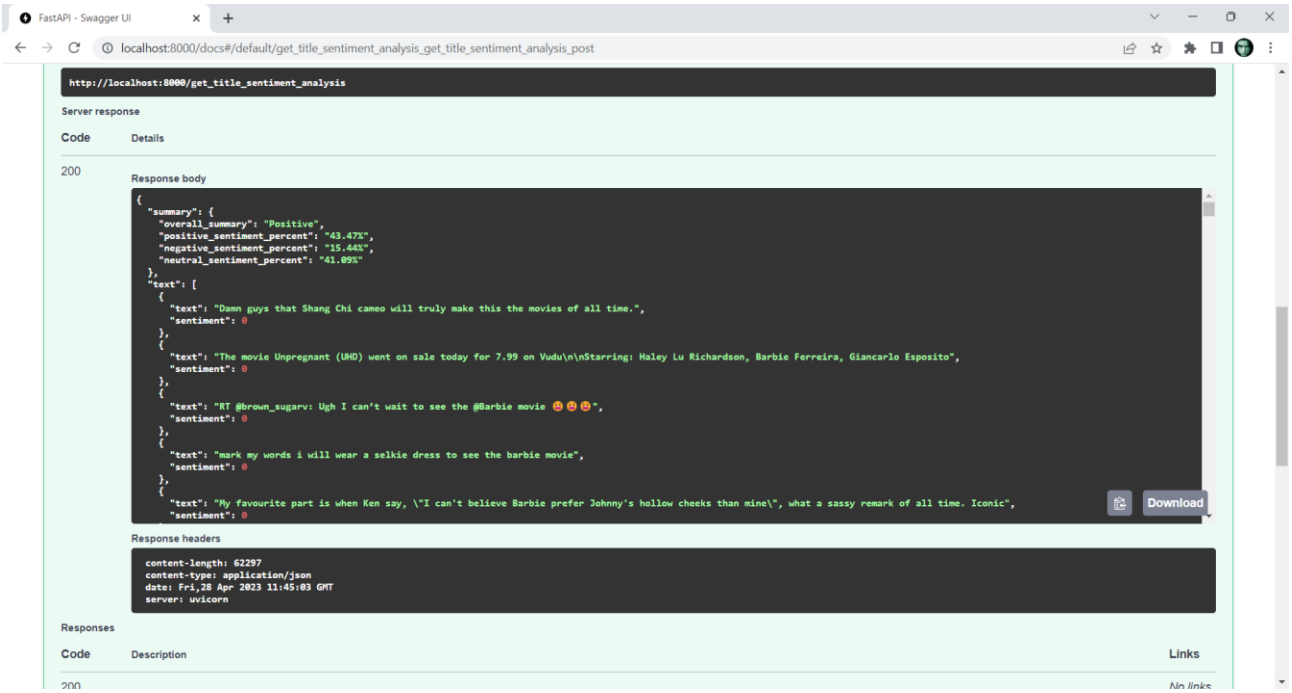

Рисунок 3.6 Результат роботи системи для фільму Barbie (2023)

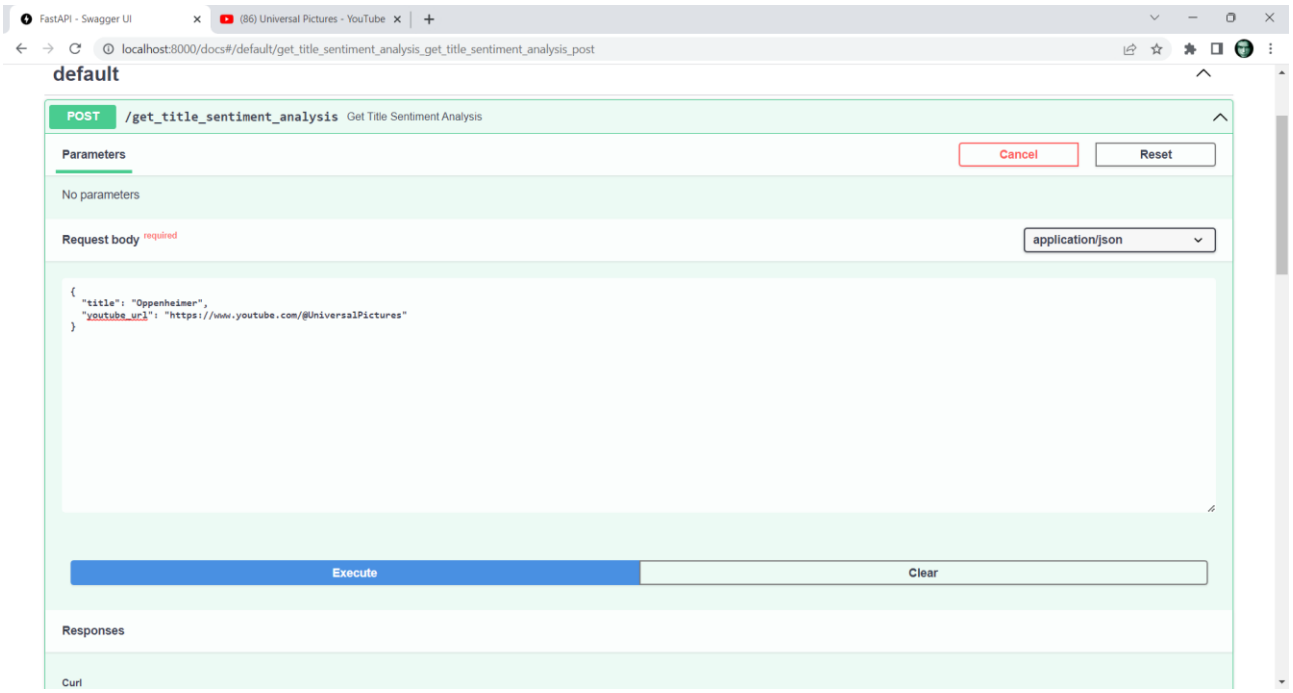

Рисунок 3.7 Запит для фільму Oppenheimer (2023)

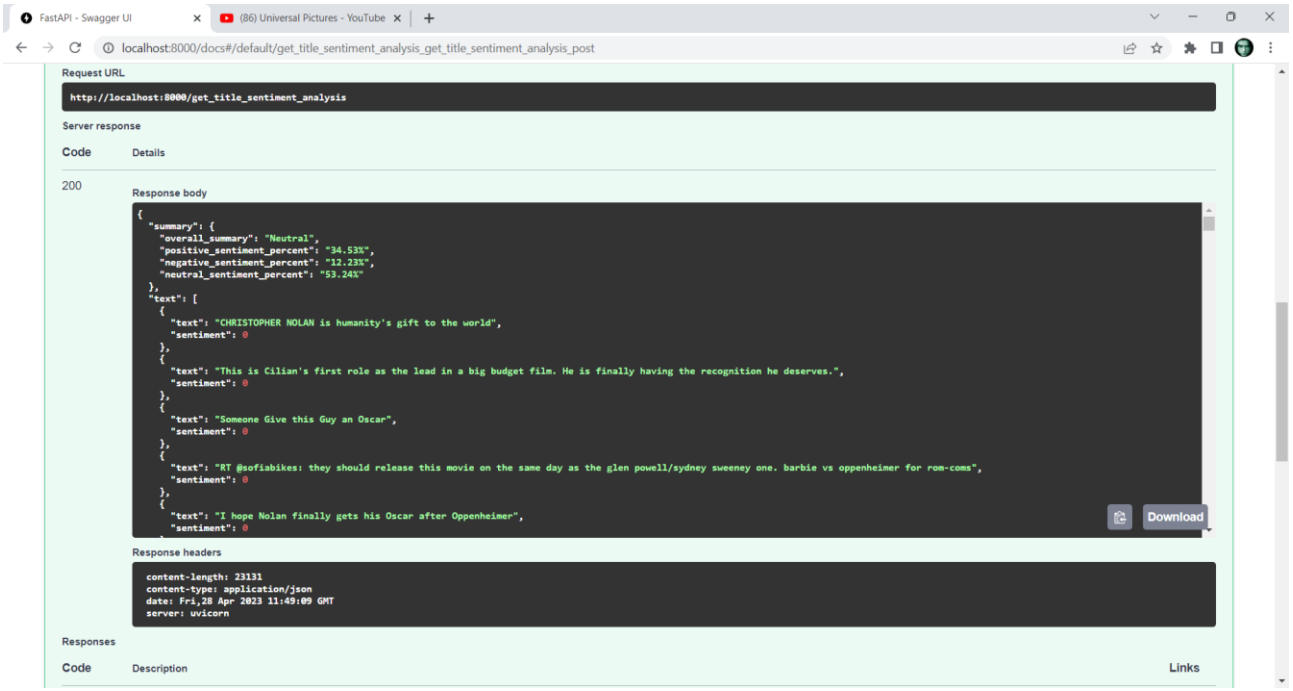

Рисунок 3.8 Результат роботи для фільму Oppenheimer (2023)

3.7 Висновки з тестування

За результатами порівняльного аналізу двох моделей ми визначили, що краще із задачею класифікації сентиментів справляється модель на основі сенти-ознак та має точність 80%. Саме її ми і використали у розроблюваній програмній системі.

Тестування програмної системи показало, що вона добре виконує завдання завантаження коментарів з інтернету, що стосуються бажаного запиту, проводить сентиментальний аналіз цих текстів і повертає користувачу зведену статистику проведеного аналізу. Далі ця статистика може бути використана у подальшому аналізі даних.

ВИСНОВКИ

В ході даної роботи було проаналізовано проблему сентиментального аналізу коментарів з інтернету. Було проаналізовано різні методи вибору ознак та класифікації сентиментів. Було прийняте рішення про розробку програмної системи для аналізу коментарів про нові релізи кіно та серіалів, з якою користувачі взаємодіятимуть за допомогою API. Було розроблено архітектуру програмної системи, яка включає в себе: модуль API, який забезпечує взаємодію користувача із програмною системою; модулі завантаження текстів з Twitter і YouTube, які виконують запити до API відповідних соціальних мереж для завантаження необхідних текстів; модуль агрегації текстів, який агрегує отримані з Twitter та YouTube текстів; модуль класифікації сентиментів, який проводить попередню обробку текстів, вибір ознак, використовує моделі класифікації афекту та класифікації полярності для проведення сентиментального аналізу текстів та консолідує отримані результати у загальну статистику. Було розроблено, навчено та проведено порівняльний аналіз трьох моделей для класифікації сентиментів – уніграмної моделі на основі наївного баєсового класифікатора, моделі із відступом на основі наївного баєсового класифікатора та моделі на основі сенти-ознак, і було обрану модель на основі сенти-ознак для застосування у програмній системі як ту, яка найкраще виконує поставлену задачу. Було реалізовану запропоновану програмну систему та проведено її тестування, яке показало, що розроблена нами програмна система справляється із поставленою задачею та повертає корисні статистичні дані щодо нових релізів кіно та серіалів, які в можуть бути використані в подальшому аналізі даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Taboada, M. Sentiment Analysis: An Overview from Linguistics. *Annual Review of Linguistics. 2016.* 10.1146/annurev-linguistics-011415-040518.

2. Quirk, R., Greenbaum, S., Leech, G., Svartvik, J. A Comprehensive Grammar of the English Language. London: Longman, 1985.

3. Wiebe, J., Wilson, T., Bruce, R., Bell, M., Martin, M. Learning Subjective Language. *Computational Linguistics*. 2004. с. 277-308. 10.1162/0891201041850885.

4. Pang, B., Lee, L., Vaithyanathan, S. Thumbs up? Sentiment Classification using Machine Learning Techniques. *Proceedings of Conference on Empirical Methods in Natural Language Processing,* 2002, с. 79–86.

5. Mejova, Y. Sentiment Analysis: An Overview. Comprehensive Exam Paper : 16.11.2009 / Computer Science Department, University of Iowa

6. Medhaat, W., Hassan, A., Korashy, H. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal, Volume 5, Issue 4*. 2014, с. 1093-1113.

7. Garg, A., Roth, D. Understanding Probabilistic Classifiers. ред. L. de Raedt, P. Flach. *Machine Learning: ECML 2001 - 12th European Conference on Machine Learning,* 2001, с. 179-191.

8. Yuan, G.-X., Ho, C.-H., Lin, C.-J. Recent Advances of Large-Scale Linear Classification. *Proceedings of the IEEE, Volume 100, Issue 9.* 2012. c. 2584-2603.

9. Karoly, A. I., Fuller, R., Galambos, P. Unsupervised Clustering for Deep Learning: A tutorial survey. *Acta Polytechnica Hungarica*. 2018. c. 29-53. 10.12700/APH.15.8.2018.8.2.

10. Kanojia, D., Joshi, A. Applications and Challenges of SA in Real-life Scenarios. 2023. 10.48550/arXiv.2301.09912.

11. Kumar, A., Sebastian, T. M. Sentiment Analysis: A Perspective on its Past, Present and Future. *International Journal of Intelligent Systems and Applications.* 2012. 10.5815/ijisa.2012.10.01.

12. Agarwal, A., Xie, B., Vovsha, I., Rambow, O., Passonneau, R. Sentiment Analysis of Twitter Data. *Proceedings of the Workshop on Language in Social Media.* 2011. c. 30–38

13. Prasad, S. Micro-blogging Sentiment Analysis Using Bayesian Classification Methods. 2010.

14. Zibran, M. F. CHI-Squared Test of Independence. *Department of Computer Science, University of Calgary, Alberta, Canada*. 2007.

15. Jurafsky, D., Martin, J. F. Naive Bayes and Sentiment Classification. *Speech and Language Processing, 3rd Edition (Draft)*. 2019. c. 56-74.

16. Popescu, M.-C., Perescu-Popescu, L., Balas, V. E., Mastorakis, N. Multilayer Perceptron and Neural Networks. *WSEAS Transactions on Circuits and Systems. 2009.*

17. Belue, L. M. Multilayer perceptrons for classification. Master's thesis : 01.03.1992 / AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

18. Sentiment140 dataset with 1.6 million tweets. Kaggle : веб-сайт. URL: https://www.kaggle.com/datasets/kazanova/sentiment140

19. Na, J.-C., Kyiang, W. Y. M. Sentiment Analysis of User-Generated Content on Drug Review Websites. *JOURNAL OF INFORMATION SCIENCE THEORY AND PRACTICE, Volume 3, Issue 1*. 2015. с. 6-23.

20. Derczynski, L. Complementarity, F-score, and NLP Evaluation. *Proceedings of the Tenth International Conference on Language Resources and Evaluation*. 2016. с. 261–266.

ДОДАТКИ

ДОДАТОК А

Лістинг коду

bayes_model.py

```
import random
import re
import json
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer
from additional_data.emojis import positive_emojis, negative_emojis,
exclamation_emojis


chi_threshold = 3.841


def get_texts():
    with open('affect_texts.json') as tweets_json:
        all_texts = json.load(tweets_json)

    all_positive_texts = [text['text'] for text in all_texts if text['class'] ==
1]
    all_negative_texts = [text['text'] for text in all_texts if text['class'] ==
0]

    return all_positive_texts, all_negative_texts


def process_text(tokenizer, stemmer, stop_words, text: str):
    clean_text = re.sub('@[^ ]+|http[^ ]+|RT', '', text)

    tokenized_text = tokenizer.tokenize(clean_text.lower())

    processed_text = []

    for token in tokenized_text:

        if token not in stop_words:
            stemmed_token = stemmer.lemmatize(token)

            processed_text.append(stemmed_token)

    return tuple(processed_text)


def get_freqs(positive_texts, negative_texts):
    freqs = {}

    for tweet in positive_texts:
        for word in tweet:
            pair = (word, 1)
```

```
            if pair in freqs:
                freqs[pair] += 1
            else:
                freqs[pair] = 1

    for tweet in negative_texts:
        for word in tweet:
            pair = (word, 0)

            if pair in freqs:
                freqs[pair] += 1
            else:
                freqs[pair] = 1

    return freqs


def lookup(freqs, word, label):
    pair = (word, label)

    return freqs.get(pair, 0)


def train_naive_bayes_bigrams(freqs, train_x, train_y):
    logliklihood = {}

    bigrams_vocab = set(pair[0] for pair in freqs)
    vocab = {}

    for pair in freqs:
        if pair[0][0] not in vocab:
            vocab[pair[0][0]] = {
                0: {},
                1: {}
            }

        vocab[pair[0][0]][pair[1]][pair] = freqs[pair]

    D = len(train_y)

    D_pos = len(list(filter(lambda x: x == 1, train_y)))
    D_neg = len(list(filter(lambda x: x == 0, train_y)))

    logprior = {
        0: np.log(D_neg / D),
        1: np.log(D_pos / D)
    }

    for bigram in bigrams_vocab:
        big_pos = lookup(freqs, bigram, 1)
        big_neg = lookup(freqs, bigram, 0)

        w_N_pos = sum(vocab[bigram[0]][1].values())
        w_N_neg = sum(vocab[bigram[0]][0].values())

        w_V_pos = len(vocab[bigram[0]][1].keys())
        w_V_neg = len(vocab[bigram[0]][0].keys())
```

```
        prob_pos = np.log((big_pos + 1) / (w_N_pos + w_V_pos)) if w_V_pos > 0
else None
        prob_neg = np.log((big_neg + 1) / (w_N_neg + w_V_neg)) if w_V_neg > 0
else None

        logliklihood[bigram] = {
            0: prob_neg,
            1: prob_pos
        }

    return logliklihood, logprior



def train_naive_bayes(freqs, train_x, train_y):
    logliklihood = {}

    vocab = set(pair[0] for pair in freqs)

    N_pos = N_neg = V_pos = V_neg = 0

    for pair in freqs:
        if pair[1] == 0:
            N_neg += freqs[pair]
            V_neg += 1
        else:
            N_pos += freqs[pair]
            V_pos += 1

    D = len(train_y)

    D_pos = len(list(filter(lambda x: x == 1, train_y)))
    D_neg = len(list(filter(lambda x: x == 0, train_y)))

    logprior = {
        0: np.log(D_neg / D),
        1: np.log(D_pos / D)
    }

    for word in vocab:
        pos_freq = lookup(freqs, word, 1)
        neg_freq = lookup(freqs, word, 0)

        prob_pos = np.log((pos_freq + 1) / (N_pos + V_pos))
        prob_neg = np.log((neg_freq + 1) / (N_neg + V_neg))

        logliklihood[word] = {
            0: prob_neg,
            1: prob_pos
        }

    return logliklihood, logprior


def naive_bayes_predict(text, logprior, logliklihood):
    p = logprior.copy()
    classes = p.keys()
```

```
    for word in text:
        if word in logliklihood:
            for cl in classes:
                if logliklihood[word][cl]:
                    p[cl] += logliklihood[word][cl]

    return max(p, key=p.get)


def naive_bayes_predict_recall(text, logprior, logliklihood_bigrams,
logliklihood_unigrams):
    bigrams_text = get_bigrams(text)
    p = logprior.copy()
    classes = p.keys()

    for bigram in bigrams_text:
        if bigram in logliklihood_bigrams:
            for cl in classes:
                if logliklihood_bigrams[bigram][cl]:
                    p[cl] += logliklihood_bigrams[bigram][cl]
        else:
            for index, word in enumerate(bigram):
                if word in logliklihood_unigrams:
                    for cl in classes:
                        p[cl] += logliklihood_unigrams[word][cl]

    return max(p, key=p.get)


def naive_bayes_test(test_x, test_y, logprior, logliklihood):
    y_hat = []

    for text in test_x:
        y_hat.append(naive_bayes_predict(text, logprior, logliklihood))

    error = np.mean(np.absolute(test_y - y_hat))
    accuracy = 1 - error

    return accuracy, y_hat


def naive_bayes_test_recall(test_x, test_y, logprior, logliklihood_bigrams,
logliklihood_unigrams):
    y_hat = []

    for text in test_x:
        y_hat.append(naive_bayes_predict_recall(text, logprior,
logliklihood_bigrams, logliklihood_unigrams))

    error = np.mean(np.absolute(test_y - y_hat))
    accuracy = 1 - error

    return accuracy, y_hat


def feature_selection(positive_texts, negative_texts):
    corpus = positive_texts + negative_texts
    vocab_list = set([word for text in corpus for word in text])
    vocab = {
```

```
        word: {
            0: [],
            1: []
        }
        for word in vocab_list
    }

    number_of_docs = len(corpus)
    pos_docs = len(positive_texts)
    neg_docs = len(negative_texts)

    for text in positive_texts:
        unique_text = set(text)

        for word in unique_text:
            vocab[word][1].append(text)

    for text in negative_texts:
        unique_text = set(text)

        for word in unique_text:
            vocab[word][0].append(text)

    selected_words = []

    for word in vocab:
        print('word', word)
        pos_docs_with_word = len(vocab[word][1])
        neg_docs_with_word = len(vocab[word][0])
        pos_docs_no_word = pos_docs - pos_docs_with_word
        neg_docs_no_word = neg_docs - neg_docs_with_word

        docs_with_word = pos_docs_with_word + neg_docs_with_word

        pos_fraq = pos_docs / number_of_docs

        exp_pos_docs_with_word = docs_with_word * pos_fraq
        exp_neg_docs_with_word = docs_with_word - exp_pos_docs_with_word
        exp_pos_docs_no_word = pos_docs - exp_pos_docs_with_word
        exp_neg_docs_no_word = neg_docs - exp_neg_docs_with_word

        obs_values = [pos_docs_with_word, neg_docs_with_word, pos_docs_no_word,
neg_docs_no_word]
        exp_values = [exp_pos_docs_with_word, exp_neg_docs_with_word,
exp_pos_docs_no_word, exp_neg_docs_no_word]

        chi_sqr = sum([np.square(obs_values[i] - exp_values[i]) / exp_values[i]
for i in range(4)])

        if chi_sqr > chi_threshold:
            selected_words.append(word)

    selected_words = set(selected_words)

    return selected_words


def select_words(words, positive_texts, negative_texts):
    selected_positive_texts = [[word for word in text if word in words] for text
```

```
in positive_texts]
    selected_negative_texts = [[word for word in text if word in words] for text
in negative_texts]

    return selected_positive_texts, selected_negative_texts


def select_most_frequent_words(freqs: dict):
    all_words_count = sum(freqs.values()) * 0.8
    sorted_freqs = {k: v for k, v in sorted(freqs.items(), key=lambda item:
item[1], reverse=True)}

    most_frequent_words = []
    count = 0

    for word in sorted_freqs:
        most_frequent_words.append(word[0])
        count += sorted_freqs[word]

        if count >= all_words_count:
            break

    return set(most_frequent_words)


def turn_to_set(_list):
    list_of_tuples = [tuple(item) for item in _list]

    return set(list_of_tuples)

def turn_to_list(_set):
    list_of_lists = [list(item) for item in _set]

    return list_of_lists


def get_bigrams(text):
    bigrams = [(text[i], text[i + 1]) for i in range(len(text) - 1)]

    return bigrams

def get_p(tp, fp):
    return tp / (tp + fp)

def get_r(tp, fn):
    return tp / (tp + fn)

def get_f1(p, r):
    return 2 * (p * r) / (p + r)

def get_evaluation(y_hat, num_pos):
    sb_pos = list(y_hat[:num_pos])
    sb_neg = list(y_hat[num_pos:])

    print()
    print('Pos')
    tp = sb_pos.count(1)
    tn = sb_neg.count(0)
    fp = len(sb_neg) - tn
```

```
    fn = len(sb_pos) - tp

    p = get_p(tp, fp)
    r = get_r(tp, fn)

    f1 = get_f1(p, r)

    print('p', p)
    print('r', r)
    print('f1', f1)

    print()
    print('Neg')
    tp = sb_neg.count(0)
    tn = sb_pos.count(1)
    fp = len(sb_pos) - tn
    fn = len(sb_neg) - tp

    p = get_p(tp, fp)
    r = get_r(tp, fn)

    f1 = get_f1(p, r)

    print('p', p)
    print('r', r)
    print('f1', f1)


def main():
    all_positive_texts, all_negative_texts = get_texts()

    tokenizer = RegexpTokenizer(f'(\w+|{"|".join(positive_emojis +
negative_emojis + exclamation_emojis)})')
    stemmer = WordNetLemmatizer()
    stop_words = stopwords.words('english')

    stop_words.extend(['u', 'r', 'ur'])

    all_processed_positive_texts = [process_text(tokenizer, stemmer, stop_words,
text) for text in all_positive_texts]
    all_processed_negative_texts = [process_text(tokenizer, stemmer, stop_words,
text) for text in all_negative_texts]

    freqs = get_freqs(all_processed_positive_texts,
all_processed_negative_texts)
    most_frequent_words = select_most_frequent_words(freqs)

    concentrated_positive_texts, concentrated_negative_texts =
select_words(most_frequent_words, all_processed_positive_texts,
all_processed_negative_texts)

    selected_features = feature_selection(concentrated_positive_texts,
concentrated_negative_texts)

    selected_positive_texts, selected_negative_texts =
select_words(selected_features, concentrated_positive_texts,
concentrated_negative_texts)

    freqs = get_freqs(selected_positive_texts, selected_negative_texts)
```

```
    train_pos_indexes = random.choices(range(len(selected_positive_texts)),
k=int(len(selected_positive_texts) * 0.75))
    train_neg_indexes = random.choices(range(len(selected_negative_texts)),
k=int(len(selected_negative_texts) * 0.75))

    test_pos_indexes = list(set(range(len(selected_positive_texts))) -
set(train_pos_indexes))
    test_neg_indexes = list(set(range(len(selected_negative_texts))) -
set(train_neg_indexes))

    train_pos = [selected_positive_texts[i] for i in train_pos_indexes]
    train_neg = [selected_negative_texts[i] for i in train_neg_indexes]

    test_pos = [all_processed_positive_texts[i] for i in test_pos_indexes]
    test_neg = [all_processed_negative_texts[i] for i in test_neg_indexes]

    train_x = train_pos + train_neg
    test_x = test_pos + test_neg

    train_y = np.append(np.ones((len(train_pos))), np.zeros((len(train_neg))))
    test_y = np.append(np.ones((len(test_pos))), np.zeros((len(test_neg))))

    logliklihood, logprior = train_naive_bayes(freqs, train_x, train_y)

    # bigrams

    bigrams_pos = [get_bigrams(text) for text in all_processed_positive_texts]
    bigrams_neg = [get_bigrams(text) for text in all_processed_negative_texts]

    bigrams_freqs = get_freqs(bigrams_pos, bigrams_neg)
    most_frequent_bigrams = select_most_frequent_words(bigrams_freqs)

    concentrated_positive_bigrams, concentrated_negative_bigrams =
select_words(most_frequent_bigrams, bigrams_pos, bigrams_neg)

    selected_bigrams = feature_selection(concentrated_positive_bigrams,
concentrated_negative_bigrams)

    selected_positive_bigrams, selected_negative_bigrams =
select_words(selected_bigrams, concentrated_positive_bigrams,
concentrated_negative_bigrams)

    bigrams_freqs = get_freqs(selected_positive_bigrams,
selected_negative_bigrams)

    bigrams_train_pos_indexes =
random.choices(range(len(selected_positive_bigrams)),
k=int(len(selected_positive_bigrams) * 0.75))
    bigrams_train_neg_indexes =
random.choices(range(len(selected_negative_bigrams)),
k=int(len(selected_negative_bigrams) * 0.75))

    bigrams_test_pos_indexes = list(set(range(len(selected_positive_bigrams))) -
set(train_pos_indexes))
    bigrams_test_neg_indexes = list(set(range(len(selected_negative_bigrams))) -
set(train_neg_indexes))

    bigrams_train_pos = [selected_positive_bigrams[i] for i in
```

```
bigrams_train_pos_indexes]
    bigrams_train_neg = [selected_negative_bigrams[i] for i in
bigrams_train_neg_indexes]

    bigrams_test_pos = [bigrams_pos[i] for i in bigrams_test_pos_indexes]
    bigrams_test_neg = [bigrams_neg[i] for i in bigrams_test_neg_indexes]

    bigrams_train_x = bigrams_train_pos + bigrams_train_neg
    bigrams_test_x = bigrams_test_pos + bigrams_test_neg

    bigrams_train_y = np.append(np.ones((len(bigrams_train_pos))),
np.zeros((len(bigrams_train_neg))))
    bigrams_test_y = np.append(np.ones((len(bigrams_test_pos))),
np.zeros((len(bigrams_test_neg))))

    bigrams_logliklihood, bigrams_logprior =
train_naive_bayes_bigrams(bigrams_freqs, bigrams_train_x, bigrams_train_y)

    accuracy, unigrams_y_hat = naive_bayes_test(test_x, test_y, logprior,
logliklihood)

    recall_accuracy, recall_accuracy_y_hat = naive_bayes_test_recall(test_x,
test_y, logprior, bigrams_logliklihood, logliklihood)

    print('unigrams')
    get_evaluation(unigrams_y_hat, int(len(test_y) / 2))

    print('recall')
    get_evaluation(recall_accuracy_y_hat, int(len(test_y) / 2))

    return




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

senti-features_model.py

```
import json
import re
import nltk
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer
from nrclex import NRCLex
from sklearn.neural_network import MLPClassifier
from additional_data.emojis import positive_emojis, negative_emojis,
exclamation_emojis
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from joblib import dump, load

sentiments = SentimentIntensityAnalyzer()
```

```
def get_texts():
    with open('affect_texts.json') as tweets_json:
        all_texts = json.load(tweets_json)

    all_positive_texts = [text['text'] for text in all_texts if text['class'] ==
1]
    all_negative_texts = [text['text'] for text in all_texts if text['class'] ==
0]

    return all_positive_texts, all_negative_texts


def process_text(words, tokenizer, stemmer, stop_words, text: str):
    clean_text = re.sub('@[^ ]+|http[^ ]+|RT', '', text)

    tokenized_text = tokenizer.tokenize(clean_text.lower())

    processed_text = []

    for token in tokenized_text:

        if token not in stop_words:
            stemmed_token = stemmer.lemmatize(token)

            processed_text.append(stemmed_token)

    return (processed_text, text)

emotions = ['anticip', 'trust', 'surprise', 'positive', 'joy', 'anticipation',
'fear', 'anger', 'negative', 'sadness', 'disgust']


def get_affect(text: str):
    text_object = NRCLex(text)
    freqs = text_object.raw_emotion_scores

    return [np.square(freqs.get(emotion, 0)) for emotion in emotions]


def get_pos_affects(text: list):
    features = []
    pos_tagged = nltk.pos_tag(text)
    pos_words = {
        'NN': [],
        'VB': [],
        'JJ': [],
        'RB': []
    }

    for word in pos_tagged:
        pos = word[1][:2]

        if pos in pos_words:
            pos_words[pos].append(word[0])

    features.extend([len(pos_words[pos]) for pos in pos_words])

    return features
```

```
def get_text_basic_statistics(tokenized_text, raw_text):
    dictionary_words = len(tokenized_text)
    words = len(raw_text.split(' '))
    exclamation = raw_text.count('!')
    capitalized = sum([1 for letter in raw_text if letter.isupper()]) /
len(raw_text)
    negative_words = len(re.findall(" +(no|not|non) +|(n'*t) +| +(non-)",
raw_text))

    targets = len(re.findall('@[^ ]+', raw_text))
    urls = len(re.findall('http[^ ]+', raw_text))
    hashtags = len(re.findall('#[^ ]+', raw_text))

    features = [dictionary_words, words, exclamation, capitalized,
negative_words, targets, urls, hashtags]

    return features


def get_emojis(text):
    pos_emojis = re.findall('|'.join(positive_emojis), text)
    neg_emojis = re.findall('|'.join(negative_emojis), text)
    ex_emojis = re.findall('|'.join(exclamation_emojis), text)

    return [len(pos_emojis), len(neg_emojis), len(ex_emojis)]

def get_polarity_scores(text):
    scores = sentiments.polarity_scores(text)

    return [scores['pos'], scores['neg'], scores['neu'], scores['compound']]

count = 0

def get_features(text: tuple):
    global count
    count += 1
    print('text', count, text)
    tokenized_text = text[0]
    raw_text = text[1]
    features = []

    features.extend(get_text_basic_statistics(tokenized_text, raw_text))
    features.extend(get_affect(' '.join(tokenized_text)))
    features.extend(get_pos_affects(tokenized_text))
    features.extend(get_emojis(raw_text))
    features.extend(get_polarity_scores(' '.join(tokenized_text)))

    return features


def model_test(model, test_x, test_y):
    y_hat = model.predict(test_x)

    error = np.mean(np.absolute(test_y - y_hat))
    accuracy = 1 - error

    return accuracy, y_hat
```

```
def get_p(tp, fp):
    return tp / (tp + fp)

def get_r(tp, fn):
    return tp / (tp + fn)

def get_f1(p, r):
    return 2 * (p * r) / (p + r)

def get_evaluation(y_hat, num_pos):
    sb_pos = list(y_hat[:num_pos])
    sb_neg = list(y_hat[num_pos:])

    print()
    print('Pos')
    tp = sb_pos.count(1)
    tn = sb_neg.count(0)
    fp = len(sb_neg) - tn
    fn = len(sb_pos) - tp

    p = get_p(tp, fp)
    r = get_r(tp, fn)

    f1 = get_f1(p, r)

    print('p', p)
    print('r', r)
    print('f1', f1)

    print()
    print('Neg')
    tp = sb_neg.count(0)
    tn = sb_pos.count(1)
    fp = len(sb_pos) - tn
    fn = len(sb_neg) - tp

    p = get_p(tp, fp)
    r = get_r(tp, fn)

    f1 = get_f1(p, r)

    print('p', p)
    print('r', r)
    print('f1', f1)


def main():
    raw_positive_texts, raw_negative_texts = get_texts()

    tokenizer = RegexpTokenizer(f'\w+')
    stemmer = WordNetLemmatizer()
    stop_words = stopwords.words('english')
    words = set(nltk.corpus.words.words())

    stop_words.extend(['u', 'r', 'ur'])

    tokenized_positive_texts = [process_text(words, tokenizer, stemmer,
stop_words, text) for text in raw_positive_texts]
```

```
    tokenized_negative_texts = [process_text(words, tokenizer, stemmer,
stop_words, text) for text in raw_negative_texts]

    positive_texts_features = [get_features(text) for text in
tokenized_positive_texts]
    negative_texts_features = [get_features(text) for text in
tokenized_negative_texts]

    train_pos = positive_texts_features[:int(len(positive_texts_features) *
0.75)]
    train_neg = negative_texts_features[:int(len(negative_texts_features) *
0.75)]
    test_pos = positive_texts_features[int(len(positive_texts_features) *
0.75):]
    test_neg = negative_texts_features[int(len(negative_texts_features) *
0.75):]

    train_x = train_pos + train_neg
    test_x = test_pos + test_neg

    train_y = np.append(np.ones((len(train_pos))), np.zeros((len(train_neg))))
    test_y = np.append(np.ones((len(test_pos))), np.zeros((len(test_neg))))

    clf = MLPClassifier(solver='adam', alpha=1e-5, activation='tanh',
hidden_layer_sizes=(30, 50), random_state=1)

    clf.fit(train_x, train_y)

    accuracy, y_hat = model_test(clf, test_x, test_y)
    print(accuracy)

    dump(clf, 'classification_models/affect_classification_model.joblib')

    get_evaluation(y_hat, int(len(test_y) / 2))

    return

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

twitter_scraper.py

```
import requests
import json


class TweeterScraper():
    def __init__(self):
        self.headers = {
            'Authorization': f'Bearer ###’
        }

    def get_tweets(self, title, number_of_tweets):
        query = f'{title} Movie'
        url =
f'https://api.twitter.com/2/tweets/search/recent?query={query}&max_results=100'
        next_token = None
```

```
        tweets = []

        while len(tweets) < number_of_tweets:
            tweeter_url = url + f'&next_token={next_token}' if next_token else
url

            response = requests.get(tweeter_url, headers=self.headers)
            json_tweets = json.loads(response.text)

            tweets.extend([tweet['text'] for tweet in json_tweets['data']])
            next_token = json_tweets['meta'].get('next_token')

            if not next_token:
                break

        return tweets


def get_tweeter_scraper():
    return TweeterScraper()
```

youtube_scraper.py

```
import asyncio

import aiohttp
import re
import json


class YoutubeScraper():
    def __init__(self):
        self.headers = {
            'accept-language': 'en-US,en;q=0.9',
            'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36'
        }
        self.api_context = None
        self.key_words = {
            'videos': 'browse',
            'comments': 'next',
        }
        self.base_url =
'https://www.youtube.com/youtubei/v1/{keyword}?key={api_key}&prettyPrint=false'
        self.videos_url = None
        self.comments_url = None

        self.structure = ['onResponseReceivedActions', 0,
'appendContinuationItemsAction', 'continuationItems']
        self.comments_header_structure = ['onResponseReceivedEndpoints', 1,
'reloadContinuationItemsCommand', 'continuationItems']
        self.comments_structure = ['onResponseReceivedEndpoints', 0,
'appendContinuationItemsAction', 'continuationItems']

        self.video_structure = ['richItemRenderer', 'content', ['videoRenderer',
'reelItemRenderer']]
```

```
        self.comment_structure = ['commentThreadRenderer', 'comment',
'commentRenderer', 'contentText', 'runs', 0, 'text']

        self.video_base_url = 'https://www.youtube.com/watch?v={video_id}'

    async def get_title_comments(self, query, channel_url):
        async with aiohttp.ClientSession() as session:
            await self.initialize_repository(session, channel_url)

            url = f'{channel_url}/videos'
            videos = await self.get_videos(session, url)
            selected_videos = [video['videoId'] for video in videos if
query.lower() in video['title']['runs'][0]['text'].lower()]

            tasks = (self.get_comments(session, video_id) for video_id in
selected_videos)

            comments_for_videos = await asyncio.gather(*tasks)
            comments = [comment for video_comments in comments_for_videos for
comment in video_comments]

            return comments

    async def initialize_repository(self, session, channel_url):
        response = await session.get(channel_url, headers=self.headers)

        data_match = re.search('window\.ytplayer={};\n*ytcfg\.set\((.*?)\);',
await response.text())

        if data_match:
            data_str = data_match.group(1)
        else:
            raise AttributeError

        data = json.loads(data_str)

        self.videos_url = self.base_url.format(keyword=self.key_words['videos'],
api_key=data['INNERTUBE_API_KEY'])
        self.comments_url =
self.base_url.format(keyword=self.key_words['comments'],
api_key=data['INNERTUBE_API_KEY'])

        self.api_context = data['INNERTUBE_CONTEXT']

    async def get_videos(self, session, url):
        response = await session.get(url, headers=self.headers)
        videos, continuation = await self.get_initial_continuation(await
response.text())

        while continuation and len(videos) < 1000:
            new_videos, continuation = await self.get_items(session,
self.videos_url, continuation, self.structure, self.video_structure)
            videos.extend(new_videos)

        return videos

    async def get_comments(self, session, videoId):
        comments = []
```

```
        url = self.video_base_url.format(video_id=videoId)
        response = await session.get(url, headers=self.headers)

        additional_video_data_str = re.search('ytInitialData *=
*(\{.*?});</script>',
                                              await response.text()).group(1)
        additional_video_data = json.loads(additional_video_data_str)
        additional_video_info =
additional_video_data['contents']['twoColumnWatchNextResults']['results']['resul
ts'][
            'contents'] \
            if 'contents' in additional_video_data and 'results' in
additional_video_data['contents'][
            'twoColumnWatchNextResults'] else None

        if additional_video_info:
            comments_tab = self.get_tab(additional_video_info,
'itemSectionRenderer', targetId='comments-section')

            if comments_tab:
                header_continuation =
comments_tab['contents'][0]['continuationItemRenderer']['continuationEndpoint'][
                    'continuationCommand']['token']

                comments, continuation = await self.get_items(session,
self.comments_url, header_continuation, self.comments_header_structure,
self.comment_structure)

                while continuation and len(comments) < 100:
                    new_comments, continuation = await self.get_items(session,
self.comments_url, continuation, self.comments_structure,
self.comment_structure)
                    comments.extend(new_comments)

        return comments


    async def get_items(self,
                        session,
                        url,
                        continuation: str,
                        structure,
                        item_structure: list = None):
        data = {
            'context': self.api_context,
            'continuation': continuation
        }

        response = await session.post(url, json=data)
        response_data = json.loads(await response.text())

        raw_items = self.get_item_with_structure(response_data, structure)

        continuation = raw_items.pop(-
1)['continuationItemRenderer']['continuationEndpoint']['continuationCommand'][
            'token'] \
            if len(raw_items) > 0 and 'continuationItemRenderer' in raw_items[-
1] else None
        items = [self.get_item_with_structure(item, item_structure) for item in
```

```
                raw_items] if item_structure else raw_items

        return items, continuation

    async def get_initial_continuation(self, text):
        initial_data_str = re.search('ytInitialData *= *(.*?);</script>',
text).group(1)
        initial_data = json.loads(initial_data_str)

        videos_tab =
self.get_tab(initial_data['contents']['twoColumnBrowseResultsRenderer']['tabs'],
'tabRenderer',
                                    title='Videos')
        if videos_tab:
            if videos_tab.get('content', {}).get('richGridRenderer',
{}).get("contents"):
                raw_items =
videos_tab['content']['richGridRenderer']['contents']

                continuation = \
                raw_items.pop(-
1)['continuationItemRenderer']['continuationEndpoint']['continuationCommand']['t
oken'] \
                    if 'continuationItemRenderer' in raw_items[-1] else None

                items = [self.get_item_with_structure(item,
self.video_structure) for item in raw_items]

                return items, continuation

    @classmethod
    def get_item_by_element(cls, items, element):
        if isinstance(items, dict):
            return items.get(element, {})
        else:
            if len(items) > 0:
                return items[element]

    @classmethod
    def get_item_with_structure(cls, item, structure):
        for element in structure:
            if isinstance(element, list):
                for subelement in element:
                    if subelement in item:
                        item = cls.get_item_by_element(item, subelement)
                        break
            else:
                item = cls.get_item_by_element(item, element)

        return item

    @classmethod
    def get_tab(cls, tabs, key, **conditions):
        tabs = [tab[key] for tab in tabs if key in tab]

        for tab in tabs:
            all_conditions_met = True

            for key in conditions:
```

```
                if key not in tab or tab[key] != conditions[key]:
                    all_conditions_met = False

            if all_conditions_met:
                return tab

def get_youtube_scraper():
    return YoutubeScraper()
```

classifier.py

```
from joblib import load
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer
import re
import nltk
from nrclex import NRCLex
from additional_data.emojis import positive_emojis, negative_emojis,
exclamation_emojis
from nltk.sentiment.vader import SentimentIntensityAnalyzer


class Classifier():
    def __init__(self):
        self.affect_classifier =
load('../classification_models/affect_classification_model.joblib')
        self.polarity_classifier =
load('../classification_models/polarity_classification_model.joblib')

        self.tokenizer = RegexpTokenizer(f'\w+')
        self.stemmer = WordNetLemmatizer()
        self.stop_words = stopwords.words('english')

        self.sentiments = SentimentIntensityAnalyzer()

        self.emotions = ['anticip', 'trust', 'surprise', 'positive', 'joy',
'anticipation', 'fear', 'anger', 'negative', 'sadness', 'disgust']

    def classify_texts(self, texts: list):
        processed_texts = [self.process_text(text) for text in texts]
        texts_with_features = [self.get_features(text) for text in
processed_texts]

        affect_classified_texts = [self.classify_affect(text) for text in
texts_with_features]

        neutral_texts = [text for text in affect_classified_texts if
text['sentiment'] == 0]
        polar_texts = [text for text in affect_classified_texts if
text['sentiment'] == 1]

        polarity_classified_texts = [self.classify_polarity(text) for text in
polar_texts]

        classified_texts = [{
```

```
            'text': text['text'],
            'sentiment': text['sentiment']
        } for text in neutral_texts + polarity_classified_texts]

        sentiments = [text['sentiment'] for text in classified_texts]

        positive_sentiment_percent = sentiments.count(1) / len(sentiments) * 100
        negative_sentiment_percent = sentiments.count(-1) / len(sentiments) *
100
        neutral_sentiment_percent = sentiments.count(0) / len(sentiments) * 100

        overall_sentiment = 'Positive' if positive_sentiment_percent >
neutral_sentiment_percent and positive_sentiment_percent >
negative_sentiment_percent else \
            'Negative' if negative_sentiment_percent >
positive_sentiment_percent and negative_sentiment_percent >
neutral_sentiment_percent else 'Neutral'

        summary = {
            'overall_summary': overall_sentiment,
            'positive_sentiment_percent': f'{positive_sentiment_percent:.2f}%',
            'negative_sentiment_percent': f'{negative_sentiment_percent:.2f}%',
            'neutral_sentiment_percent': f'{neutral_sentiment_percent:.2f}%'
        }

        result = {
            'summary': summary,
            'texts': classified_texts
        }

        return result

    def process_text(self, text: str):
        clean_text = re.sub('@[^ ]+|http[^ ]+|RT', '', text)

        tokenized_text = self.tokenizer.tokenize(clean_text.lower())

        processed_text = []

        for token in tokenized_text:

            if token not in self.stop_words:
                stemmed_token = self.stemmer.lemmatize(token)

                processed_text.append(stemmed_token)

        return {
            'text': text,
            'processed_text': processed_text
        }

    def get_affect(self, text: str):
        text_object = NRCLex(text)
        freqs = text_object.raw_emotion_scores

        return [freqs.get(emotion, 0) for emotion in self.emotions]

    def get_pos_affects(self, text: list):
        features = []
```

```
        pos_tagged = nltk.pos_tag(text)
        pos_words = {
            'NN': [],
            'VB': [],
            'JJ': [],
            'RB': []
        }

        for word in pos_tagged:
            pos = word[1][:2]

            if pos in pos_words:
                pos_words[pos].append(word[0])

        features.extend([len(pos_words[pos]) for pos in pos_words])

        return features

    def get_text_basic_statistics(self, tokenized_text, raw_text):
        dictionary_words = len(tokenized_text)
        words = len(raw_text.split(' '))
        exclamation = raw_text.count('!')
        capitalized = sum([1 for letter in raw_text if letter.isupper()]) /
len(raw_text)

        targets = len(re.findall('@[^ ]+', raw_text))
        urls = len(re.findall('http[^ ]+', raw_text))
        hashtags = len(re.findall('#[^ ]+', raw_text))

        features = [dictionary_words, words, exclamation, capitalized, targets,
urls, hashtags]

        return features

    def get_negative_words(self, text):
        negative_words = re.findall(" +(no|not|non) +|(n'*t) +| +(non-)", text)

        return [len(negative_words)]

    def get_emojis(self, text):
        pos_emojis = re.findall('|'.join(positive_emojis), text)
        neg_emojis = re.findall('|'.join(negative_emojis), text)
        ex_emojis = re.findall('|'.join(exclamation_emojis), text)

        return [len(pos_emojis), len(neg_emojis), len(ex_emojis)]

    def get_polarity_scores(self, text):
        scores = self.sentiments.polarity_scores(text)

        return [scores['pos'], scores['neg'], scores['neu'], scores['compound']]

    def get_features(self, text: dict):
        processed_text = text['processed_text']
        raw_text = text['text']
        features = []

        features.extend(self.get_text_basic_statistics(processed_text,
raw_text))
        features.extend(self.get_affect(' '.join(processed_text)))
```

```
        features.extend(self.get_pos_affects(processed_text))
        features.extend(self.get_negative_words(raw_text))
        features.extend(self.get_emojis(raw_text))
        features.extend(self.get_polarity_scores(' '.join(processed_text)))

        text['features'] = features

        return text

    def classify_affect(self, text):
        features = np.array(text['features'])
        features = features.reshape(1, -1)

        affect = int(self.affect_classifier.predict(features)[0])

        text['sentiment'] = affect

        return text

    def classify_polarity(self, text):
        features = np.array(text['features'])
        features = features.reshape(1, -1)

        polarity = int(self.polarity_classifier.predict(features)[0])

        text['sentiment'] = polarity if polarity == 1 else -1

        return text


def get_classifier():
    return Classifier()
```

root.py

```
from fastapi import APIRouter, Depends
from api.api_models.models import TitleModel
from api.scrapers.tweeter_scraper import TweeterScraper, get_tweeter_scraper
from api.scrapers.youtube_scraper import YoutubeScraper, get_youtube_scraper
from utilities.classifier import Classifier, get_classifier
router = APIRouter()


@router.post('/get_title_sentiment_analysis')
async def get_title_sentiment_analysis(title: TitleModel,
                                       twitter: TweeterScraper =
Depends(get_tweeter_scraper),
                                       youtube: YoutubeScraper =
Depends(get_youtube_scraper),
                                       classifier: Classifier =
Depends(get_classifier)):
    tweets = twitter.get_tweets(title.title, 5000)
    youtube_comments = await youtube.get_title_comments(title.title,
title.youtube_url)

    all_texts = list(set(tweets + youtube_comments))
```

```
result = classifier.classify_texts(all_texts)

return result
```