Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

АНАЛІЗ ТОНАЛЬНОСТІ ТЕКСТУ МЕТОДАМИ МАШИННОГО НАВЧАННЯ

ANALYSIS OF TEXT TONALITY BY MACHINE LEARNING METHODS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІс-21
Фесіна Олексій Олександрович

(підпис)

Керівник:
к.т.н., доцент
Мельник Катерина Вікторівна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Фесіна Олексій Олександрович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Аналіз тональності тексту методами машинного навчання*

Керівник роботи *к.т.н., доцент Мельник Катерина Вікторівна*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Теоретична частина
Аналітична частина
Практична частина
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Інтерфейс системи
Презентація
Зображення коду

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Мельник К.В.*		
Теоретичне дослідження та практична реалізація	*Мельник К.В.*		
Практична реалізація об'єкта проектування	*Мельник К.В.*		
Висновки	*Мельник К.В.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	До 15.11.2022 р.	Виконано
2.	*Огляд літератури із досліджуваної проблеми*	До 15.12.2022 р.	Виконано
3.	*Розділ 1*	До 02.02.2023 р.	Виконано
4.	*Розділ 2*	До 02.03.2023 р.	Виконано
5.	*Висновок*	До 02.04.2023 р.	Виконано
6.	*Формування списку використаних джерел*	До 15.04.2023 р.	Виконано
7.	*Формування додатків*	До 02.05.2023 р.	Виконано
8.	*Оформлення ілюстративного матеріалу*	До 15.05.2023 р.	Виконано
9.	*Нормоконтроль*	До 25.05.2023 р.	Виконано
10.	*Інструментальна перевірка на академічний плагіат*	До 01.06.2023 р.	Виконано
11.	*Представлення кваліфікаційної роботи бакалавра до захисту*	До 07.06.2023 р.	Виконано

Здобувач вищої освіти _____ Фесінв О.О.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Мельник К.В.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Фесіна О.О. Аналіз тональності тексту методами машинного навчання. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел.

Перший розділ присвячено огляду предметної області, тут розглядаються основні поняття про аналіз тональності тексту, практичний застосунок та їх недоліки.

В другому розділі проведено розгляд методів аналізу. Їх видів та видів інструментів аналізу, наведено приклади онлайн інструментів, розглянуто популярні бібліотеки NLP та розглянуто етапи аналізу тексту.

Третій розділ присвячено опису використання NLTK, NUMPY, PANDAS та SCIKIT-LEARN на мові програмування Python для аналізу текстової бібліотеки з оглядами фільмів доступної в бесплатному доступі NLTKL.

Об'єкт – технології аналізу тональності тексту методами машинного навчання.

Предмет – токенізація та сегментація, лематизація та стемінг, синтаксичний аналіз.

Метою роботи є опанування знаннями та навиками аналізу тональності тексту методами машиного навчання, отримати розумуння NLP, та використати отриманні навички на практиці.

Ключові слова: NLP, NLTK, Python, Аналіз настроїв,

ABSTRACT

Fesina O.O. Analysis of text tonality using machine learning methods. Manuscript.

Bachelor's qualifying thesis of the OP "Computer Engineering" specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

The qualification work consists of an introduction, three sections, conclusions, and a list of used sources.

The first chapter is dedicated to the overview of the subject area, here the main concepts of the analysis of the tonality of the text, their practical application and their shortcomings are considered.

In the second section, analysis methods are considered. Their types and types of analysis tools, examples of online tools are given, popular NLP libraries are considered, and the stages of text analysis are considered.

The third chapter describes the use of NLTK, NUMPY, PANDAS, and SCIKIT-LEARN in the Python programming language to analyze a text library of movie reviews freely available at NLTKL.

The object is the technology of text tonality analysis using machine learning methods.

The subject is tokenization and segmentation, lemmatization and stemming, syntactic analysis.

The purpose of the work is to master the knowledge and skills of text tonality analysis using machine learning methods, to gain an understanding of NLP, and to use the acquired skills in practice.

Keywords: NLP, NLTK, Python, tonality analysis.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів

NLP розшифровується як Natural Language Processing, яка є підгалуззю інформатики та штучного інтелекту, яка зосереджується на взаємодії між комп'ютерами та людською мовою.

Python — це популярна мова програмування високого рівня, яка використовується для широкого кола програм, від веб-розробки та наукових обчислень до аналізу даних і штучного інтелекту.

NLTK означає Natural Language Toolkit, що є бібліотекою Python, яка використовується для завдань обробки природної мови, таких як аналіз тексту, токенізація, аналіз і класифікація.

Harvard General Inquirer — це комп'ютерна програма, розроблена в 1960-х роках дослідниками з Гарвардського університету. Його основна мета — аналізувати текстові дані та кількісний аналіз змісту, стилю та структури тексту.

CNN розшифровується як згорточна нейронна мережа, яка є типом глибокої нейронної мережі, яка зазвичай використовується для аналізу зображень і відео, але також використовується в інших сферах, як-от обробка природної мови.

WordNetLemmatizer — це інструмент для лемматизації слів англійською мовою. WordNetLemmatizer є частиною інструментарію Natural Language Toolkit (NLTK), він використовує лексичну базу даних WordNet, яка є великим електронним словником, який групує англійські слова в набори синонімів, які називаються синсетами, і надає інформацію про їхні семантичні зв'язки.

RNN означає повторювану нейронну мережу, яка є типом нейронної мережі, призначеної для обробки послідовних даних, таких як часові ряди або природна мова. На відміну від нейронних мереж прямого зв'язку, які обробляють вхідні дані у фіксованому порядку, RNN можуть підтримувати внутрішню пам'ять або стан для послідовної обробки вхідних даних.

NumPy розшифровується як Numerical Python, це бібліотека Python, яка використовується для наукових обчислень і аналізу даних.

Pandas — це бібліотека Python для обробки та аналізу даних. Він надає високопродуктивні, прості у використанні структури даних і інструменти аналізу

даних, що робить його незамінним інструментом для дослідників даних, аналітиків і розробників, які працюють зі структурованими даними.

ВСТУП

Актуальність теми. Аналіз тексту за допомогою методів машинного навчання залишається дуже актуальним і є областю активних досліджень і розробок станом на 2023 рік. Алгоритми машинного навчання все частіше використовуються для обробки, розуміння та створення людської мови, сприяючи розвитку обробки природної мови (NLP).

Машинне навчання використовується для розуміння настроїв, переданих у блоці тексту, будь то позитивні, негативні чи нейтральні. Це може бути особливо цінним для компаній, які хочуть зрозуміти відгуки клієнтів або настрої соціальних мереж щодо свого бренду чи продукту.

Машинне навчання, особливо використання моделей глибокого навчання, революціонізувало сферу машинного перекладу, зробивши автоматизований переклад дедалі точнішим і плавнішим.

Підсумовуючи, використання методів машинного навчання в аналізі тексту є не тільки актуальною, але й швидко розвивається сферою. Застосування варіюються від аналізу настроїв, класифікації тексту та вилучення інформації до створення тексту, перекладу, відповідей на запитання, резюмування, розпізнавання іменованих об’єктів і моделювання тем. Ці можливості мають глибокий вплив на безліч галузей, включаючи, але не обмежуючись, обслуговування клієнтів, маркетинг, журналістику та технології.

Метою роботи Аналіз тональності тексту шляхом впровадження техніки машинного навчання з метою практичного застосування отриманих знань.

Об'єкт дослідження – Аналіз тональності тексту за допомогою машинних методів та його технологічна реалізація.

Предмет дослідження – Розробка моделі аналізу текстових даних

Завдання, які необхідно виконати:

— Реалізувати огляд теми аналізу тональності тексту

— Дослідити інструменти аналізу та NLP

— Спроектувати модель аналізу тексту

РОЗДІЛ 1
АНАЛІТИЧНА ЧАСТИНА

1.1 Огляд аналізу тональності тексту

Аналіз тональності тексту, також відомий як аналіз настроїв, — це техніка, метою якої є ідентифікація та кількісна оцінка емоційного тону фрагмента тексту. Цей аналіз стає все більш важливим, оскільки все більше компаній і організацій звертаються до соціальних медіа та інших онлайн-платформ для взаємодії зі своїми клієнтами та зацікавленими сторонами. Аналізуючи настрої цих взаємодій, компанії можуть отримати уявлення про ставлення та думки своєї цільової аудиторії, що може стати основою для їхніх маркетингових і комунікаційних стратегій.[1]

1.2 Практичний застосунок

Одним із практичних застосувань аналізу тональності тексту є обслуговування клієнтів. Аналізуючи настрої відгуків клієнтів, компанії можуть визначати потенційні проблеми або сфери, що викликають занепокоєння, що дозволяє їм швидко й активно реагувати на потреби клієнтів. Аналіз настроїв також можна використовувати для моніторингу брендів, дозволяючи компаніям відстежувати, як їх бренд сприймається громадськістю, і відповідним чином коригувати свої повідомлення.

Ще одна сфера, де аналіз тональності тексту важливий, це політичні кампанії. Аналізуючи соціальні медіа та інші онлайн-платформи, політичні кампанії можуть отримати уявлення про думки та занепокоєння виборців, дозволяючи їм створювати повідомлення та політику, які резонують із цільовою аудиторією.

1.3 Недоліки

Аналіз тональності тексту також представляє кілька проблем. Однією з найбільших проблем є точне визначення настрою фрагмента тексту. Це особливо складно, коли мова йде про сарказм чи іронію, які машинам може бути важко виявити. Крім того, різні мови та культури можуть по-різному виражати почуття, що може ускладнити аналіз міжкультурних настроїв.

1.4 Використання методів машинного навчання в аналізі тональності тексту

Методи машинного навчання широко використовуються в аналізі тексту та довели свою ефективність у багатьох програмах. Зокрема, методи машинного навчання, які стосуються використання алгоритмів, які вивчають дані для прогнозування або прийняття рішень, набули значної популярності в останні роки завдяки своїй здатності аналізувати великі обсяги текстових даних за короткий проміжок часу.

Одним із найбільш часто використовуваних методів машинного навчання в аналізі тексту є контрольоване навчання. У цьому підході модель машинного навчання навчається на позначеному наборі даних, де кожна точка даних пов’язана з відомою міткою або категорією. На основі цих даних модель вчиться передбачати мітку нових, невидимих точок даних. Цей підхід використовувався для таких завдань, як аналіз настроїв, моделювання теми та класифікація тексту.

Інший метод машинного навчання, який використовується в аналізі тексту, — навчання без нагляду. Цей підхід не покладається на дані з мітками, а натомість намагається визначити закономірності чи групи в даних. Неконтрольоване навчання використовувалося для таких завдань, як кластеризація подібних документів або визначення загальних тем у великих колекціях тексту.

Глибоке навчання — ще один тип машинного навчання, який успішно застосовувався для аналізу тексту. Моделі глибокого навчання, такі як нейронні

мережі, можуть вивчати складні моделі та зв’язки в текстових даних, що робить їх ефективними для таких завдань, як обробка природної мови, генерація тексту та переклад мови.

1.5 Плюси використання методів машинного навчання

На додаток до ефективності методів машинного навчання в аналізі тексту, вони також пропонують кілька переваг перед традиційними методами, заснованими на правилах. Наприклад, методи на основі правил покладаються на попередньо визначені правила, створені експертами домену, що може зайняти багато часу та важко масштабуватись для великих наборів даних або складних завдань. З іншого боку, методи машинного навчання можуть автоматично навчатися на основі даних без необхідності ручного втручання, що робить їх більш адаптивними та масштабованими.

Крім того, машинні методи навчання можуть покращуватися з часом за наявності нових даних. У міру того, як стане доступним більше даних, моделі машинного навчання можна перенавчати та налаштовувати, щоб покращити їх продуктивність, забезпечуючи більш точні та відповідні результати.

Однак існують також деякі проблеми, пов’язані з використанням методів машинного навчання в аналізі тексту. Однією з основних проблем є проблема упередженості. Моделі машинного навчання неупереджені настільки, наскільки неупереджені дані, які використовуються для їх навчання, і якщо дані містять упередження, ці упередження можуть бути посилені в результатах, створених моделлю. Таким чином, важливо переконатися, що дані, які використовуються для навчання моделей, є різноманітними та репрезентативними для цільової групи, щоб уникнути збереження упереджень.

Іншою проблемою є потреба у високоякісних анотованих даних для навчання моделей. Анотовані дані – це дані, позначені певними категоріями або атрибутами, які необхідні для методів навчання під наглядом. Однак процес анотування даних може бути тривалим і дорогим, особливо для складних завдань.

РОЗДІЛ 2
ТЕОРЕТИЧНА ЧАСТИНА

2.1 Методи аналізу тональності тексту

Аналіз тональності тексту — це процес аналізу настрою або емоційного тону фрагмента тексту. Існує кілька методів проведення аналізу тональності тексту, зокрема:

Аналіз на основі лексики: цей підхід передбачає використання попередньо створеного лексикону або словника слів із пов’язаними оцінками настрою. Оцінки настрою можуть бути позитивними, негативними або нейтральними, і вони призначаються окремим словам на основі їх семантичного значення. Потім обчислюється загальний настрій фрагмента тексту шляхом агрегування балів настрою слів у тексті.

Аналіз на основі машинного навчання: цей підхід передбачає навчання моделі машинного навчання на позначеному наборі даних текстів із відомими показниками настрою. Модель вчиться визначати шаблони в тексті, які асоціюються з позитивними, негативними або нейтральними настроями. Коли модель навчена, її можна використовувати для прогнозування настрою нових текстів.

Аналіз на основі правил: цей підхід передбачає використання набору створених вручну правил для визначення настроїв у тексті. Правила можуть базуватися на конкретних шаблонах слів або комбінаціях слів, які асоціюються з певними емоціями. Аналіз на основі правил може бути більш точним, ніж аналіз на основі лексики, але його розробка також може бути більш трудомісткою.

Гібридні підходи: багато дослідників і практиків поєднують кілька методів, щоб досягти більш точного аналізу настроїв. Наприклад, підхід, заснований на лексиконах, може бути використаний як базовий, з моделлю машинного навчання, використаною для точного налаштування результатів. Крім того, підхід, заснований на правилах, може бути використаний для визначення конкретних

типів настроїв, а модель машинного навчання використовується для виявлення більш тонких варіацій тону.

2.1.1 Аналіз тональності тексту на основі лексики

Лексичний аналіз тональності тексту — метод визначення емоційної тональності тексту за словами, які вживаються в ньому. Цей підхід ґрунтується на використанні лексикону, який є словником, який містить слова та пов’язані з ними емоційні тони.

Лексикон, який використовується для аналізу тональності, може бути або лексиконом загального призначення, таким як Harvard General Inquirer (Harvard General Inquirer — це комп’ютерна програма, розроблена в 1960-х роках дослідниками з Гарвардського університету. Його основна мета — аналізувати текстові дані та кількісний аналіз змісту, стилю та структури тексту.), або лексиконом для певної галузі, створеним спеціально для певної галузі чи теми. Лексикон містить перелік слів і фраз, які асоціюються з позитивними, негативними або нейтральними емоціями.

Щоб проаналізувати тональність тексту за допомогою лексикону, текст спочатку попередньо обробляється для видалення стоп-слів, знаків пунктуації та інших нерелевантних елементів. Потім кожне слово в тексті шукається в лексиконі та визначається його емоційний тон. Потім емоційні відтінки всіх слів у тексті об’єднуються, щоб визначити загальну тональність тексту.

Існують різні підходи до поєднання емоційної тональності слів у тексті. Один з поширених підходів полягає в тому, щоб просто підрахувати кількість слів, пов’язаних з кожним емоційним тоном, і використовувати це як міру тональності. Інший підхід полягає у використанні зваженої схеми, де слова, які сильніше асоціюються з певним емоційним тоном, отримують більшу вагу.

Аналіз тональності тексту на основі лексиконів можна використовувати для широкого спектру програм, таких як аналіз настроїв, моніторинг бренду та аналіз відгуків клієнтів. Однак він має певні обмеження, такі як нездатність вловити контекст і відтінки мови, а також залежність від якості та повноти використаного лексикону.

Іншим обмеженням аналізу тональності тексту на основі лексикону є те, що він може не вміти вловити інтенсивність або ступінь емоцій у тексті. Наприклад, текст, який містить як позитивні, так і негативні слова, можна класифікувати як нейтральний, якщо кількість позитивних і негативних слів приблизно однакова, навіть якщо загальний настрій тексту є сильно позитивним або негативним.

Щоб подолати ці обмеження, деякі просунуті методи, засновані на словнику, використовують такі методи, як аналіз контексту, тегування частин мови та усунення неоднозначності слів для підвищення точності аналізу тональності. Ці методи намагаються визначити контекст, у якому використовується слово, і значення, у якому воно мається на увазі, що може допомогти вловити відтінки мови та зробити більш точний аналіз тональності.

Інший підхід до аналізу тональності — це методи, засновані на машинному навчанні, які використовують алгоритми для вивчення емоційних відтінків слів та їх комбінацій із навчального набору позначених даних. Ці методи можуть фіксувати складніші зв’язки між словами та проводити більш точний тональний аналіз. Однак вони потребують великої кількості мічених даних для навчання, отримання яких може бути дорогим і трудомістким.

Аналіз тональності тексту на основі лексикону широко використовується в аналізі настроїв, який передбачає автоматичну класифікацію думок, ставлень та емоцій, виражених у тексті. Аналіз настроїв має багато практичних застосувань, таких як моніторинг брендів, аналіз відгуків клієнтів, моніторинг соціальних медіа та дослідження ринку.

Наприклад, у моніторингу брендів компанії можуть використовувати аналіз настроїв, щоб відстежувати сприйняття громадськістю свого бренду та продуктів, аналізуючи відгуки клієнтів, публікації в соціальних мережах та інший онлайн-контент. Це може допомогти компаніям визначати сфери для вдосконалення, реагувати на скарги та занепокоєння клієнтів і приймати обґрунтовані бізнес-рішення.

Так само в аналізі відгуків клієнтів аналіз настроїв можна використовувати для автоматичної класифікації відгуків клієнтів як позитивних, негативних або

нейтральних і визначення загальних тем і проблем. Це може допомогти компаніям визначити області для вдосконалення та визначити пріоритети своїх зусиль для покращення задоволеності клієнтів.

У моніторингу соціальних медіа аналіз настроїв можна використовувати для відстеження настроїв дописів і коментарів, пов’язаних із певною темою чи подією, як-от запуск продукту чи політична кампанія. Це може дати цінну інформацію про громадську думку та допомогти організаціям відповідно скоригувати свої стратегії та повідомлення.

2.1.2 Аналіз тональності тексту на основі правил

Аналіз тональності тексту на основі правил передбачає використання набору попередньо визначених правил для визначення емоційного тону чи почуття, вираженого у фрагменті тексту. Процес зазвичай включає розбиття тексту на складові частини, такі як окремі слова та фрази, а потім їх аналіз на основі попередньо визначених правил або алгоритмів.

Один підхід до аналізу на основі правил передбачає створення словника або лексикону слів і фраз, які асоціюються з різними емоційними станами, такими як позитивні, негативні чи нейтральні. Кожному слову в словнику присвоюється бал або вага на основі його емоційного відтінку. Наприклад, слово «любов» може отримати оцінку +5 за позитивний настрій, тоді як слово «ненависть» може отримати оцінку -5 за негативний настрій.

Потім алгоритм на основі правил працює, аналізуючи текст і призначаючи оцінку настрою кожному слову чи фразі на основі оцінок у словнику. Потім ці бали агрегуються, щоб отримати загальну оцінку настрою для тексту.

Інший підхід до аналізу на основі правил передбачає використання граматичних правил і структур для визначення емоційного тону тексту. Наприклад, певні структури речень, такі як вигуки чи риторичні запитання, можуть сильніше асоціюватися з позитивними чи негативними настроями.

Загалом, хоча аналіз на основі правил може бути корисним для виявлення широких моделей почуттів у тексті, він обмежений необхідністю попередньо

визначених правил і не завжди може вловлювати нюанси та складність людських емоцій, виражених мовою.

2.1.3 Аналіз тональності тексту гібридними методами

Гібридні підходи до аналізу тональності тексту поєднують кілька методів і алгоритмів для підвищення точності та надійності аналізу настрою. Деякі приклади гібридних підходів до аналізу тональності тексту:

Підхід на основі лексики з машинним навчанням: цей підхід поєднує метод на основі лексики, де для визначення тональності тексту використовується попередньо визначений набір слів із позитивними, негативними чи нейтральними значеннями, з алгоритмами машинного навчання, які навчаються з попередніх даних для підвищення точності аналізу.

Підхід на основі правил із глибоким навчанням: у цьому підході визначається набір правил для визначення тональності тексту, і це поєднується з алгоритмами глибокого навчання, які використовують нейронні мережі для підвищення точності аналізу.

Лінгвістичний підхід із ансамблевим навчанням: цей підхід поєднує лінгвістичні методи, які аналізують граматичну структуру та синтаксис тексту з ансамблевим навчанням, де кілька алгоритмів навчаються на різних характеристиках даних, а їхні результати поєднуються для підвищення точності аналіз.

Гібридний підхід із тематичним моделюванням: у цьому підході алгоритми тематичного моделювання використовуються для визначення основних тем у тексті, які потім використовуються в поєднанні з іншими методами аналізу настроїв для визначення тональності тексту. Цей підхід особливо корисний, коли на тональність тексту сильно впливає контекст або тема обговорення.

Загалом гібридні підходи до аналізу тональності тексту використовують сильні сторони різних алгоритмів і методів для підвищення точності та надійності аналізу настрою.

2.2 Інструменти аналізу

Для аналізу тональності тексту доступно декілька інструментів, зокрема:

Інструменти аналізу настроїв: ці інструменти використовують алгоритми обробки природної мови (NLP) для аналізу тексту та визначення виражених у ньому настроїв, будь то позитивні, негативні чи нейтральні. Деякі популярні інструменти аналізу настроїв включають IBM Watson, Google Cloud Natural Language API та Amazon Comprehend.

Лінгвістичне дослідження та підрахунок слів (LIWC): LIWC — це програмне забезпечення для аналізу тексту, яке визначає лінгвістичні та психологічні аспекти використання мови. Він широко використовується в дослідженнях соціальної, когнітивної та психологічної психології особистості. LIWC аналізує текст на основі попередньо визначеного набору категорій слів і генерує бали для різних мовних особливостей, таких як тон, емоції та соціальна поведінка.

Інструменти аналізу емоцій: ці інструменти визначають приховані емоції та почуття, виражені в тексті. Вони використовують алгоритми машинного навчання, щоб класифікувати текст за різними емоціями, як гнів, страх, радість і смуток. Деякі популярні інструменти аналізу емоцій включають Receptiviti, SenticNet і Affectiva.

Інструменти на основі лексиконів: ці інструменти спираються на вже існуючий лексикон або словник слів і пов’язані з ними оцінки настрою. Інструмент порівнює слова в тексті з лексиконом і призначає оцінку настрою на основі наявності позитивних чи негативних слів. Приклади інструментів на основі лексиконів включають VADER (Valence Aware Dictionary і sEntiment Reasoner) і SentiWordNet.

2.2.1 Інструменти аналізу настроїв

Інструменти аналізу настроїв є поширеним інструментом для аналізу тональності тексту.

Аналіз настроїв є дуже популярним застосуванням можливостей машинного навчання. Аналізуючи зміст кожного тексту, можна оцінити, наскільки

позитивною чи негативною є вага речення чи всього тексту. Це може мати величезне значення, якщо необхідно відфільтрувати негативні відгуки про свій продукт або представити лише хороші.

В Інтернеті є безліч моделей аналізу настроїв та інструментів для Python. [3] На рисунку 2.1 наведено приклад застосування одного з найпростіших методів аналізу: знадобиться 2 рядки коду, щоб виконати базовий аналіз настрою.

```
# import the package:
from pattern.en import sentiment
# perform the analysis:
x = 'project looks amazing, great job'
sentiment(x)
```

Рисунок 2.1 — Найпростіший метод аналізу

Вихідні дані:

(0,7000000000000001, 0,825)

Як показано вище, після імпортування пакета просто потрібно викликати функцію sentiment і надати їй значення рядка. Результатом є кортеж, першим значенням якого є «полярність» (наскільки позитивним є речення за шкалою від -1 до 1). Друге значення — це суб'єктивність, яка говорить нам, наскільки певний алгоритм щодо оцінки свого першого значення, цього разу шкала починається з 0 і закінчується на 1. Наведено приклад на рисунку 2.2:

```
y = 'plot looks terrible, spines are too small'
sentiment(y)
```

Рисунок 2.2 — Найпростіший метод аналізу

вихідні дані:

(-0,625, 0,7)

Дається досить низьке перше значення, і алгоритм все ще досить впевнено оцінює свою полярність. Нижче наведений більш складний приклад для аналізу як показано на рисунку 2.3:

```
z = 'improve the comment, first line looks bad'
sentiment(z)
```

Рисунок 2.3 — Найпростіший метод аналізу

вихідні дані:

(-0,22499999999999992, 0,5)

Модель більш вагається щодо цього прикладу. Рядок містить мінус-слово, але він уже не такий впевнений.

Наведено приклад застосування функції настроїв до вмісту відгуку на рисунку 2.4:

```
df['sentiment'] = df['feedback_clean2_lem'].apply(lambda x: sentiment(x))
df['polarity'] = df['sentiment'].str[0]
df['subjectivity'] = df['sentiment'].str[1]
df = df.drop(columns='sentiment')
```

Рисунок 2.4 — Вімірювання полярності та суб'єктивності відгуків

Було виміряно полярність і суб'єктивність кожного повідомлення відгуку.

Ключові слова

Вилучення ключових слів із заданого рядка — ще один потужний трюк, який може покращити аналіз.

Пакет Rake надає список усіх n-грамів та їх ваги, витягнутих із тексту. Чим вище значення, тим важливішим є n-грам, який розглядається. Після аналізу тексту є можливість відфільтрувати лише n-грами з найвищими значеннями як показано на рисунку 2.5. [11]

```
from rake_nltk import Rake
# set the parameteres (length of keyword phrase):
r = Rake(include_repeated_phrases=False, min_length=1, max_length=3)
text_to_rake = df['feedback'][31]
r.extract_keywords_from_text(text_to_rake)
# filter out only the top keywords:
words_ranks = [keyword for keyword in r.get_ranked_phrases_with_scores() if keyword[0] > 5]
words_ranks
```

Рисунок 2.5 — Фільтрування ключові слова

Вихідні дані:

[(9.0, '" професійний "'),

(9.0, «уникання розмовної мови»),

(8.0, «добре структурований проект»),

(8.0, «також включено Антарктиду»),

(8.0, «додати крапку»)]

Однак потрібно мати на увазі, що модель використовує стоп-слова, щоб оцінити, які слова є важливими в реченнях. Якби додали цю модель текстом, очищеним від стоп-слів, то не отримали би жодного результату.

У цьому прикладі Рейк вирішив, що «професійний» або «уникання розмовної мови» є найважливішими ключовими словами вхідного тексту. Для подальшого аналізу не потрібні будуть цікаві числові значення ключових слів. Просто необхідно отримати кілька ключових слів для кожної публікації. Наведу приклад на рисунку 2.6 використання простої функції для вилучення лише найпопулярніших ключових слів і застосуємо її до стовпця «відгук»:

```
def rake_it(text):
    r.extract_keywords_from_text(text)
    r.get_ranked_phrases()
    keyword_rank = [keyword for keyword in r.get_ranked_phrases_with_scores() if keyword[0] > 5]
    # select only the keywords and return them:
    keyword_list = [keyword[1] for keyword in keyword_rank]
    return keyword_list

df['rake_words'] = df['feedback'].apply(lambda x: rake_it(x))
```

Рисунок 2.6 — Вилучення найпопулярніших ключових слів

Витягнувши ключові слова з кожної публікації, наступним кроком буде перевірка які з них найпопулярніші як показано на рисунку 2.7. Вони зберігаються як список усередині комірки, тож нам доведеться мати справу з цією перешкодою:

```
# function from: towardsdatascience.com/dealing-with-list-values-in-pandas-dataframes-a177e534f173
def to_1D(series):
    return pd.Series([x for _list in series for x in _list])

to_1D(df['rake_words']).value_counts()[:10]
```

Рисунок 2.7 — Вивід найпопулярніших слів

Вихідні дані:

Меню файлів jupyter 24

Кероване використання проекту 24

щасливого кодування :) 22

все виглядає добре 20

посібник зі стилю sql 16

перший керований проект 16

все виглядає добре 15

кнопка нової теми 15

файл блокнота jupyter 14

перша клітинка коду 13

dtype: int64

Моделювання теми

Моделювання теми може швидко дати уявлення про зміст тексту. На відміну від вилучення ключових слів із тексту, моделювання тем є набагато більш просунутим інструментом, який можна налаштувати відповідно до потреб.[10]

Наведено проста базова версія цієї моделі для кожного вмісту відгуку.

Наведено приклад з однієї публікації відгуку. Імпортовано необхідні пакети, скомпільований текст і створено необхідні словники та матриці на рисунку 2.8.

```
import gensim
from gensim import corpora
import re

doc_complete = df['feedback_clean2_lem'][0]
docs = word_tokenize(doc_complete)
docs_out = []
docs_out.append(docs)
dictionary = corpora.Dictionary(docs_out)
doc_term_matrix = [dictionary.doc2bow(doc) for doc in docs_out]
```

Рисунок 2.8 — Імпортування та компілювання

Виконавши всю підготовчу роботу, настав час навчити модель і отримати результати. Вручну встановлюються багато параметрів як показано на рисунку 2.9, такі як: кількість тем, скільки слів використовується на тему. Але цей список можна продовжувати, і, як і у випадку з багатьма моделями ML, можливо витратити багато часу на налаштування цих параметрів, щоб вдосконалити свою модель.[12]

```
Lda = gensim.models.ldamodel.LdaModel
ldamodel = Lda(doc_term_matrix, num_topics=5, id2word=dictionary, passes=50, random_state=4)
ldamodel.print_topics(num_topics=5, num_words=4)
```

Рисунок 2.9 — Список параметрів

Вихідні дані:

[(0, '0.032*"notebook" + 0.032*"look" + 0.032*"process" + 0.032*"month"'),
(1, '0.032*"notebook" + 0.032*"look" + 0.032*"process" + 0.032*"month"'),
(2, '0.032*"important" + 0.032*"stay" + 0.032*"datasets" + 0.032*"process"'),
(3,

'0.032*"httpswww1nycgovsitetlcabouttlctriprecorddatapage" + 0.032*"process" + 0.032*"larger" + 0.032*"clean"'),

(4, '0.113*"function" + 0.069*"inside" + 0.048*"memory" + 0.048*"ram"')]

Модель надасть список кортежів, кожен кортеж містить слова та їх вагу. Чим більше число, тим важливіше слово.

Наступний крок буде застосування моделі до кожного повідомлення відгуку. Для спрощення роботи, потрбіно витягнути лише слова теми, а не значення «ваги». Таким чином можливе легке виконання value_counts для вилучених тем і вивеведення, які теми були найпопулярнішими (відповідно до моделі). Щоб виконати тематичне моделювання для кожної комірки в стовпці, потрібно розробити функцію. Як вхідні значення потрібно буде використовувати вміст комірки (текст) і кількість слів для теми як показано на рисунку 2.10:

```
def get_topic(x, n):
    docs = word_tokenize(x)
    docs_out = []
    docs_out.append(docs)
    dictionary = corpora.Dictionary(docs_out)
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in docs_out]
    Lda = gensim.models.ldamodel.LdaModel
    ldamodel = Lda(doc_term_matrix, num_topics=5, id2word=dictionary, passes=50, random_state=1)
    topics = ldamodel.print_topics(num_topics=2, num_words=n)
    topics_list = []
    for elm in topics:
        content = elm[1]
        no_digits = ''.join([i for i in content if not i.isdigit()])
        topics_list.append(re.findall(r'\w+', no_digits, flags=re.IGNORECASE))
    return topics_list
```

Рисунок 2.10 — Фільтрування слів теми

Наведений приклад, як виглядають теми з максимальною довжиною 4 слова як показано на рисунку 2.11:

```
df['topic_4'] = df['feedback_clean2_lem'].apply(lambda x: get_topic(x,4))
to_1D(df['topic_4']).value_counts()[:10]
```

Рисунок 2.11 — Вивід тем з максимальною довжиною 4 слова:

Вихідні дані:

[keep, nice] 8

[nice, perform] 4

[nice] 4

[nan] 4

[sale, take, look, finish] 2

[learning, keep, saw, best] 2

[library, timezones, learning, httpspypiorgprojectpytz] 2

[job, graph, please, aesthetically] 2

[thanks, think, nice, learning] 2

[especially, job, like, nice] 2

dtype: int64

А тепер наведено приклад теми максимум з 3 слів як показано рисунку 2.12:

```
df['topic_3'] = df['feedback_clean2_lem'].apply(lambda x: get_topic(x,3))
to_1D(df['topic_3']).value_counts()[:10]
```

Рисунок 2.12 — Вивід тем з максимальною довжиною 3 слова

Вихідні дані:

[keep, nice] 8

[share, thank, please] 4

[nan] 4

[nice] 4

[nice, perform] 4

[guide, documentation, project] 3

[] 3

[cell] 3

[guide, project, documentation] 3

[plot] 3

dtype: int64

K-means кластеризація

Модель Kmeans може кластеризувати дані на основі різних вхідних даних, це, мабуть, найпопулярніша модель неконтрольованого машинного навчання. Просто вибрати, скільком кластерам потрібно призначити дані, залежно від функцій і модель виконує кластеризацію. Будучи моделлю ML, користувач не може просто заповнити її необробленим текстом, необхнідно векторизувати текстові дані, а потім заповнити ними модель. По суті, перетворюємо текстові дані в числові дані. Існує багато способів векторизації даних, наведу приклад TfidfVectorizer як показано на рисунку 2.13:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

tfidfconverter = TfidfVectorizer(max_features=5000, min_df=0.1, max_df=0.7, stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(df['feedback']).toarray()

Kmean = KMeans(n_clusters=8, random_state=2)
Kmean.fit(X)
df['label'] = Kmean.labels_
```

Рисунок 2.13 — Векторизація даних

Тепер потрібно подивитись, чи мають різні кластери велику різницю в полярності як показано на рисунку 2.14:

```
df.groupby('label')['polarity'].mean()
```

Рисунок 2.14 — Перевірка полярності

Вихідні дані:

label

0 0.405397

1 0.312328

2 0.224152

3 0.210876

4 0.143431

5 0.340016

6 0.242555

7 0.241244

Name: polarity, dtype: float64

Наступний кроком буде перевірка інших чисел на основі номера кластера, призначеного нашою моделлю як показано на рисунку 2.15:

```
polar = df.groupby('label')['polarity'].mean()
subj = df.groupby('label')['subjectivity'].mean()
level = df.groupby('label')['level'].mean()
val_cnt = df['label'].value_counts()
length = df.groupby('label')['len'].mean()

cluster_df = pd.concat([val_cnt,polar,subj,level, length], axis=1)
cluster_df = cluster_df.rename(columns={'label':'count'})
cluster_df.index.name = 'label'
cluster_df
```

Рисунок 2.15 — Перевірка чисел на основі номера кластера

Як наведено на рисунку 2.16 можна помітити деякі цікаві тенденції в цій таблиці, наприклад. кластер номер 0 має досить позитивний вміст (висока полярність/polarity), також модель настрою, яку було обрано для використовування раніше в цьому кластері, досить впевнена щодо свого моделювання (високе значення суб'єктивності/subjectivitiy). Це може бути викликано дуже короткою середньою довжиною тексту в цьому кластері (314/len).

Необхідно звернути увагу, що модель Kmeans була заповненна даними, векторизованими за допомогою Tfidf, є кілька способів векторизації текстових даних перед подачею їх до моделі.

label	count	polarity	subjectivity	level	len
0	87	0.405397	0.635069	1.49425	314.287
1	150	0.312328	0.536363	1.55333	742.253
2	60	0.224152	0.469265	1.5	594.267
3	136	0.210876	0.513048	1.46324	1429.1
4	66	0.143431	0.34258	1.4697	251.227
5	118	0.340016	0.581554	1.29661	903.11
6	302	0.242555	0.495008	1.45033	724.209
7	92	0.241244	0.431905	1.52174	398.228

Рис 2.16 — Вивід кластерів номерів 0-7

Групування речень

Як уже згадувалося раніше: зміст кожного допису відгуку є досить складною сумішшю компліментів і конструктивної критики. Ось чому раніше модель кластеризації не працювала добре, коли її просили кластеризувати публікації. Але якби користувач розділив всі публікації на речення та попросили модель згрупувати речення, повинно покращити результати моделі. Як показано на рисунку 2.17. [13]

```
from nltk.corpus import stopwords
stop = set(stopwords.words('english'))
from nltk import sent_tokenize
from nltk import pos_tag
import string
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
exclude = set(string.punctuation)
def clean(doc):
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    return punc_free
def lemmatize_it(sent):
    empty = []
    for word, tag in pos_tag(word_tokenize(sent)):
        wntag = tag[0].lower()
        wntag = wntag if wntag in ['a', 'r', 'n', 'v'] else None
        if not wntag:
            lemma = word
            empty.append(lemma)
        else:
            lemma = lemmatizer.lemmatize(word, wntag)
            empty.append(lemma)
    return " ".join(empty)
doc_complete = sent_tokenize(df['feedback'].sum())
doc_clean = [clean(doc) for doc in doc_complete]
doc_clean_lemmed = [lemmatize_it(doc) for doc in doc_clean]
sentences = pd.DataFrame(doc_clean_lemmed)
sentences.columns = ['sent']
sentences['orig'] = doc_complete
sentences['keywords'] = sentences['orig'].apply(lambda x: rake_it(x))
tfidfconverter = TfidfVectorizer(max_features=10, min_df=0.1, max_df=0.9, stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(sentences['sent']).toarray()
Kmean = KMeans(n_clusters=8)
Kmean.fit(X)
sentences['label'] = Kmean.labels_
```

Рисунок 2.17 — Розділ публікації на речення та згрупування речень

Спосіб перевіріти найпопулярніші ключові слова для кожної мітки як показано на рисунку 2.18 і рисунку 2.19: [8]

```
to_1D(sentences[sentences['label'] == 0]['keywords']).value_counts()[:10]
```

Рисунок 2.18 — Перевірка найпопулярніших слів для мітки 0

Вихідні дані:

everything looks nice 15

everything looks good 13

sql style guide 10

print () function 8

social love attention 7

data cleaning process 7

looks pretty good 6

screen shot 2020 5

everything looks great 5

jupyter notebook file 5

dtype: int64

```
to_1D(sentences[sentences['label'] == 2]['keywords']).value_counts()[:10]
```

Рисунок 2.19 — Перевірка найпопулярніших слів для мітки 2

Вихідні дані:

first code cell 8

avoid code repetition 7

items (): spine 4

might appear complex 4

may appear complex 4

might consider creating 3

1st code cell 3

print () function 3

little suggestion would 3

one code cell 3

dtype: int64

Кластеризація

Подібно до кластеризації дописів і речень, при потребі можливо виконати кластеризацію n-грам як показано на рисунку 2.20: [14]

```
from nltk.util import ngrams
import collections

tokenized = word_tokenize(sentences['sent'].sum())
trigrams = ngrams(tokenized, 3)
trigrams_freq = collections.Counter(trigrams)
trigram_df = pd.DataFrame(trigrams_freq.most_common())
trigram_df.columns = ['trigram','count']
trigram_df['tgram'] = trigram_df['trigram'].str[0]+' '+trigram_df['trigram'].str[1]+' '+trigram_df['trigram'].str[2]

tfidfconverter = TfidfVectorizer(max_features=100, min_df=0.01, max_df=0.9, stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(trigram_df['tgram']).toarray()

Kmean = KMeans(n_clusters=8, random_state=1)
Kmean.fit(X)
trigram_df['label'] = Kmean.labels_
```

Рисунок 2.20 — Кластеризація n-gram

Конкордації та Сполучення — Concordance and Collocations

У контексті NLP конкорданс — це набір місць розташування слів разом із їхнім контекстом. Ми можемо використовувати конкордації, щоб знайти:

– Скільки разів з’являється слово.

– Де з’являється кожен випадок.

– Які слова оточують кожен випадок.

У NLTK це можна зробити, викликавши .concordance(). Щоб використовувати його, потрібен екземпляр класу nltk.Text, який також можна створити за допомогою списку слів.[18]

Перш ніж викликати .concordance(), потрібно створити новий список слів з вихідного тексту корпусу, щоб там був весь контекст, навіть стоп-слова як показано на рисунку 2.21:

```
>>> text = nltk.Text(nltk.corpus.state_union.words())
>>> text.concordance("america", lines=5)
Displaying 5 of 1079 matches:
 would want us to do . That is what America will do . So much blood has already
ay , the entire world is looking to America for enlightened leadership to peace
beyond any shadow of a doubt , that America will continue the fight for freedom
 to make complete victory certain , America will never become a party to any pl
nly in law and in justice . Here in America , we have labored long and hard to
```

Рисунок 2.21 — Використання concordance()

Зауважу, що concordance() вже ігнорує регістр, дозволяючи користувачам бачити контекст усіх варіантів регістру слова в порядку появи. Зауважу також, що ця функція не показує розташування кожного слова в тексті.

Крім того, оскільки concordance() лише виводить інформацію на консоль, він не ідеальний для маніпулювання даними. Щоб отримати придатний для використання список, який також дасть нам інформацію про місцезнаходження кожного випадку, використовується .concordance_list() на рисунку 2.22:

```
>>> concordance_list = text.concordance_list("america", lines=2)
>>> for entry in concordance_list:
...     print(entry.line)
...
 would want us to do . That is what America will do . So much blood has already
ay , the entire world is looking to America for enlightened leadership to peace
```

Рисунок 2.22 — Використання concordance()

concordance_list() дає нам список об’єктів ConcordanceLine, які містять інформацію про те, де зустрічається кожне слово, а також кілька інших властивостей, які варто вивчити. Список також відсортовано в порядку появи.

Сам клас nltk.Text має кілька інших цікавих особливостей. Одним із них є .vocab(), який варто згадати, оскільки він створює частотний розподіл для певного тексту.

Примірник тексту та супровідний розподіл частот з використання nltk.word_tokenize() та .vocab(), показано на рисунку 2.23:

```
>>> words: list[str] = nltk.word_tokenize(
...     """Beautiful is better than ugly.
...     Explicit is better than implicit.
...     Simple is better than complex."""
... )
>>> text = nltk.Text(words)
>>> fd = text.vocab()  # Equivalent to fd = nltk.FreqDist(words)
>>> fd.tabulate(3)
    is better   than
     3      3      3
```

Рисунок 2.23 — Використання nltk.word_tokenize() .vocab(),

.vocab() — це по суті ярлик для створення частотного розподілу з екземпляра nltk.Text. Таким чином, не потрібно робити окремий виклик, щоб створити новий об’єкт nltk.FreqDist.

Іншою потужною особливістю NLTK є його здатність швидко знаходити колакації за допомогою простих викликів функцій. Словосполучення — це ряд слів, які часто зустрічаються разом у певному тексті. Наприклад, у корпусі «Стан Союзу» слова «Сполучені Штати» та «США» дуже часто з’являються поруч. Ці два слова, що з’являються разом, є словосполученням. [17]

Словосполучення можуть складатися з двох і більше слів. NLTK надає класи для обробки кількох типів співрозміщень:

– Біграми: часті поєднання двох слів.
– Триграми: часті поєднання трьох слів.
– Квадрограми: часті комбінації з чотирьох слів.

NLTK надає спеціальні класи для пошуку сполучень у тексті. За зразком, який ви бачили досі, ці класи також побудовані зі списків слів на рисунку 2.24:

```
1 words = [w for w in nltk.corpus.state_union.words() if w.isalpha()]
2 finder = nltk.collocations.TrigramCollocationFinder.from_words(words)
```

Рисунок 2.24 — Використання TrigramCollocationFinder

Екземпляр TrigramCollocationFinder шукатиме саме триграми. NLTK також має класи BigramCollocationFinder і QuadgramCollocationFinder для біграм і квадраграм відповідно. Усі ці класи мають ряд утиліт для надання інформації про всі ідентифіковані колакації.

Одним із найкорисніших інструментів є властивість ngram_fd. Ця властивість містить частотний розподіл, побудований для кожного словосполучення, а не для окремих слів.

Використовуючи ngram_fd, можна знайти найпоширеніші словосполучення у наданому тексті на рисунку 2.25:

```
>>> finder.ngram_fd.most_common(2)
[(('the', 'United', 'States'), 294), (('the', 'American', 'people'), 185)]
>>> finder.ngram_fd.tabulate(2)
  ('the', 'United', 'States') ('the', 'American', 'people')
                          294                            185
```

Рисунок 2.25 — Використання ngram_fd

Навіть не потрібно створювати частотний розподіл, оскільки це вже властивість екземпляра пошуку колокацій.

2.2.2 Інструменти аналізу тональності

Аналіз тональності тексту можна виконувати за допомогою різних методів, включаючи, але не обмежуючись, позначення частин мови, лексичний аналіз, моделі машинного навчання та аналіз тону. Нижче описано, як працюють ці методи.

Позначення частин мови (POS): це процес позначення слова в тексті як відповідного певній частині мови (іменника, дієслова, прикметника тощо) на основі його визначення та контексту. Позначення тегами POS часто є попереднім етапом аналізу тексту та обробки природної мови, і воно може бути корисним для аналізу настроїв, надаючи більше контексту про те, як використовуються слова. Наприклад, слово «чудово» може бути прикметником в одному контексті (наприклад, «Я чудово провів час на вечірці») та іменником в іншому (наприклад, «Олександр Великий»).[21]

Лексичний аналіз: розглядає «лексичний» або словесний тон фрагмента тексту. Це часто робиться за допомогою словника настроїв або лексикону, де кожному слову присвоюється оцінка настрою (позитивна, негативна або нейтральна). Сентемент тексту можна визначити, склавши бали сентименту всіх слів. Цей підхід часто простий, але може не вдатися точно вловити настрої в більш складних текстах або текстах із нюансами.

Моделі машинного навчання: ці моделі можна навчити визначати почуття на основі позначених прикладів. Наприклад, модель машинного навчання можна

навчити на рецензіях на фільми, де кожна рецензія позначається як позитивна чи негативна. Модель вчиться асоціювати певні слова чи послідовності слів із позитивним чи негативним почуттям. Методи машинного навчання можуть бути більш точними, ніж лексичний аналіз, оскільки вони можуть навчитися розпізнавати більш складні моделі та враховувати контекст. Існують різні типи моделей машинного навчання, які можна використовувати, зокрема, але не обмежуючись ними, дерева рішень, випадкові ліси, SVM, наивний Байєс і моделі глибокого навчання, такі як рекурентні нейронні мережі (RNN) або трансформатори.[6]

Аналіз тону: аналіз тону, підгрупа аналізу настроїв, намагається зрозуміти емоційний тон слів. Він використовує аналіз тексту для визначення та розуміння настроїв у реченнях і потенційно може допомогти визначити такі тони, як радість, страх, смуток, гнів, аналітичний, впевнений, невпевнений тощо. Це може бути особливо корисним у таких контекстах, як обслуговування клієнтів, де розуміння Емоційний тон клієнта може допомогти краще вирішити його питання.[7]

Ефективність цих методів може бути різною залежно від характеру тексту та конкретного завдання. Їх також можна поєднувати різними способами. Наприклад, модель машинного навчання може використовувати теги частини мови як функції введення.

2.3 Приклади інструментів аналізу настроїв

Деякі приклади онлайн інструментів аналізу настроїв включають:

Google Cloud Natural Language APIGoogle – це хмарна служба обробки природної мови (NLP), яку пропонує Google. Вперше він був випущений у 2016 році та є частиною набору сервісів Google Cloud Platform. API розроблений, щоб надати розробникам інструменти для аналізу та розуміння тексту природною мовою в різноманітних форматах, включаючи звичайний текст, HTML і PDF.

Творцями Google Cloud Natural Language API є команда інженерів і спеціалістів із обробки даних Google, які розробили сервіс за допомогою

алгоритмів машинного навчання та методів NLP. Команда, що стоїть за API, має великий досвід у розробці інструментів NLP і працювала над різними пов’язаними проектами в Google, зокрема Google Translate і Google Search.

Google Cloud Natural Language API надає низку функцій для аналізу тексту природною мовою, зокрема аналіз настроїв, розпізнавання об’єктів, аналіз синтаксису та класифікацію вмісту. API розроблений таким чином, щоб бути простим у використанні, з простим інтерфейсом RESTful API, який дозволяє розробникам швидко інтегрувати функції NLP у свої програми.

Деякі з ключових функцій Google Cloud Natural Language API включають:

- Аналіз настрою: API може аналізувати текст, щоб визначити, чи має він позитивний, негативний чи нейтральний настрій.
- Розпізнавання сутностей: API може ідентифікувати та класифікувати сутності, згадані в тексті, наприклад людей, організації та місця розташування.
- Аналіз синтаксису: API може аналізувати граматичну структуру речень у тексті, зокрема ідентифікувати підмет, об’єкт і присудок.
- Класифікація вмісту: API може аналізувати текст, щоб визначити загальну тему, як-от спорт, політика чи розваги.

Google Cloud Natural Language API доступний як платний сервіс, вартість якого залежить від кількості запитів і обсягу оброблених даних. Існує також безкоштовна пробна версія для розробників, які хочуть протестувати API, перш ніж оформити платну підписку.

IBM Watson Natural Language Understanding — це хмарна служба обробки природної мови (NLP), яку пропонує IBM. Вперше він був випущений у 2017 році та є частиною набору когнітивних обчислювальних послуг IBM Watson. API розроблений, щоб надати розробникам інструменти для аналізу та розуміння тексту природною мовою в різноманітних форматах, включаючи звичайний текст, HTML і PDF.

Творці IBM Watson Natural Language Understanding — це команда інженерів і спеціалістів із обробки даних у IBM, які розробили цю послугу, використовуючи

комбінацію алгоритмів машинного навчання та підходів до NLP на основі правил. Команда, що стоїть за API, має великий досвід у розробці інструментів NLP і працювала над низкою пов'язаних проектів у IBM, включаючи служби Watson Assistant і Watson Discovery.

IBM Watson Natural Language Understanding надає низку функцій для аналізу тексту природною мовою, включаючи аналіз настроїв, розпізнавання сутностей, вилучення ключових слів та ідентифікацію понять. API розроблений таким чином, щоб бути простим у використанні, з простим інтерфейсом RESTful API, який дозволяє розробникам швидко інтегрувати функції NLP у свої програми.

Деякі з ключових функцій IBM Watson Natural Language Understanding включають аналіз настрою, розпізнавання сутностей, вилучення ключових слів, ідентифікація концепції.

IBM Watson Natural Language Understanding API доступний як платна послуга, вартість якої залежить від кількості запитів і обсягу оброблених даних. Існує також безкоштовна пробна версія для розробників, які хочуть протестувати API, перш ніж оформити платну підписку.

VADER (Valence Aware Dictionary and Sentiment Reasoner) — це заснований на правилах інструмент аналізу настроїв, який спеціально розроблений для аналізу даних соціальних мереж. Він був розроблений дослідниками з Університету Джорджії та доступний як інструмент із відкритим кодом.

VADER працює, аналізуючи настрої текстових даних на основі попередньо визначеного набору правил і евристик. Інструмент використовує лексикон слів і фраз із попередньо визначеними балами настрою, щоб ідентифікувати настрої в текстових даних. Лексикон включає понад 7500 лексичних функцій, які були спеціально вибрані, щоб вловити нюанси настроїв у даних соціальних мереж.

Окрім лексикону, VADER також використовує набір правил і евристик для підвищення точності аналізу настроїв. Наприклад, інструмент враховує наявність слів-заперечень, таких як «не» та «ніколи», які можуть змінити зміст речення.

VADER розроблений як швидкий і ефективний, з простим API, який дозволяє розробникам швидко інтегрувати функцію аналізу настроїв у свої програми. Інструмент може аналізувати настрої на рівні речень або документів і може виявляти настрої кількома різними мовами.

Однією з переваг VADER є його здатність обробляти унікальні характеристики даних соціальних мереж, такі як використання сленгу, абревіатур і смайлів. Інструмент також здатний фіксувати інтенсивність настрою, дозволяючи більш детально аналізувати текстові дані.

TextBlob — це бібліотека Python для обробки текстових даних, включаючи завдання обробки природної мови (NLP), такі як аналіз настроїв, додавання тегів до частин мови та вилучення фраз іменників. Це проект із відкритим вихідним кодом, розроблений розробниками з Каліфорнійського університету в Берклі, який розроблено таким чином, щоб він був простим у використанні та гнучким.

Однією з ключових особливостей TextBlob є його простота. Бібліотека надає простий, легкий у використанні API, який полегшує виконання стандартних завдань NLP. Наприклад, щоб виконати аналіз настрою фрагмента тексту, потрібно викликати конструктор TextBlob(), отримати доступ до властивості настрою, щоб отримати оцінку настрою та значення полярності.

На додаток до аналізу настроїв, TextBlob надає низку інших функцій NLP, включаючи тегування частин мови, виділення іменників і розпізнавання іменованих об’єктів. Бібліотека також включає підтримку мовного перекладу та перевірку орфографії.

TextBlob побудовано на основі Natural Language Toolkit (NLTK), яка є іншою популярною бібліотекою Python для NLP. Це означає, що TextBlob має доступ до широкого спектру ресурсів і інструментів NLP, включаючи корпуси, класифікатори та інші алгоритми NLP.

Однією з переваг TextBlob є його здатність обробляти широкий спектр текстових даних, включаючи необроблений текст, файли HTML і PDF. Бібліотека підтримує низку форматів файлів і може бути легко інтегрована з іншими бібліотеками та інструментами Python.

Загалом TextBlob — це потужна та гнучка бібліотека Python для виконання ряду завдань NLP. Його простий API і широкий набір функцій роблять його популярним вибором серед розробників і дослідників, які працюють з текстовими даними. [5]

SentiStrength — це інструмент аналізу настроїв, призначений для аналізу настроїв коротких текстів, таких як твіти, дописи у Facebook та онлайн-огляди. Він був розроблений Майком Телуолом, Кеваном Баклі та Георгіосом Палтоглу та доступний як настільна програма та веб-служба.

Однією з ключових особливостей SentiStrength є його здатність обробляти складні дані соціальних мереж. Інструмент використовує комбінацію лінгвістичних і статистичних методів для аналізу настроїв у коротких текстах. Він також враховує наявність смайлів, використання великих літер та інших текстових особливостей, які часто зустрічаються в даних соціальних мереж.

SentiStrength використовує подвійний підхід до аналізу настрою з окремими балами для позитивних і негативних настроїв. Інструмент також надає оцінку міцності, яка відображає інтенсивність настроїв у тексті.

SentiStrength розроблений, щоб бути гнучким і налаштовуватися. Користувачі можуть визначати власні лексикони та правила, щоб пристосувати аналіз настроїв до своїх конкретних потреб. Інструмент також включає підтримку кількох мов, що робить його цінним ресурсом для аналізу настроїв у багатомовних даних.

Однією з переваг SentiStrength є його швидкість і ефективність. Інструмент може швидко аналізувати великі обсяги даних, що робить його придатним для аналізу потоків соціальних мереж у режимі реального часу. Він також доступний як настільна програма, яка дозволяє користувачам аналізувати дані в автономному режимі.

Загалом SentiStrength — це потужний інструмент аналізу настроїв, спеціально розроблений для аналізу коротких текстів, таких як публікації в соціальних мережах і онлайн-огляди. Його гнучкий підхід, який можна налаштувати, у поєднанні зі швидкістю та ефективністю роблять його цінним

ресурсом для дослідників і компаній, які зацікавлені в розумінні настроїв у даних соціальних мереж.

2.4 Бібліотеки NLP

Natural Language Toolkit (NLTK) — це провідна платформа для створення програм на Python для обробки даних людської мови. Він надає прості у використанні інтерфейси для більш ніж 50 корпусів і лексичних ресурсів, таких як WordNet, а також набір бібліотек для обробки тексту для класифікації, токенізації, сформування основи, тегування, синтаксичного аналізу та семантичного міркування, оболонки для індустріальних бібліотек NLP, а також активну групу обговорення та список розсилки.[2]

NLTK — це «Набір інструментів природної мови», потужна бібліотека Python, яка робить роботу з текстом простою та цікавою. NLTK був розроблений Стівеном Бердом, Едвардом Лопером і Алексом Рубінштейном для задоволення власних дослідницьких потреб і використовується багатьма людьми, включаючи студентів, дослідників і розробників для обробки мовних даних.[4]

Він надає прості у використанні інтерфейси для більш ніж 50 корпусів і лексичних ресурсів, таких як WordNet, а також набір бібліотек для обробки тексту для класифікації, токенізації, сформування основи, тегування, синтаксичного аналізу та семантичного міркування, оболонки для індустріальних бібліотек NLP, і активний дискусійний форум.[3]

Hugging Face transformers — це платформа, яка надає спільноті API для доступу та використання найсучасніших попередньо навчених моделей, доступних у центрі Hugging Face.

Попередньо навчений трансформатор — це модель трансформатора, яка була навчена та підтверджена кимось (моральною людиною чи індустрією), і яку ми можемо використовувати як відправну точку для подібного завдання.

Hugging Face пропонує версії для спільноти та організації. Версія спільноти включає безкоштовну, яка має обмеження, і професійну версію, яка коштує 9 доларів на місяць. Організації, з іншого боку, включають лабораторні та

корпоративні рішення, які не є безкоштовними. Hugging Face надає API для майже 31 бібліотеки. Більшість із них є глибоким навчанням, наприклад Pytorch, Tensorflow, Jax, ONNX, fastai, Stable-Baseline 3 тощо.

Деякі з попередньо навчених моделей було навчено за допомогою багатомовних завдань усунення шуму в Javascript, Python, Rust і Bash/Shell. Цей курс із обробки природної мови Python допоможе вам отримати необхідні навички для успішного виконання поширених завдань з очищення тексту.

spaCy — це безкоштовна бібліотека з відкритим кодом для розширеної обробки природної мови (NLP) на Python.

spaCy розроблено спеціально для використання у виробництві та допомагає створювати програми, які обробляють і «розуміють» великі обсяги тексту. Його можна використовувати для створення систем вилучення інформації чи розуміння природної мови або для попередньої обробки тексту для глибокого навчання.

У той час як деякі функції spaCy працюють незалежно, інші вимагають завантаження навчених конвеєрів, які дозволяють spaCy передбачати лінгвістичні анотації – наприклад, чи є слово дієсловом чи іменником. Навчений конвеєр може складатися з кількох компонентів, які використовують статистичну модель, навчену на позначених даних. Наразі spaCy пропонує навчені конвеєри для різних мов, які можна встановити як окремі модулі Python. Пакети конвеєрів можуть відрізнятися розміром, швидкістю, використанням пам’яті, точністю та даними, які вони містять. Пакет, який користувач обирає, завжди залежить від випадку використання та текстів, з якими ви працюєте. Для випадку використання загального призначення невеликі стандартні пакети завжди є хорошим початком. Зазвичай вони включають такі компоненти:

Двійкові ваги для тегера частини мови, аналізатора залежностей і розпізнавання іменованих об’єктів для передбачення цих анотацій у контексті.

Лексичні записи в словнику, тобто слова та їхні незалежні від контексту атрибути, такі як форма чи написання.

Файли даних, такі як правила лематизації та таблиці пошуку.

Вектори слів, тобто багатовимірні представлення значень слів, які дозволяють визначити, наскільки вони схожі одне на одного.

Параметри конфігурації, як-от параметри мови й обробки конвеєра, а також реалізації моделі, щоб перевести spaCy у правильний стан під час завантаження конвеєра.

Fairseq — це набір інструментів моделювання послідовності, написаний на PyTorch, який дозволяє дослідникам і розробникам навчати спеціальні моделі для перекладу, резюмування, мовного моделювання та інших завдань генерації тексту.

Fairseq можна розширити за допомогою наданих користувачем плагінів. Вони підтримують п'ять видів плагінів:

Моделі визначають архітектуру нейронної мережі та інкапсулюють усі параметри, які можна вивчати.

Критерії обчислюють функцію втрат, враховуючи результати та цілі моделі.

Завдання зберігають словники та надають помічники для завантаження/ітерації наборів даних, ініціалізації моделі/критерію та обчислення втрат.

Оптимізатори оновлюють параметри моделі на основі градієнтів.

Планувальники швидкості навчання оновлюють швидкість навчання протягом курсу навчання.

2.5 7 Етапів роботи NLP

Етап 1: Сегментація речення

Сегментація речень є першим кроком у конвеєрі NLP. Він ділить увесь абзац на різні речення для кращого розуміння. Наприклад, «Лондон є столицею та найбільш густонаселеним містом Англії та Сполученого Королівства. Стоячи на річці Темза на південному сході острова Велика Британія, Лондон був великим поселенням протягом двох тисячоліть. Він був заснований римлянами , який назвав його Лондініум».

Після використання сегментації речень ми отримуємо такий результат:

«Лондон — столиця та найбільш густонаселене місто Англії та Сполученого Королівства».

«Лондон, що стоїть на річці Темзі на південному сході острова Великобританія, був великим поселенням протягом двох тисячоліть».

«Він був заснований римлянами, які назвали його Лондініум».[19]

Етап 2: Токенізація слів

Токенізація слів розбиває речення на окремі слова або лексеми. Це допомагає зрозуміти контекст тексту. Під час токенізації речення «Лондон є столицею та найбільш густонаселеним містом Англії та Сполученого Королівства» воно розбивається на окремі слова, тобто «Лондон», «є», «те», «столиця», «та», «найбільш», «населений», «місто», «з», «Англія», «і», «Великобританія», «Королівство», «.»Існує кілька різноманітних методів попередньої обробки даних, усі завдання можна розділити на кілька загальних, важливих кроків: очищення даних, інтеграція даних, зменшення даних і перетворення даних.[19]

Етап 3: Стемминг

Визначення коренів допомагає в попередній обробці тексту. Модель аналізує частини мови, щоб зрозуміти, про що саме йдеться в реченні.

Створення коренів нормалізує слова в їх основі або кореневій формі. Іншими словами, це допомагає передбачити частини мови для кожної лексеми. Наприклад парочка слів має корінь «intelligen». Однак в англійській мові немає такого слова, як «intelligen».

Етап 4: Лематизація

Лематизація видаляє флективні закінчення та повертає канонічну форму слова чи леми. Це схоже на коріння, за винятком того, що лема є справжнім словом. Наприклад, «грає» і «грав» є формами слова «грати». Отже, гра є лемою цих слів. На відміну від основи (згадаймо «intelligen»), «грати» є правильним словом. [20]

Етап 5: Аналіз стоп-слів

Наступний крок — розглянути важливість кожного слова в певному реченні. В англійській мові деякі слова зустрічаються частіше, ніж інші, наприклад «is», «a», «the», «and». Оскільки вони часто з'являються, конвеєр NLP позначає їх як стоп-слова. Вони відфільтровані, щоб зосередитися на більш важливих словах.

Крок 6: Розбір залежностей

Далі йде синтаксичний аналіз залежностей, який в основному використовується, щоб дізнатися, як усі слова в реченні пов'язані одне з одним. Щоб знайти залежність, ми можемо побудувати дерево та призначити одне слово як батьківське. Основне дієслово в реченні буде виконувати роль кореневого вузла.

Крок 7: Позначення частини мови (POS).

POS-теги містять дієслова, прислівники, іменники та прикметники, які допомагають граматично правильно вказати значення слів у реченні як показано на рисунку 2.26.

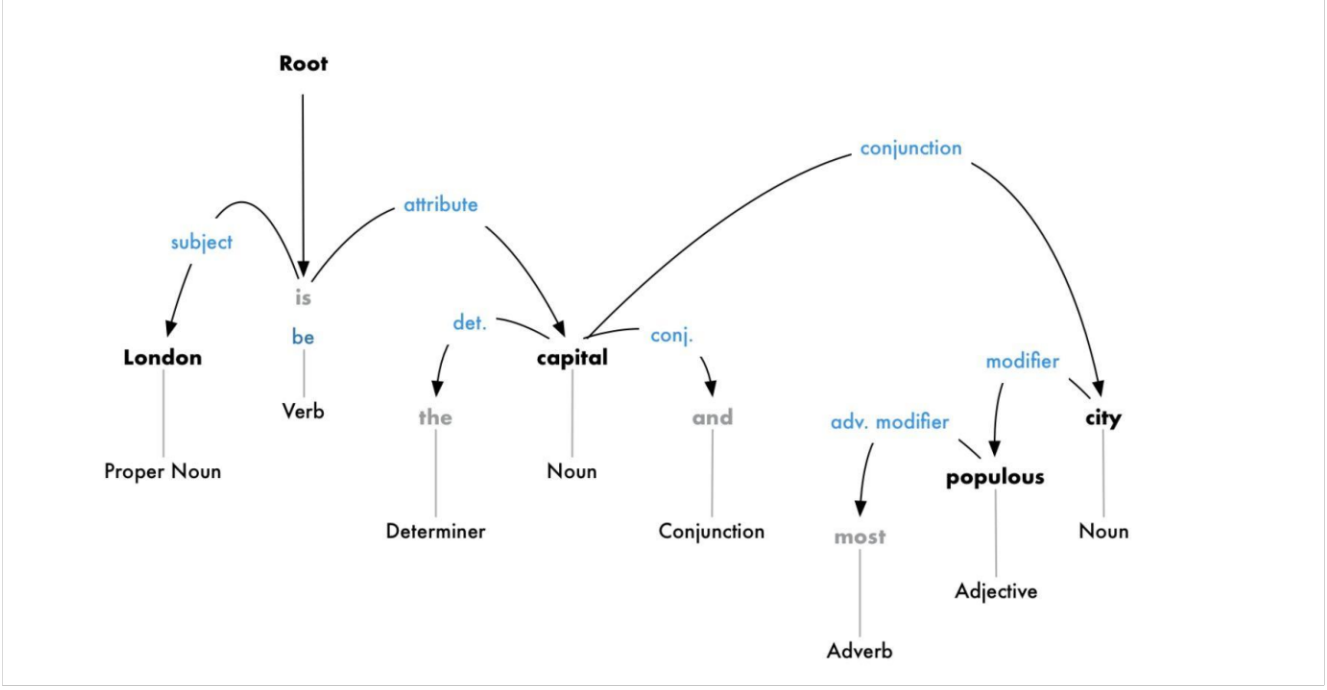

Рисунок 2.27 – Позначення частин мови

РОЗДІЛ 3
ПРАКТИЧНА ЧАСТИНА

3.1 Підготовка та встановлення бібліотек

Для виконання практичної частини була обрана мова програмування Python (3.11.3) та бібліотеки: NLTK (3.8.1), NUMPY (1.24.2) та Pandas (2.0.0) та scikit-learn 1.2.2.

Спершу для виконання практичної частини необхідно встановити мову програмування Python та необхідні бібліотеки. Використовуючи просту команду *pip install* встановити NLTK (як на рис. 3.1) , NUMPY, PANDAS та scikit-learn.

```
1 !pip install nltk
```

Рисунок 3.1 – Встановлення бібліотеки NLTK

Оскільки різні набори тексту (або корпусу) життєво важливі в комп'ютерній лінгвістиці, NLTK також надає доступ до багатьох із цих наборів, моделей і попередньо навчених утиліт.

Крім того, було імпортовано nltk і виконаний наступний крок, щоб завантажити та встановити деякі з більш популярних текстових модулів: stop-words, words, wordnet, punkt і набори даних, як-от огляди фільмів і твіттер як на рисунку 3.2, 3.3, 3.4.

```
>> import nltk
>> nltk.download('popular')
nltk_data] Downloading collection 'popular'
nltk_data]    |
nltk_data]    | Downloading package cmudict to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\cmudict.zip.
nltk_data]    | Downloading package gazetteers to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\gazetteers.zip.
nltk_data]    | Downloading package genesis to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\genesis.zip.
nltk_data]    | Downloading package gutenberg to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\gutenberg.zip.
nltk_data]    | Downloading package inaugural to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\inaugural.zip.
nltk_data]    | Downloading package movie_reviews to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\movie_reviews.zip.
nltk_data]    | Downloading package names to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\names.zip.
nltk_data]    | Downloading package shakespeare to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\shakespeare.zip.
nltk_data]    | Downloading package stopwords to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\stopwords.zip.
nltk_data]    | Downloading package treebank to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\treebank.zip.
nltk_data]    | Downloading package twitter_samples to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\twitter_samples.zip.
nltk_data]    | Downloading package omw to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    | Downloading package omw-1.4 to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    | Downloading package wordnet to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    | Downloading package wordnet2021 to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    | Downloading package wordnet31 to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    | Downloading package wordnet_ic to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\wordnet_ic.zip.
nltk_data]    | Downloading package words to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping corpora\words.zip.
nltk_data]    | Downloading package maxent_ne_chunker to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping chunkers\maxent_ne_chunker.zip.
nltk_data]    | Downloading package punkt to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
nltk_data]    |   Unzipping tokenizers\punkt.zip.
nltk_data]    | Downloading package snowball_data to
nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
```

```
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]    |   Unzipping taggers\averaged_perceptron_tagger.zip.
[nltk_data]    |
[nltk_data]  Done downloading collection popular
True
```

Рисунок 3.2– Використання комадни nltk.download(“popular”)

```
>>> nltk.download('movie_reviews')
[nltk_data] Downloading package movie_reviews to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
True
```

Рисунок 3.3 – Встановлення модуля Movie reviews

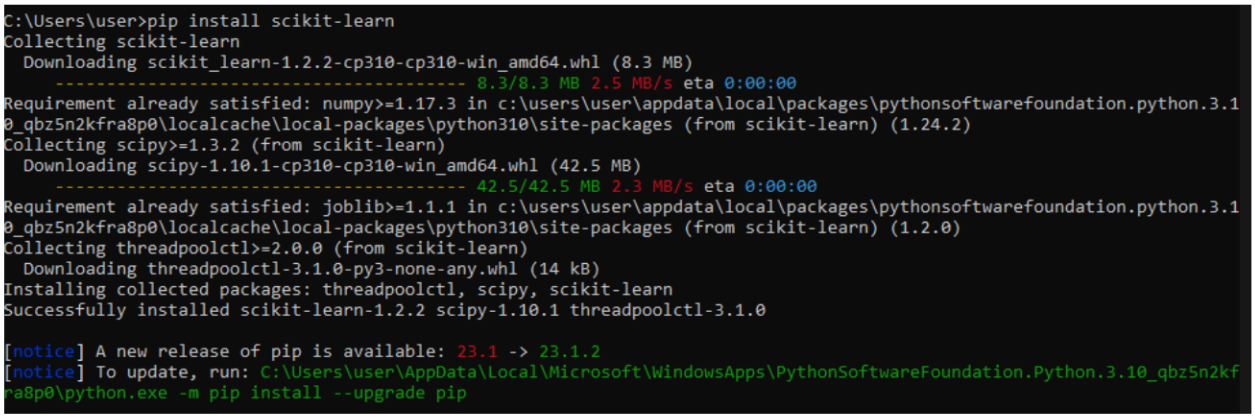

Рисунок 3.4 – Використання pip install scikit-learn

Було відфільтровано stop words тому що, якщо подивись на пакет stop words, то побачимо список слів з англійської мови, які необхідні для граматичної та семантичної структури кожного речення. Такі слова, як is, are, the, of, you, me, we. тощо, є невід'ємною частиною речення, але не обов'язково додають багато інформації до того, що речення намагається сказати. Якщо, наприклад, потрібно визначити, позитивне чи негативне речення, ці слова не допоможуть в завданні і, зрештою, лише непотрібно збільшать розміри даних.

Модуль англійських stop_words містить список із 179 поширених англійських слів як показано на рисунку 3.5.

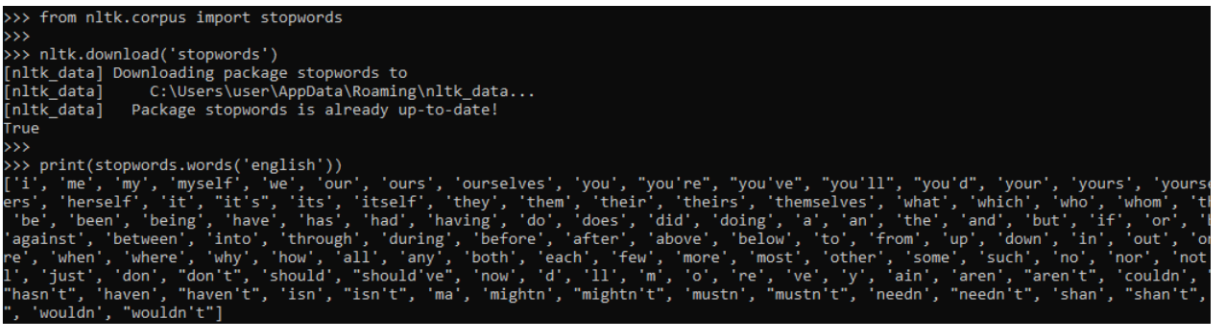

Рисунок 3.5 – Список слів модуля Stop words

Тепер цей список можливо використати, щоб відфільтрувати стоп-слова з даного речення як показано на рисунку 3.6. Будь-який текст, наданий як вхідні дані, потрібно токенізувати. Токенізація перетворює рядок тексту на список токенів (відокремлюючи всі слова, спеціальні символи тощо).

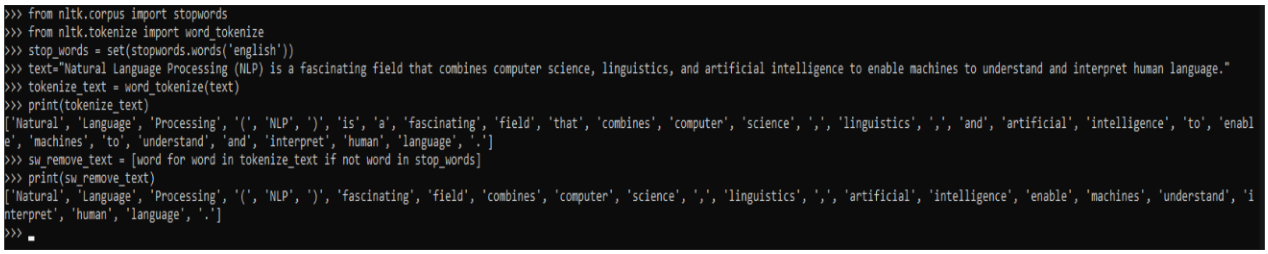

Рисунок 3.6 – Фільтрування Stop words

Класифікація тексту за допомогою NLTK

Оскільки використовується модуль NLTK, був використаний один із наборів даних, доступних безпосередньо з корпусу NLTK, для створення моделей. Було обрано використовувати набір даних Movie Reviews від NLTK. Цей набір даних містить майже 160 000 рецензій на фільми, кожна з яких позначена як позитивна чи негативна. Отже, наступним кроком буде binary text classification.

Класифікація бінарного тексту (binary text classification) — це завдання обробки природної мови (NLP), яке передбачає присвоєння однієї з двох можливих міток певному текстовому документу або введеним даних. Дві мітки, як правило, "позитивний" і "негативний", або "1" і "0", залежно від конкретного застосування. [15]

Було завантажено необхідний набір даних із NLTK, використовуючи nltk.corpus import movie_reviews. Далі скориставшися наведеним нижче кодом, було завантажено дані для обох міток у Pandas DataFrame. DataFrame Pandas — це двовимірна структура даних із мітками в бібліотеці Pandas Python. Це схоже на електронну таблицю або таблицю SQL, де дані організовані в рядки та стовпці. Фрейм даних Pandas можна створити з різних джерел, включаючи файли CSV, електронні таблиці Excel і бази даних SQL як показано на рисунку 3.7.

```
>>> import pandas as pd
>>> import nltk
>>>
>>> from nltk.corpus import movie_reviews
>>>
>>> df_dict = {'text': [], 'label': []}
>>>
>>> # Positive reviews
>>> for fileid in movie_reviews.fileids(categories='pos'):
...     df_dict['text'].append(list(movie_reviews.words(fileid)))
...     df_dict['label'].append(1)
...
>>> # Negative reviews
>>> for fileid in movie_reviews.fileids(categories='neg'):
...     df_dict['text'].append(list(movie_reviews.words(fileid)))
...     df_dict['label'].append(0)
...
>>> df = pd.DataFrame(df_dict)
>>>
```

Рисунок 3.7 – Створення DataFrame та класифікація бінарного тексту

Наступний крок полягає в використані технікі попередньої обробки даних, про яку було згадано вище, і використовуючи nltk, видалити stop words з тексту та провести лематизацію тексту (Лематизація тексту - це процес зведення слів до їх базової форми (леми) за допомогою морфологічного аналізу.) як показано на рисунку 3.8. Окрім NLTK, доступно багато бібліотек для обробки тексту, які можуть очищати дані та безпосередньо генерувати числові вектори.

```
>>> from nltk.corpus import stopwords
>>> from nltk.stem import WordNetLemmatizer
>>> import string
>>> import pandas as pd
>>>
>>> stop_words = set(stopwords.words('english'))
>>> string_punctuation = string.punctuation
>>> df['text'] = df['text'].apply(lambda x: [term for term in x if term not in stop_words])
>>> df['text'] = df['text'].apply(lambda x: [term for term in x if term not in string_punctuation])
>>> df['text'] = df['text'].apply(lambda x: [WordNetLemmatizer().lemmatize(w) for w in x])
>>> df.head()
                                                text  label
0  [film, adapted, comic, book, plenty, success, ...      1
1  [every, movie, come, along, suspect, studio, e...      1
2  [got, mail, work, alot, better, deserves, orde...      1
3  [jaw, rare, film, grab, attention, show, singl...      1
4  [moviemaking, lot, like, general, manager, nfl...      1
```

Рисунок 3.8 – Попередня обробка тексту

Цей код виконує попередню обробку тексту у DataFrame під назвою "df".

По-перше, він імпортує необхідні бібліотеки: "stopwords" і "WordNetLemmatizer" з модулів "nltk.corpus" і "nltk.stem" відповідно, а також модуля "string".

WordNetLemmatizer — це інструмент для лемматизації слів англійською мовою. WordNetLemmatizer є частиною інструментарію Natural Language Toolkit (NLTK), він використовує лексичну базу даних WordNet, яка є великим електронним словником, який групує англійські слова в набори синонімів, які називаються синсетами, і надає інформацію про їхні семантичні зв’язки.

Потім він створює набір англійських стоп-слів за допомогою функції "stopwords" і рядок, що містить усі знаки пунктуації за допомогою атрибута "punctuation" модуля "string".

Потім код застосовує три функції за допомогою методу "apply" стовпця "df['text']":

Перша функція видаляє всі стоп-слова з кожного списку слів у стовпці «текст» DataFrame.

Друга функція видаляє всі знаки пунктуації з кожного списку слів у стовпці «текст» DataFrame.

Третя функція лематизує кожне слово в кожному списку слів у стовпці «текст» DataFrame. Лематизація - це процес зведення слів до їх основи або кореневої форми.

Нарешті, код відображає кілька перших рядків модифікованого DataFrame за допомогою методу «head». Отриманий DataFrame має ту саму структуру, що й оригінальний DataFrame, але з текстовими даними, попередньо обробленими шляхом видалення стоп-слів і знаків пунктуації та лематизації решти слів.

У наведеному вище коді було видалено стоп-слова та знаки пунктуації з тексту та лематизували його за допомогою WordNet lemmatizer. NLTK Sentence Tokenizer непотрібний, оскільки завантажувач даних оглядів фільмів завантажував маркери безпосередньо з джерела. Далі було розглянуто весь словниковий запас набору даних і зроблено огляд найпоширеніших слів у ньому як показано на рисунку 3.9.

```
>>> all_text = '  '.join(['  '.join(s).translate(str.maketrans(' ', ' ', str
ext'].values]).split('  ')
>>> all_words = nltk.FreqDist(all_text)
```

Рисунок 3.9 – Розподіл слів за кількістю з'явлення в тексті

Цей код обробляє текстові дані, що зберігаються в стовпці «text » DataFrame під назвою «df», і створює частотний розподіл усіх слів, присутніх у тексті.

Метод «join» використовується для об’єднання всіх рядків у стовпці «text» у «df» в один рядок із двома пробілами (' ') між кожним рядком. Це створює довгий рядок, який містить усі текстові дані.

Потім використовується метод «translate», щоб видалити всю пунктуацію з цього довгого рядка. Метод str.maketrans використовується для створення таблиці перекладу, яка відображає всі знаки пунктуації на пробіли.

Потім використовується метод «strip», щоб видалити будь-які пробіли на початку або в кінці кожного рядка в довгому рядку.

Отриманий довгий рядок потім розбивається на список окремих слів, використовуючи два пробіли (' ') як роздільник. Це створює список усіх слів, присутніх у тексті.

Метод «FreqDist» з бібліотеки «nltk» потім використовується для створення частотного розподілу всіх слів у списку. Це підраховує, скільки разів кожне унікальне слово з’являється в тексті, і зберігає результати в об’єкті, схожому на словник як показано на рисунку 3.10.

```
>>> all_words.most_common(15)
[('film', 11055), ('movie', 6980), ('one', 6029), ('character', 3879), ('like', 3789), ('time', 2979), ('get', 2814)
, ('scene', 2671), ('make', 2634), ('even', 2568), ('good', 2429), ('story', 2346), ('would', 2110), ('much', 2050),
 ('also', 1967)]
>>>
```

Рисунок 3.10 – Список слів, які найчастіше зустрічаються у фреймі даних

Таким чином, було вибрано лише частину (але значну) унікальних слів у цьому словнику та з’ясовувано, чи зустрічається кожне з цих слів у огляді. Це створить розріджену закодовану матрицю, де стовпці представляють слова, а

значення є двійковими значеннями, що представляють присутність цього слова в огляді як показано на рисунку 3.11.

```
>>> word_features = [x[0] for x in all_words.most_common(1500)]
>>>
>>> def find_features(words):
...     features = {}
...     for word in word_features:
...         features[word] = (word in words)
...     return np.array(list(features.values()))
...
>>> feature_vector = np.array([find_features(s) for s in df['text'].values])
>>> y = df['label'].values
>>> print(feature_vector.shape, y.shape)
(2000, 1500) (2000,)
```

Рисунок 3.11 – Створення списку найпоширеніших слів

Вище наведений код визначає список під назвою "word_features", який містить 1500 найпоширеніших слів із колекції "all_words".

Він визначає функцію під назвою "find_features", яка приймає список слів як вхідні дані та повертає масив NumPy двійкових функцій, які вказують, чи присутнє кожне слово у "word_features" у вхідному списку слів. Отриманий вектор ознак має таку саму довжину, що й «word_features», і складається з 0 і 1.

Він визначає масив NumPy під назвою «feature_vector», який містить двійкові вектори ознак для кожного текстового запису в стовпці «text» у pandas DataFrame під назвою «df».

Він визначає масив NumPy під назвою «y», який містить мітки зі стовпця «label» DataFrame «df».

Він друкує фігури масивів "feature_vector" і "y" на консоль.

Наступним кроком є використання цієї інформацію про 1500 ознак, щоб навчити деякі моделі виконувати завдання класифікації.

3.2 Побудова моделі класифікації

Було використано scikit-learn, щоб створювати практичні моделі машинного навчання[16]. Було обрано декілька моделей, у тому числі k найближчих сусідів,

дерево рішень, випадковий ліс, логістичну регресію та машинні класифікатори опорних векторів. Також було виконано налаштування гіперпараметрів за допомогою GridSearchCV, щоб отримати найкращі моделі як показано на рисунку 3.12.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, train_test_split

classifiers = [KNeighborsClassifier(),
               DecisionTreeClassifier(),
               RandomForestClassifier(),
               LogisticRegression(),
               SGDClassifier(max_iter=100),
               SVC(kernel='linear')]

params_dict = [dict(n_neighbors=[5,10,25,35,50]),
               dict(max_depth=[5,10,15]),
               dict(n_estimators=[200,400,600,800], max_depth=[5,10]),
               dict(C=[0.05,0.1,0.5,1.0], penalty=['l2'], max_iter=[500,1000,1500]),
               dict(max_iter=[200,500,1000,1500]),
               dict(kernel=['linear', 'sigmoid', 'poly', 'rbf'])]
```

Рисунок 3.12 – Створеня класифікаторів для моделей

Цей код імпортує кілька класифікаторів із scikit-learn і створює список цих класифікаторів для подальшого використання в процесі вибору моделі. Включені класифікатори:

- KNeighborsClassifier()
- DecisionTreeClassifier()
- RandomForestClassifier()
- LogisticRegression()
- SGDClassifier(max_iter=100)
- SVC(kernel='linear')

Він також створює список словників під назвою params_dict. Кожен словник у цьому списку представляє набір гіперпараметрів, які будуть використовуватися під час налаштування відповідного класифікатора під час процесу вибору моделі. Словники в params_dict такі:

Гіперпараметрами для KNeighborsClassifier є «n_neighbors», це список значень для перевірки кількості сусідів для класифікації.

Гіперпараметрами для DecisionTreeClassifier є 'max_depth', який є списком значень для перевірки максимальної глибини дерева рішень.

Гіперпараметрами для RandomForestClassifier є 'n_estimators', який є списком значень для перевірки кількості дерев у лісі, і 'max_depth', який є списком значень для перевірки максимальної глибини кожного дерева рішень у лісі. .

Гіперпараметрами для LogisticRegression є «C», який є списком значень для перевірки сили регулярізації, «penalty», який має значення «l2», щоб вказати регулярізацію L2, і «max_iter», який є списком значень для тест на максимальну кількість ітерацій для збіжності.

Гіперпараметрами для SGDClassifier є 'max_iter', який є списком значень для перевірки максимальної кількості ітерацій для збіжності.

Гіперпараметрами для SVC є «ядро», яке є списком значень для перевірки типу ядра для використання у SVM.

Ці гіперпараметри використовуватимуться в поєднанні з GridSearchCV, функцією з модуля model_selection scikit-learn, для виконання пошуку в просторі гіперпараметрів для кожного класифікатора та пошуку найкращої комбінації гіперпараметрів, яка забезпечує найвищу продуктивність для даного набору даних як показано на рисунку 3.13.

```
>>> names = ['knn', 'decision tree', 'random forest', 'logistic regression', 'sgd classifier', 'svm']
>>>
>>> X_train, X_test, y_train, y_test = train_test_split(feature_vector, y, test_size=0.2, random_state=42)
>>>
>>> for name, params, model in zip(names, params_dict, classifiers):
...     clf = GridSearchCV(model, params)
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(f'{name} {clf.best_params_}: {np.mean(y_test==y_pred)}')
...
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [5, 10, 25, 35, 50]})
knn {'n_neighbors': 35}: 0.605
GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [5, 10, 15]})
decision tree {'max_depth': 10}: 0.6375
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'max_depth': [5, 10],
                         'n_estimators': [200, 400, 600, 800]})
random forest {'max_depth': 10, 'n_estimators': 600}: 0.7775
GridSearchCV(estimator=LogisticRegression(),
             param_grid={'C': [0.05, 0.1, 0.5, 1.0],
                         'max_iter': [500, 1000, 1500], 'penalty': ['l2']})
logistic regression {'C': 0.05, 'max_iter': 500, 'penalty': 'l2'}: 0.8025
GridSearchCV(estimator=SGDClassifier(max_iter=100),
             param_grid={'max_iter': [200, 500, 1000, 1500]})
sgd classifier {'max_iter': 200}: 0.795
GridSearchCV(estimator=SVC(kernel='linear'),
             param_grid={'kernel': ['linear', 'sigmoid', 'poly', 'rbf']})
svm {'kernel': 'rbf'}: 0.8
```

Рисунок 3.13 – Результати тестування точності моделей

Цей код спочатку визначає список імен, який містить імена класифікаторів, які були імпортовані та створені раніше в сценарії.

Потім він використовує функцію train_test_split від scikit-learn, щоб розділити вектор ознак feature_vector і мітки y на навчальні та тестові набори. Навчальний набір містить 80% даних, а тестовий набір містить решту 20%. Параметр random_state має значення 42, щоб забезпечити відтворюваність.

Далі код повторює список імен, а також створені раніше списки params_dict і класифікаторів, використовуючи функцію zip(), щоб згрупувати відповідні елементи трьох списків разом. Для кожної ітерації він створює об’єкт GridSearchCV під назвою clf, використовуючи відповідний класифікатор і гіперпараметри. Потім він підбирає об’єкт clf до навчальних даних за допомогою X_train і y_train. Після підгонки моделі він використовує метод прогнозування clf для прогнозування тестового набору, зберігаючи результати в y_pred. Нарешті, він друкує назву класифікатора, найкращі гіперпараметри, знайдені за допомогою пошуку сітки (clf.best_params_), і точність моделі на тестовому наборі (np.mean(y_test==y_pred)).

Загалом, цей код виконує налаштування гіперпараметрів і вибір моделі за допомогою пошуку в сітці, а потім оцінює найкращу модель, знайдену за допомогою тестового набору. Точність найкращої моделі на тестовому наборі друкується для кожного з класифікаторів, перелічених у назвах.

Класифікатор KNN з n_neighbors=35 має точність 0,605.

Класифікатор дерева рішень із max_depth=10 має точність 0,6375.

Класифікатор випадкового лісу з max_depth=10 і n_estimators=600 має точність 0,7775.

Класифікатор логістичної регресії з C=0,05, max_iter=500 і penalty='l2' має точність 0,8025.

Класифікатор SGD з max_iter=200 має точність 0,795.

Класифікатор SVM із kernel='rbf' має точність 0,8.

Що показує класифікатор логістичної регресії дає найкращі результати з точністю 80,25%, а SVM наближається до 80%.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи був проведений аналіз теми та вивчено методи аналізу тональності тексту машиним методами навчання та виконано практичне завдання.

Під час створення першого розділу кваліфікаційної роботи було дане комплексне розуміння аналізу тональності тексту, підкресливши його значення в аналізі текстових даних. Було обговорено обмеження та практичне застосування аналізу тексту, наголошуючи на необхідності використання методів машинного навчання та технік NLP для досягнення точних результатів.

Під час виконання наступного розділу кваліфікаційної роботи було розглянуто широкий спектр інструментів аналізу, включаючи провідні галузеві платформи, такі як IBM Watson, Google Cloud Language API, Amazon Comprehend і LIWC. Ці інструменти були ретельно оцінені, щоб визначити їх можливості та придатність для аналізу тональності тексту. Крім того, було розглянуто відомі бібліотеки NLP, а саме NLTK, трансформатори Hugging Face, spaCy та Fairseq, щоб зрозуміти їхні функції та їхній потенційний внесок у сферу аналізу тексту.

Отже, в результаті виконання кваліфікаційної роботі успішно розроблена модель аналізу тексту. Модель використовувала такі бібліотеки, як NLTK, NumPy, Pandas і scikit-learn, для попередньої обробки даних, вилучення відповідних функцій і впровадження різних класифікаторів. Завдяки використанню набору даних оглядів фільмів, доступного в NLTK, модель змогла точно класифікувати настрої.

Було використано й оцінено кілька класифікаторів, у тому числі K-найближчих сусідів (KNN), дерево рішень, випадковий ліс, логістичну регресію, стохастичний градієнтний спуск (SGD) і машину опорних векторів (SVM).

Розроблену модель можна використовувати для аналізу тексту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andreas C. Müller, Sarah Guido "Introduction to Machine Learning with Python" 2016 URL: https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/ (дата звернення 10.12.2022)

2. Офіційна документація NLTK URL: http://www.nltk.org/ (дата звернення 10.12.2022)

3. Natural Language Processing with Python (NLTK) by Steven Bird, Ewan Klein, and Edward Loper. URL: https://www.nltk.org/book/ (дата звернення 10.12.2022)

4. Sentiment Analysis with Python NLTK URL: https://stackabuse.com/sentiment-analysis-with-python-nltk/ (дата звернення 10.12.2022)

5. TextBlob: Simplified Text Processing URL: https://textblob.readthedocs.io/en/dev/ (дата звернення 13.12..2022)

6. A Comprehensive Guide to Understand and Implement Text Classification in Python URL: https://www.analyticsvidhya.com/blog/2021/06/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/ (дата звернення 13.12.2022)

7. A Complete Guide to Sentiment Analysis with Python URL: https://www.analyticsvidhya.com/blog/2021/06/a-complete-guide-to-sentiment-analysis-with-python/ (дата звернення 14.12.2022)

8. Python Text Processing with NLTK 2.0 Cookbook URL: https://www.packtpub.com/product/python-text-processing-with-nltk-2-0-cookbook/9781849513609 (дата звернення 12.12.2022)

9. Sentiment Analysis Using Python: A Comprehensive Guide URL: https://towardsdatascience.com/sentiment-analysis-using-python-a-comprehensive-guide-into-classification-algorithms-c67b8ce2c2fe (дата звернення 15.12.2022)

10. Introduction to Sentiment Analysis in Python URL: https://towardsdatascience.com/introduction-to-sentiment-analysis-in-python-7a4009f54c5f (дата звернення 20.12.2022)

11. A Comprehensive Guide to Text Classification with Python URL: https://

www.analyticsvidhya.com/blog/2021/05/a-comprehensive-guide-to-text-classification-with-python/ (дата звернення 22.12.2022)

12. An Introduction to Natural Language Processing with Python and NLTK URL: https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/ (дата звернення 25.12.2022)

13. Sentiment Analysis with NLTK URL: https://www.datacamp.com/community/tutorials/simplifying-sentiment-analysis-python (дата звернення 3.12.2022)

14. NLTK Sentiment Analysis Tutorial URL: https://pythonprogramming.net/sentiment-analysis-nltk-tutorial/ (дата звернення 7.12.2022)

15. Practical Text Classification With Python and Keras URL: https://realpython.com/python-keras-text-classification/ (дата звернення 12.12.2022)

16. Text Classification with Machine Learning in Python with Scikit-Learn URL: https://stackabuse.com/text-classification-with-machine-learning-in-python-with-scikit-learn/ (дата звернення 16.12.2022)

17. Using Machine Learning and Natural Language Processing Tools for Text Analysis URL: https://www.dataquest.io/blog/using-machine-learning-and-natural-language-processing-tools-for-text-analysis/ (дата звернення 18.12.2022)

18. Natural Language Processing – Concordance URL: https://avidml.wordpress.com/2017/08/05/natural-language-processing-concordance/ (дата звернення 24.12.2022)

19. What Is Text Analysis? URL: https://aws.amazon.com/what-is/text-analysis/?nc1=h_ls (дата звернення 24.12.2022)

20. Sentiment Analysis: First Steps With Python's NLTK Library URL: https://realpython.com/python-nltk-sentiment-analysis/ (дата звернення 22.02.2023)

21. Detecting subjectivity and tone with automated text analysis tools https://medium.com/pew-research-center-decoded/detecting-subjectivity-and-tone-with-automated-text-analysis-tools-5f0e662224b8 (дата звернення 20.02.2023)