НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НН Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Методи і моделі машинного навчання для текстової аналітики»

Виконав:

студент IV курсу, групи КА-95

Петренко Микола Миколайович _____

Керівник:

Ст. викладач, к.т.н. Савастьянов В.В. _____

Консультант з економічного розділу:

Доцент, к.е.н., Рощина Н.В. _____

Консультант з нормоконтролю:

К.ф.-м.н., Статкевич В.М. _____

Рецензент:

Провідний науковий співробітник відділу _____

прикладної інформатики Інституту телекомунікацій

і глобального інформаційного простору

НАН України, д.т.н. О. М. Терентьєв

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
НН Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» травня 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Петренкові Миколі Миколайовичу

1. Тема роботи «Методи і моделі машинного навчання для текстової аналітики», керівник роботи Савастьянов Володимир Володимирович, ст. викладач, к.т.н., затверджені наказом по університету від «30» травня 2023 р. № 20650-с

2. Термін подання студентом роботи 12.06.2023.

3. Вихідні дані до роботи

4. Зміст роботи – Аналіз задачі оцінки аспектів тексту. Методи рішення задачі оцінки аспектів тексту за допомогою нейромереж. Програмна реалізація та експериментальні результати.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) - презентація.

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
економічний	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання _____.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики (напряму) дослідження.	01.09.2022 – 30.09.2022	виконано
2	Аналіз актуальності задач стосовно тематики дослідження	01.10.2022 – 30.10.2022	виконано
3	Аналіз відомих результатів стосовно тематики дослідження	01.11.2022 – 30.11.2022	виконано
4	Формулювання задач дослідження	01.12.2022 – 30.12.2022	виконано
5	Уточнення теми дипломної роботи	21.02.2023	виконано
6	Збір статичних даних, попередній аналіз даних	01.03.2023 – 30.03.2023	виконано
7	Розробка програмного продукту для виконання обчислювальних експериментів	01.03.2023 – 30.04.2023	виконано
8	Виконання обчислювальних експериментів, аналіз та оформлення результатів	01.05.2023 – 20.05.2023	виконано
9	Оформлення пояснювальної записки у цілому	19.05.2023 – 25.05.2023	виконано
10	Підготовка презентації для захисту	25.05.2023 – 27.05.2023	виконано
11	Попередній захист дипломної роботи	28.05.2023 – 30.05.2023	виконано

Студент Микола ПЕТРЕНКО

Керівник Володимир САВАСТЬЯНОВ

РЕФЕРАТ

Дипломна робота: 105 с., 15 рис., 8 табл., 2 дод., 23 джерела.

МЕТОДИ І МОДЕЛІ МАШИННОГО НАВЧАННЯ ДЛЯ ТЕКСТОВОЇ АНАЛІТИКИ.

Темою роботи є використання моделей та методів машинного навчання в сфері текстової аналітики.

Об’єктом дослідження є аналіз настроїв за аспектами.

Предметом дослідження є модель латентної регресії аспектів та модель довгої-короткочасної пам'яті.

Метою даної роботи є порівняння показників класичних методів розв’язку задачі з методом латентної регресії аспектів.

Актуальність роботи пов’язана з збільшенням кількості текстів для обробки та розвитком машинного навчання.

В результаті роботи на мові Python було побудовано набір допоміжних класів для написання нейромереж довгої короткочасної пам'яті (ДКЧП) та латентної регресії аспектів (ЛРА), демонстраційно натренерованих для порівняння результатів аналізу аспектів на поданому наборі даних.

ABSTRACT

Bachelor's Thesis: 105 p., 15 fig., 8 tab., 8, 2 app., 23 sources.

METHODS AND MODELS OF MACHINE LEARNING FOR TEXT ANALYTICS.

The topic of the work is the use of models and methods of machine learning in the field of text analytics.

The object of the study is the analysis of ratings by aspects.

The subject of research is the model of latent regression of aspects and the model of long-short-term memory.

The purpose of this work is to compare the performance of classical methods of solving the problem with the method of latent regression of aspects.

The relevance of the work is related to the increase in the number of texts for processing and the development of machine learning.

As a result of the work in Python, a set of auxiliary classes was built for writing long short-term memory (LSTM) and latent aspect regression (LAR) neural networks, demonstratively trained to compare the results of aspect analysis on the given data set.

ЗМІСТ

ВСТУП

В наш час текст став невід'ємною частиною людського життя. Він повністю оточує нас, від книг та журналів до рекламних оголошень на вулицях та екранів мобільних телефонів та моніторів. Використовуючи текст ми не тільки комунікуємо між собою але й отримуємо нову інформацію, пізнаємо світ, тощо.

На початку існування людство не володіло писемністю взагалі. Відома легенда про перського царя Дарія I свідчить, що одного разу він отримав послання від скіфів. Послання включало чотири предмети: птаха, мишу, жабу і стріли. Гонець, що доставив послання, повідомив, що більше нічого повідомляти йому не веліли, і з тим розпрощався з царем. Постало питання, як же інтерпретувати це послання. Цар Дарій вирішив, що скіфи віддають себе в його владу і на знак покірності принесли йому землю, воду і небо, бо миша означає землю, жаба — воду, птах — небо, а стріли означають, що скіфи відмовляються від опору. Проте один з мудреців заперечив Дарію. Він тлумачив послання скіфів зовсім інакше: «Якщо ви, перси, як птахи, не відлетите в небо, або, як миші, не зариєтеся в землю, або, як жаби, не пострибаєте в болото, то не повернетеся, уражені цими стрілами». Як виявилося надалі, цей мудрець мав рацію [1].

Переказана легенда розкриває той факт, що спочатку люди намагалися передавати інформацію за допомогою різних предметів, та з самої легенди можна зробити висновок, що такий текст важко інтерпретувати. Надалі виникали різні види писемності, такі як піктографічне письмо, ієрогліфічне письмо та, нарешті, алфавітне письмо. Зупинимось на алфавітному письмі, адже воно найлегше для інтерпретації та аналізу.

Алфавітне письмо виникло в результаті додавання греками символів на позначення голосних звуків до фінікійського алфавіту через специфіку утворення форм слів. В результаті письмо перейшло на ще універсальніший

рівень. Тепер, використовуючи близько 30 знаків, які з легкістю могла вивчити будь-яка людина, можна було передати практично будь-які слова усної мови. Алфавітне письмо через свою простоту швидко розповсюдилося по всьому світу.

З моменту виникнення алфавітного письма по наш час назбиралась неймовірно велика кількість текстів (як рукописних так і друкованих). Таку кількість просто фізично неможливо прочитати та проаналізувати власноруч та на щастя розвиток методів машинного навчання та нейронних мереж допоможе справитись з цією задачею набагато швидше та простіше.

Машинне навчання — це підгалузь штучного інтелекту в галузі інформатики, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (тобто, поступово покращувати продуктивність у певній задачі) з даних, без того, щоби бути програмованими явно.

Тематика першого розділу присвячена опису текстової аналітики та роль методів та алгоритмів машинного навчання в ній.

Другий розділ присвячений безпосередньо методам рішення поставленої задачі.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення та ключові завдання текстової аналітики

Інтелектуальний аналіз тексту це напрям інтелектуального аналізу даних та штучного інтелекту, метою якого є отримання інформації з колекцій текстових документів, ґрунтуючись на застосуванні ефективних, у практичному плані, методів машинного навчання та обробки природної мови. Інтелектуальний аналіз тексту використовує всі ті ж підходи до перероблення інформації, що й інтелектуальний аналіз даних, однак різниця між цими напрямками проявляється лише в кінцевих методах, а також у тому, що інтелектуальний аналіз даних має справу зі сховищами та базами даних, а не електронними бібліотеками та корпусами текстів.

Термін текстова аналітика описує набір методів лінгвістики, статистики та машинного навчання, які моделюють і структурують інформаційний зміст текстових джерел для бізнес-аналітики, розвідувального аналізу даних, дослідження, або розслідування.

Термін «текстова аналітика» також описує застосування текстової аналітики для вирішення бізнес-проблем, незалежно чи в поєднанні з запитом і аналізом впорядкованих, числових даних.

До ключових завдань інтелектуального аналізу тексту відносяться:

- категоризація текстів;
- пошук інформації;
- обробка змін у колекціях текстів;
- розробка засобів представлення інформації для користувача.

1.2 Основні поняття та терміни

В контексті моделей машинного навчання, передбачення означає використання навченої моделі для встановлення відповідності між вхідними даними і вихідними результатами. Модель навчається на певному наборі даних, де вхідні приклади пов'язані з відповідними вихідними значеннями. Після тренування модель може використовуватися для передбачення вихідного значення для нових, раніше невідомих вхідних даних.

При передбаченні модель приймає нові вхідні дані і застосовує до них вивчену функцію або правило, яке було встановлено під час навчання. Результатом передбачення є оцінка або прогнозоване значення, що вказує на ймовірне відповідне вихідне значення.

Поняття аспекту відноситься до конкретної характеристики, або аспекту, який описується в тексті або відгуку. Аспект може бути будь-якою складовою частиною об'єкта або ситуації, яку ви хочете дослідити або проаналізувати.

Машинне навчання може використовуватись для виявлення та класифікації аспектів у тексті. Зазвичай для цього застосовуються методи обробки природної мови (Natural Language Processing, NLP), такі як аналіз тональності, заснований на машинному навчанні, або алгоритми виявлення ключових слів. Модель навчається розпізнавати та класифікувати слова або фрази, які вказують на певні аспекти, що важливі для аналізу.

1.3 Постановка задачі аналізу тексту

Аналіз тексту - це процес виявлення, розуміння та вивчення основних аспектів текстової інформації. Постановка задачі аналізу тексту полягає у визначенні конкретних цілей, які необхідно досягти під час аналізу тексту.

Постановка задачі аналізу тексту використовуючи алгоритми машинного навчання передбачає обробку текстових даних з метою отримання корисної інформації, класифікації тексту, виявлення патернів або здійснення інших видів аналізу.

Основні кроки для розв'язання задачі аналізу тексту з використанням алгоритмів машинного навчання.

1. Збір і підготовка даних: Зібрати текстові дані, які ви плануєте аналізувати, і очистити їх від зайвої інформації, такої як HTML-теги, спеціальні символи тощо. Дані можуть бути у вигляді документів, рядків або корпусу текстів.
2. Токенізація: Розбити текст на окремі слова або токени. Цей процес може включати подрібнення тексту на речення і подальшу розбивку речень на окремі слова або символи. Токенізація може бути виконана за допомогою простого розбиття за пробілами або більш складними методами, такими як використання регулярних виразів або бібліотек для обробки мови.
3. Векторизація: Перетворити текстові дані в числові вектори, які можуть бути зрозумілі алгоритмам машинного навчання. Існує кілька підходів до векторизації тексту, таких як мішок слів (bag-of-words), TF-IDF (term frequency-inverse document frequency) або використання нейромереж для отримання вкладень слів (word embeddings).
4. Вибір моделі машинного навчання: Вибрати підхід машинного навчання, який найкраще відповідає вашій задачі.

В цій роботі ми розглянемо базу даних з відгуками. До неї входить як оцінка, поставлена відвідувачем, так і сам коментар безпосередньо. Мета - встановити зв’язок за ключовими словами в коментарі між текстом та оцінкою, з метою подальшого передбачення оцінки за коментарем.

Позначимо нашу множину даних C = {(d, r_d)}. В цьому випадку d - коментар, сегментований в k аспектних сегментів, $c_i(w, d)$ - кількість слів w в аспектному сегменті i (може бути рівним нулю). Тоді будемо мати модель по передбаченню оцінки, базуючись на коментарі, сегментованому в k аспектних сегментів $p(r_d | d)$:

$r \sim N(\sum_{i=1}^{k} \alpha_i(d) r_i(d), \delta^2)$ (1.1) - загальна оцінка (тобто середнє зважене оцінок аспектів);

$r_i(d) = \sum_{w \in V} c_i(w, d) \beta_{i,w}$ (1.2) - оцінка аспекту (тобто сума ваг слів в аспекті).

Тут $\bar{\alpha}(d) \sim N(\bar{\mu}, \sum)$, де N (1.3) - багатоваріантність пріор Гауса, $\beta_{i,w} \in R$ - аспектно-специфічний настрій слова w.

1.4 Набір даних Trip Advisor

Для виконання роботи використано набір даних з оцінками та рецензіями на готелі від відомого сайту tripadvisor.com. TripAdvisor — це популярна онлайн-платформа, де користувачі можуть ділитися своїм досвідом, оцінками та думками про різні готелі, ресторани, пам’ятки тощо. Набір даних містить таку інформацію.

1. Інформація про готель: детальна інформація про готелі, які переглядаються, наприклад назва готелю, розташування, адреса та зручності.
2. Текст відгуку: Фактичний текст відгуків, написаний користувачами, які зупинялися в готелях. Ці відгуки можуть відрізнятися за довжиною та змістом, починаючи від кількох речень і закінчуючи детальним описом досвіду гостей.
3. Оцінки відгуків: числові оцінки, надані рецензентами для оцінки їх загального задоволення готелем. Оцінки зазвичай складаються за шкалою від 1 до 5 зірок, причому 5 є найвищою оцінкою, що означає відмінний сервіс і якість.
4. Дата відгуку: дата, коли користувач опублікував відгук.
5. Інформація про рецензента: інформація про рецензентів, як-от ім’я користувача чи відображуване ім’я, місцезнаходження та, можливо, інші демографічні дані, надані користувачем.

Дані для дослідження зібрано в єдиний файл reviews.json. JSON (JavaScript Object Notation) — це легкий формат обміну даними, який людям легко читати й писати, а машинам – аналізувати й генерувати. Він широко використовується для представлення структурованих даних і обміну інформацією між сервером і клієнтом або між різними системами.

Дані JSON відформатовано за допомогою пар ключ-значення та організовано в ієрархічній структурі. Основні правила синтаксису JSON такі:

- дані представлені у вигляді набору пар ключ-значення;
- дані взяті у фігурні дужки {};
- кожен ключ є рядком, після якого стоїть двокрапка;
- значеннями можуть бути рядки, числа, логічні значення, значення null, масиви або інші об’єкти JSON;
- рядкові значення беруться в подвійні лапки;
- числові значення можуть бути цілими чи числами з плаваючою комою;

- масиви — це впорядковані списки значень, укладених у квадратні дужки [], а значення розділені комами;
- вкладені об’єкти представлені шляхом розміщення об’єкта JSON в іншому об’єкті JSON.

1.5 Висновки до розділу 1

Текст став невід’ємною частиною людського життя, оточуючи нас у різних формах комунікації і надаючи нам нову інформацію. З часом назбиралась велика кількість текстів, яку людина не здатна прочитати і проаналізувати самостійно. Розвиток методів машинного навчання та нейронних мереж допомагає вирішувати цю задачу швидше і простіше.

Мета цієї роботи - розглянути один з прикладів використання машинного навчання в текстовій аналітиці, а саме - встановити зв’язок між текстом та оцінкою за аспектами, використовуючи ключові слова в коментарі з метою подальшого передбачення оцінки за коментарем.

2. МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Класичні підходи до розв’язання поставленої задачі

В роботі використано Python, оскільки він є одним із найпотужніших інструментів для виконання завдань із вивчення даних. Він пропонує безліч способів аналізу настроїв. Найпопулярніші з них перераховані тут:

- використання “Text Blob”;
- використання “VADER”;
- використання моделей на основі векторизації “Bag of Words”;
- використання моделей на основі “LSTM”;
- використання моделей на основі трансформаторів.

2.1.1 Використання “Text Blob”

TextBlob — це бібліотека Python (версій 2 і 3) для обробки текстових даних. Він надає простий API для занурення в загальні завдання обробки природної мови (NLP), такі як додавання тегів до частин мови, виділення іменників, аналіз настроїв, класифікація, переклад тощо.

Основні переваги бібліотеки:

- вилучення іменникового словосполучення;
- позначення частини мови;
- аналіз настроїв;
- класифікація (використовуючи Наївний Байєс, Дерево рішень);
- токенізація (поділ тексту на слова та речення);
- частоти слів і фраз;

- розбір;
- n-грами;
- зміна слова (множина та однина) та лематизація;
- виправлення орфографії;
- додавання нові моделі або мови за допомогою розширень;
- інтеграція “WordNet”.

2.1.2 Використання “VADER”

VADER (Valence Aware Dictionary and Sentiment Reasoner) — це лексикон і інструмент аналізу настроїв на основі правил, який спеціально адаптований до настроїв, виражених у соціальних мережах. Він використовує комбінацію лексикону почуття – це список лексичних особливостей (наприклад, слів), які зазвичай позначаються відповідно до їх семантичної орієнтації як позитивні або негативні. VADER не лише розповідає про оцінку позитивності та негативності, але також повідомляє нам про те, наскільки позитивним чи негативним є настрій.

2.1.3 Використання моделей на основі векторизації “Bag of Words”

“Bag of Words” — це техніка обробки природної мови для моделювання тексту. З технічної точки зору, можна сказати, що це метод вилучення ознак за допомогою текстових даних. Цей підхід є простим і гнучким способом вилучення функцій із документів.

“Bag of Words” — це представлення тексту, що описує наявність слів у документі. Відстежується лише кількість слів, ігноруються граматичні деталі

та порядок слів. Метод називається “мішком слів”, оскільки будь-яка інформація про порядок або структуру слів у документі відкидається. Модель стосується лише того, чи зустрічаються відомі слова в документі, а не того, де в документі.

Основна перевага цього методу полягає в тому, що використовуючи його, ми перетворюємо текст на еквівалентний вектор чисел, який в подальшому набагато простіше передавати в модель машинного навчання.

2.1.4 Використання моделей на основі “LSTM”

Моделі на основі LSTM (Long Short-Term Memory) є популярними в глибинному навчанні для обробки послідовних даних, таких як текст, звук або часові ряди. LSTM є варіантом рекурентної нейронної мережі (RNN), яка здатна зберігати та використовувати інформацію про довготривалі залежності в послідовностях.

Основна ідея за LSTM полягає у використанні спеціальних блоків пам'яті, які здатні забувати або запам'ятовувати інформацію на підставі поточного вхідного сигналу та попереднього стану. Це дозволяє їм ефективно працювати з послідовностями будь-якої довжини і уникати проблеми з витіканням градієнту, що виникає у стандартних RNN.

На Рис. 2.1 зображено типову архітектуру моделі на базі LSTM.

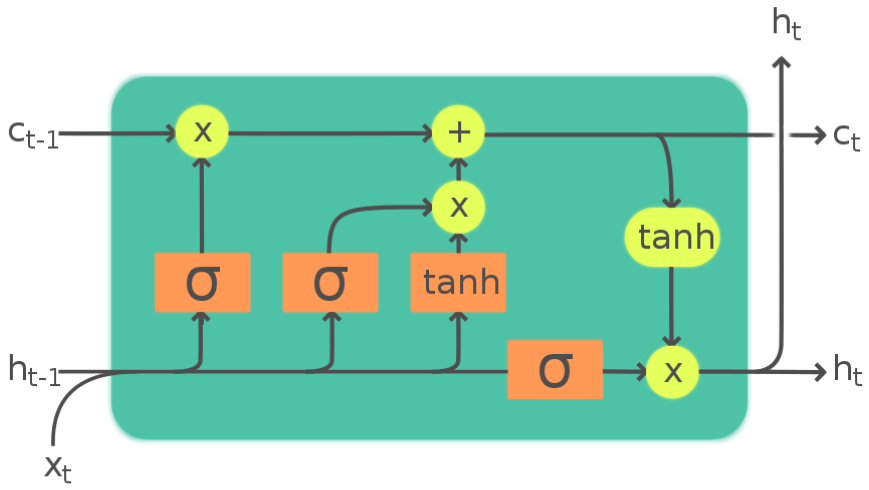


Рис. 2.1 - типова архітектура моделі на базі LSTM.

2.1.5 Використання моделей на основі трансформерів

Моделі на основі трансформерів є одними з найпотужніших та найефективніших моделей у сфері обробки природної мови. Вони використовують архітектуру трансформера, яка була вперше запропонована у роботі "Attention is All You Need" від Google Research у 2017 році.

Трансформери здатні моделювати довгострокові залежності між словами у тексті завдяки механізму уваги (attention). У звичайних рекурентних та згорткових нейронних мережах довгострокові залежності можуть бути складні для моделювання через обмежену "пам'ять" цих мереж. Трансформери уникають цього обмеження шляхом використання уваги, яка може "звертати увагу" на різні частини вхідного тексту при обчисленні ваги кожного слова у контексті.

2.2 Латентна аспектна регресія

Як противага до класичних методів рішення поставленої задачі використано алгоритм латентної аспектної регресії. В третьому розділі порівняно один з класичних підходів та цей підхід.

Латентна аспектна регресія (LAR) — це метод статистичного моделювання, який використовується в машинному навчанні та обробці природної мови (NLP) для виявлення прихованих аспектів у наборі спостережуваних змінних. LAR часто застосовується в таких завданнях, як аналіз настроїв, аналіз думок і системи рекомендацій.

Метою LAR є виявлення основних аспектів або вимірів, які впливають на дані спостереження. Зазвичай ці аспекти не можна спостерігати безпосередньо, але їх можна зробити з наявної інформації. Під час аналізу настроїв LAR може допомогти визначити основні аспекти або теми, які впливають на загальні настрої, виражені в тексті.

LAR — це форма неконтрольованого навчання, тобто для навчання не потрібні позначені дані. Натомість він покладається на дані спостереження, щоб дізнатися про приховані аспекти. Основна ідея LAR полягає в моделюванні спостережуваних змінних як функції прихованих аспектів та оцінці зв’язку між ними.

Ключові кроки регресії латентного аспекту зазвичай є такими.

1. Попередня обробка даних: це передбачає очищення та перетворення спостережуваних даних, таких як текстові документи чи оцінки користувачів, у відповідний формат для аналізу.
2. Вилучення прихованих аспектів: LAR має на меті виявити приховані аспекти або розміри, які присутні в даних. Для виділення аспектів можна використовувати різні методи, такі як прихований розподіл Діріхле (LDA) або факторізація невід’ємної

матриці (NMF). Ці методи розкладають спостережувані дані на набір прихованих аспектів.

3. Аспектна регресія: після виявлення прихованих аспектів LAR моделює зв’язок між спостережуваними змінними та прихованими аспектами. Для цієї мети можна використовувати регресійні моделі, такі як лінійна регресія або логістична регресія.
4. Оцінка моделі: продуктивність моделі LAR оцінюється за допомогою відповідних метрик оцінки залежно від конкретного завдання. Наприклад, під час аналізу настроїв такі показники, як точність, точність, запам’ятовування або оцінка F1, можна використовувати для оцінки здатності моделі передбачати настрої на основі прихованих аспектів.

LAR широко використовується в різних програмах, включаючи системи рекомендацій, аналіз тексту та аналіз відгуків клієнтів. Виявляючи приховані аспекти, LAR допомагає зрозуміти основні фактори, що впливають на спостережувані дані, і дає змогу точніше прогнозувати або давати рекомендації на основі цих аспектів.

2.3 Вибір програмних засобів

Розглянемо необхідні програмні засоби для виконання поставленої задачі.

2.3.1 Python

Python є потужним і гнучким інструментом для розробки моделей машинного навчання, який надає розробникам багато можливостей і зручних інструментів.

Використання Python для написання моделей машинного навчання має численні переваги.

1. Простота використання: Python має простий і зрозумілий синтаксис, що полегшує розробку і розуміння коду.
2. Багата екосистема: Python має широкий вибір бібліотек та фреймворків для машинного навчання, таких як TensorFlow, PyTorch, scikit-learn і Keras. Ці бібліотеки надають потужні засоби для побудови, навчання та оцінки моделей машинного навчання.
3. Швидкість розробки: Python є інтерпретованою мовою, що дозволяє розробникам швидко прототипувати та експериментувати з різними моделями. Python також надає велику кількість корисних інструментів, таких як Jupyter Notebook, які полегшують розробку і тестування моделей.
4. Навчання та документація: Для Python є багато доступних ресурсів для вивчення та документації.

2.3.2 Tensorflow

TensorFlow — це платформа машинного навчання з відкритим кодом, розроблена Google. Він надає комплексну екосистему інструментів, бібліотек і ресурсів для створення та розгортання моделей машинного навчання.

Ключові особливості TensorFlow.

1. Графік потоку даних: TensorFlow представляє обчислення як спрямований граф, де вузли представляють математичні операції, а ребра представляють потік даних між операціями. Цей підхід на основі графів забезпечує ефективне паралельне та розподілене обчислення.
2. Автоматична диференціація: TensorFlow автоматично обчислює градієнти для оптимізації моделей машинного навчання за допомогою таких методів, як зворотне поширення. Це дозволяє навчати складні моделі з мільйонами параметрів.
3. Гнучкість: TensorFlow підтримує різні мови програмування, включаючи Python. Він надає високорівневі API для швидкого створення моделей, а також низькорівневі API для детального контролю та налаштування.
4. Масштабованість: TensorFlow може масштабувати обчислення на декількох процесорах, графічних процесорах або навіть розподілених кластерах. Це дає змогу навчатися та робити висновки на основі великих наборів даних або моделей, які потребують значних обчислювальних ресурсів.

2.3.3 Numpy

NumPy — потужна бібліотека Python, яка використовується для чисельних обчислень. Вона забезпечує підтримку великих багатовимірних масивів і матриць разом із набором математичних функцій для ефективної роботи з цими масивами.

Ключові особливості NumPy.

1. Тип даних ndarray: основним об’єктом NumPy є ndarray (n-вимірний масив), який є швидким і ефективним контейнером

для великих наборів даних. Він забезпечує однорідний масив фіксованого розміру, який підтримує поелементні операції, нарізання та індексування.

2. Математичні функції: NumPy пропонує широкий спектр математичних функцій, включаючи тригонометричні функції, експоненціальні та логарифмічні функції, статистичні функції тощо. Ці функції можна поелементно застосовувати до масивів, забезпечуючи ефективні обчислення.
3. Операції з масивами: NumPy дозволяє виконувати ефективні операції з масивами, такі як поелементна арифметика, трансляція, операції лінійної алгебри, зміна форми, конкатенація, розбиття та сортування. Ці операції оптимізовано для підвищення продуктивності та можуть виконуватися на великих наборах даних.
4. Трансляція: трансляція — це потужна функція NumPy, яка дозволяє виконувати операції між масивами різних форм і розмірів. Це усуває потребу в явному циклі над масивами, що призводить до більш лаконічного та ефективного коду.
5. Інтеграція з іншими бібліотеками: NumPy добре інтегрується з іншими бібліотеками наукових і числових обчислень у Python.

2.3.4 Keras

Keras — популярна платформа глибокого навчання з відкритим вихідним кодом, яка надає високорівневий API для створення та навчання нейронних мереж.

Однією з ключових особливостей Keras є його здатність працювати поверх інших фреймворків глибокого навчання, таких як TensorFlow. Це

дозволяє користуватися основними обчислювальними можливостями цих фреймворків, водночас спрощеним та інтуїтивно зрозумілим інтерфейсом Keras.

Keras пропонує широкий спектр попередньо створених шарів, для повноцінних нейронних мереж. Ці рівні можна легко комбінувати для створення складних мережевих архітектур. Крім того, Keras надає різні алгоритми оптимізації, функції втрат і показники оцінки для полегшення навчання та оцінки моделей.

API Keras дотримується стилю послідовної або функціональної моделі. У послідовній моделі ви можете послідовно складати шари для створення моделі, тоді як у функціональній моделі ви можете створювати складніші мережеві архітектури з кількома входами та виходами. Keras також підтримує розширені методики, такі як трансферне навчання, де попередньо навчені моделі можна використовувати як відправну точку для нових завдань.

2.3.5 Matplotlib

Matplotlib — популярна бібліотека графічних зображень у Python, яка надає широкий спектр інструментів для створення статичних, анімованих та інтерактивних візуалізацій.

Matplotlib надає гнучкий і комплексний API для створення різних типів графіків, включаючи лінійні діаграми, точкові діаграми, стовпчасті діаграми, гістограми, секторні діаграми та багато іншого. Він також підтримує параметри налаштування для точного налаштування зовнішнього вигляду графіків, наприклад налаштування кольорів, додавання міток і заголовків, встановлення меж осі та додавання легенд.

Matplotlib пропонує широкий набір функціональних можливостей, таких як додавання кількох графіків до однієї фігури, створення підсхем, робота з різними стилями графіків, збереження графіків у файлах тощо.

2.3.6 Tensorboard

TensorBoard — це веб-інструмент візуалізації, наданий TensorFlow, системою машинного навчання з відкритим кодом, розробленою Google. Він використовується для перевірки та аналізу моделей машинного навчання, прогресу їх навчання та різноманітних показників, пов’язаних із ними. TensorBoard дозволяє відстежувати та налагоджувати моделі, візуалізувати дані та отримувати уявлення про їх поведінку.

Функціонал TensorBoard включає такі риси.

1. Візуалізація скалярних значень: може побудувати скалярні показники, такі як втрати, точність та інші користувацькі показники, протягом певного часу, щоб зрозуміти, як модель працює під час навчання.
2. Перегляд графіків моделі: TensorBoard дозволяє візуалізувати обчислювальний графік моделі, який показує потік даних і операцій. Це допомагає зрозуміти структуру моделі та усунути можливі проблеми.
3. Відстеження гістограм: можливість досліджувати розподіл значень для змінних і тензорів у моделі з часом. Це корисно для розуміння того, як значення змінюються під час навчання, і може допомогти виявити такі проблеми, як зникнення або зростання градієнтів.
4. Візуалізація зображень і вбудовування: TensorBoard дозволяє відображати зображення, візуалізувати вбудовування та

досліджувати багатовимірні дані за допомогою таких інструментів, як проектор для вбудовування.

5. Відстеження обчислювальних ресурсів: можливість під час навчання відстежувати та аналізувати використання обчислювальних ресурсів, таких як ЦП, ГП і пам’ять, щоб оптимізувати продуктивність і виявити вузькі місця.

2.3.7 NLTK

NLTK (Natural Language Toolkit) — це популярна бібліотека з відкритим кодом для обробки природної мови (NLP) на Python. Вона надає широкий спектр інструментів і ресурсів для таких завдань, як токенізація, формування коренів, лематизація, тегування частин мови, синтаксичний аналіз тощо.

NLTK пропонує набір бібліотек і корпусів обробки тексту (великі колекції письмових текстів), які можна використовувати для виконання різноманітних завдань обробки природної мови. Він підтримує понад 50 корпусів і лексичних ресурсів, у тому числі WordNet, лексичну базу даних, яка широко використовується для визначення значення англійських слів і семантичних зв’язків.

NLTK надає прості у використанні інтерфейси для таких завдань, як тегування частин мови, розпізнавання іменованих об’єктів, аналіз настроїв, машинний переклад і моделювання теми. Він також включає різні алгоритми та моделі, які можна використовувати для побудови та навчання моделей обробки природної мови.

2.4 Архітектура нейронних мереж

В роботі використано нейронну мережу на базі LSTM. Розглянемо ключові елементи архітектури таких нейронних мереж.

2.4.1 Вхідний рівень (input layer)

Вхідний рівень нейронної мережі - це початковий рівень, який приймає вхідні дані. Він відповідає за прийняття необроблених вхідних даних і передачу їх наступним шарам для обробки.

Кількість вузлів або нейронів у вхідному шарі визначається розмірністю вхідних даних. Кожен вузол у вхідному шарі представляє функцію або атрибут вхідних даних.

Значення вхідних вузлів можуть бути двійковими (0 або 1), безперервними значеннями або будь-яким іншим відповідним представленням залежно від характеру проблеми та етапів попередньої обробки, застосованих до даних.

Вхідний рівень не виконує жодних обчислень, але діє як канал для передачі вхідних даних на наступні рівні. Значення з вхідного рівня множаться на ваги та поширюються через мережу, зрештою приводячи до виходу мережі.

Варто зазначити, що вхідний рівень не має пов’язаної з ним функції активації, оскільки він служить лише для проходження вхідних даних. Функції активації зазвичай застосовуються в прихованих шарах і вихідному шарі нейронної мережі.

Таким чином, вхідний рівень — це початковий рівень нейронної мережі, який отримує вхідні дані та передає їх на наступні рівні для обробки.

Він представляє функції або атрибути вхідних даних і не виконує жодних обчислень і не має пов’язаних функцій активації.

2.4.2 Рівень вбудовування (embedding layer)

Рівень вбудовування — це рівень, який відображає категоричні змінні або дискретні вхідні дані в безперервне векторне представлення, також відоме як вбудовування. Він зазвичай використовується в завданнях обробки природної мови (NLP) для перетворення слів або токенів у щільні векторні представлення, які фіксують їх семантичне значення.

Рівень вбудовування зазвичай є першим шаром у моделі NLP, за яким йдуть інші рівні, такі як рекурентні нейронні мережі (RNN) або згорткові нейронні мережі (CNN). Рівень вивчає вбудовування під час процесу навчання, регулюючи вагові коефіцієнти на основі цільової функції завдання, наприклад мінімізації втрат або максимізації точності.

Рівень вбудовування можна ініціалізувати випадковим чином або за допомогою попередньо навчених вбудовувань, отриманих із великих текстових корпусів за допомогою таких методів, як word2vec, GloVe або fastText. Попередньо навчені вбудовування є корисними, коли навчальні дані для конкретного завдання обмежені, оскільки вони забезпечують відправну точку з уже отриманими значущими представленнями.

Розмір шару вбудовування визначається бажаною розмірністю векторів вбудовування. Зазвичай експериментують з різними параметрами, щоб знайти оптимальний баланс між збором достатньої інформації та керуванням складністю моделі. Розмірність векторів вбудовування є гіпер параметром, який необхідно налаштувати в процесі розробки моделі.

Під час прямого проходу рівень вбудовування приймає вхідні дані, які можуть бути послідовністю слів або токенів, і шукає відповідні вбудовування

для кожного вхідного елемента. Потім ці вбудовування передаються на наступні рівні для подальшої обробки або прогнозування. Вбудовування фіксують зв’язки та схожість між вхідними елементами, дозволяючи мережі вивчати значущі представлення для поточного завдання.

Таким чином, рівень вбудовування є важливим компонентом нейронних мереж для завдань NLP, забезпечуючи спосіб представлення дискретних вхідних даних у вигляді безперервних векторів, дозволяючи моделі ефективно вивчати та узагальнювати вхідні дані.

2.4.3 Двонаправлений рівень (bidirectional layer)

Двонаправлений рівень у нейронній мережі — це тип шару, який містить інформацію з минулих і майбутніх часових кроків або позицій. Він зазвичай використовується в задачах моделювання послідовності, таких як обробка природної мови (NLP) і розпізнавання мовлення.

У стандартній рекурентній нейронній мережі (RNN) інформація переходить від попереднього кроку до поточного. Цей односпрямований потік обмежує здатність мережі враховувати майбутній контекст під час прогнозування. Двонаправлені рівні усувають це обмеження, вводячи додатковий набір прихованих блоків, які обробляють вхідну послідовність у зворотному порядку, дозволяючи мережі отримувати інформацію з минулих і майбутніх часових кроків.

Основна ідея двонаправленого рівня полягає в тому, щоб поєднати дві окремі RNN: одну, яка обробляє вхідну послідовність у прямому напрямку (від початку до кінця), і іншу, яка обробляє її у зворотному напрямку (від кінця до початку). Приховані стани цих двох RNN потім об’єднуються або комбінуються певним чином, щоб забезпечити представлення, яке включає як минулий, так і майбутній контекст.

Під час фази навчання двонаправлений рівень зазвичай навчається за допомогою варіанту алгоритму зворотного поширення, що називається зворотним поширенням у часі (ВРТТ). ВРТТ враховує градієнти від прямого та зворотного проходів двонаправленого шару для оновлення параметрів моделі.

Двонаправлені рівні довели свою ефективність у різних завданнях, особливо в сценаріях, де важливий контекст минулих і майбутніх часових кроків.

Варто зазначити, що двонаправлені рівні вносять додаткову обчислювальну складність.

2.4.4 Щільний рівень (dense layer)

Щільний шар (також відомий як повністю зв’язаний шар) — це тип шару, де кожен нейрон або вузол шару з’єднаний з кожним нейроном попереднього шару. Це найпростіший і часто використовуваний тип рівня в багатьох архітектурах нейронних мереж.

У щільному шарі кожен нейрон отримує вхідні дані від усіх нейронів попереднього шару та застосовує до цих вхідних даних набір вагових коефіцієнтів, після чого виконується функція активації. Вихідні дані кожного нейрона в щільному шарі потім передаються як вхідні дані до нейронів наступного шару.

Метою щільного шару є вивчення складних моделей і зв’язків у даних шляхом коригування вагових коефіцієнтів під час процесу навчання. Вагові коефіцієнти визначають силу та важливість кожного вхідного сигналу, а функція активації вносить нелінійність у модель, дозволяючи їй фіксувати більш складні моделі.

Кількість нейронів у щільному шарі визначає розмірність вихідного простору. Наприклад, якщо у вас є щільний шар із 100 нейронами, вихід цього шару буде вектором довжини 100. Кількість нейронів у вхідному шарі та наступних щільних шарах залежить від конкретної проблеми та складності даних.

Щільні шари зазвичай використовуються всередині або наприкінці архітектури нейронної мережі, розташовані між вхідним і вихідним шарами або іншими типами шарів, такими як згорткові шари або повторювані шари. Розташування та кількість щільних шарів у нейронній мережі залежать від конкретної проблеми, що розглядається, і можуть бути налаштовані на основі бажаної складності та продуктивності моделі.

2.4.5 LSTM рівень (LSTM layer)

Рівень LSTM (довго короткочасна пам’ять) — це тип рекурентної нейронної мережі (RNN), який зазвичай використовується в моделях глибокого навчання, зокрема для завдань, пов’язаних із послідовними або часовими рядами даних. Рівні LSTM розроблені для усунення обмежень стандартних мереж RNN, яким важко охопити довготривалі залежності в послідовних даних.

Ключовою особливістю рівня LSTM є його комірка пам’яті, яка дозволяє мережі вибірково запам’ятовувати або забувати інформацію з часом. Комірка пам'яті підтримує внутрішній стан, який оновлюється на кожному кроці часу на основі введення та попереднього стану.

Рівень LSTM зазвичай складається з кількох компонентів:

1. Стан клітини: представляє довготривалу пам’ять LSTM. Він діє як конвеєр, дозволяючи інформації протікати вздовж

послідовності даних. Він оновлюється за допомогою комбінації вентилів забуття, введення та виведення.

2. Прихований стан: служить короткочасною пам’яттю LSTM. Він відповідає за захоплення та поширення відповідної інформації в межах послідовності. Прихований стан є похідним від стану комірки та виводиться на кожному кроці часу.
3. Забутий шлюз: визначає, яку інформацію слід відкинути зі стану комірки. Він приймає вхідні дані та попередній прихований стан як вхідні дані та виводить значення від 0 до 1 для кожного елемента стану комірки. Значення 1 вказує на те, що інформацію потрібно зберегти, тоді як 0 означає її забуття.
4. Вхідний шлюз: визначає, яка нова інформація повинна зберігатися в стані комірки. Він обчислює нове значення кандидата на основі вхідних даних і попереднього прихованого стану, а потім поєднує його з виходом вхідного шлюза для оновлення стану комірки.
5. Вихідний вентиль: контролює кількість інформації, яка виводиться на кожному кроці часу. Він вирішує, які частини стану комірки слід використовувати для обчислення прихованого стану. Вихідний вентиль приймає вхід і попередній прихований стан як вхідні дані та застосовує сигмоїдну функцію для визначення інформації, яка має бути виведена.

Під час процесу навчання рівень LSTM вчиться оптимізувати вагові коефіцієнти та зміщення, пов’язані з цими воротами та станом комірки за допомогою зворотного поширення та градієнтного спуску, що дозволяє йому фіксувати відповідні шаблони та залежності у вхідних даних.

2.4.5 Рівень кодувальника(encoder layer)

Основне призначення кодувальника полягає в перетворенні вхідних даних у представлення нижчих розмірів, яке часто називають латентним простором або вбудовуванням. Цей процес включає в себе захоплення суттєвих характеристик або шаблонів у вхідних даних і їх ущільнення у стисненому представленні. Рівні кодувальника відповідають за виконання цього перетворення.

Рівень кодера зазвичай складається з кількох складених одиниць кодування, також відомих як шари кодування або блоки кодування. Кожен блок кодування зазвичай складається з кількох підкомпонентів, таких як згорткові шари, повторювані шари (такі як LSTM або GRU) або повністю зв’язані шари. Ці підкомпоненти витягують ієрархічні та абстрактні характеристики з вхідних даних.

Рівень кодера приймає вхідні дані та застосовує різні математичні операції, такі як множення матриць, нелінійні активації (наприклад, ReLU або сигмоїда), операції об’єднання та методи нормалізації (наприклад, пакетна нормалізація). Ці операції призначені для захоплення та перетворення вхідних даних у представлення нижчих розмірів.

Поступово застосовуючи ці операції до кількох одиниць кодування, рівень кодувальника зменшує розмірність даних, що допомагає зменшити шум, отримувати важливу інформацію та вивчати більш компактні представлення. Остаточним виходом рівня кодера є стиснене представлення, яке потім можна використовувати для різноманітних наступних завдань, таких як декодування, класифікація або генерація вихідних послідовностей.

2.4.6 Рівень декодера (decoder layer)

Рівень декодера є ключовим компонентом архітектури, відомої як автокодер або модель послідовності до послідовності. Рівень декодера виконує операцію, зворотну до рівня кодера, реконструюючи вхідні дані з представлення нижчої розмірності або генеруючи вихідні послідовності на основі даного вхідного сигналу.

Рівень декодера зазвичай складається з одного або кількох прихованих шарів, за якими слідує вихідний рівень. Кожен прихований шар може використовувати різні функції активації, такі як випрямлені лінійні одиниці (ReLU), гіперболічний тангенс (tanh) або сигмовид. Ці функції активації вводять нелінійність у мережу, дозволяючи їй фіксувати складні зв’язки в даних.

У моделях послідовності до послідовності рівень декодера відповідає за генерацію вихідних послідовностей на основі вхідної послідовності. Він отримує остаточний прихований стан від рівня кодера та використовує його як початковий стан. Рівень декодера може використовувати рекурентні нейронні мережі (RNN), такі як довготривала короткочасна пам’ять (LSTM) або Gated Recurrent Unit (GRU), для захоплення послідовних залежностей і генерування виходу крок за кроком. Механізми уваги часто включені, щоб допомогти декодеру зосередитися на відповідних частинах вхідної послідовності під час процесу генерації.

2.5 Висновки до розділу 2

На даний час вже існують класичні підходи до розв’язання поставленої задачі. В наступному розділі вони будуть розглянуті більш детально з метою оцінки їх показників в роботі з текстом.

Всі перераховані інструменти роботи, а саме мова програмування Python та її бібліотеки будуть використані для побудови та оцінки моделей машинного навчання за методами, перерахованими в розділі 2.1.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ, АНАЛІЗ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТУ

3.1 Ознайомлення з набором даних

Приклад рядку даних з файлу “reviews.json”: “{'ratings': {'service': 5.0, 'cleanliness': 5.0, 'overall': 5.0, 'value': 5.0, 'location': 5.0, 'sleep_quality': 5.0, 'rooms': 5.0}, 'title': '“Nice modern new rooms, comfy, clean, great location! Very nice stay”', 'text': 'Overall it is a pleasant stay at Westin. Staffs are quiet friendly and helpful. Rooms are modern, new and clean. The club lounge is also really nice with a view. I am very pleased with my stay.\nI do notice, if you do the club level rooms, it is on the 17th floor, ask for a room not below the gym!!!! or else you will hear weight dropping on your ceiling!!! \nParking is easy, skybridge over to the mall, it is a good value!', 'author': {'username': 'lammylammy', 'num_cities': 13, 'num_helpful_votes': 38, 'num_reviews': 23, 'num_type_reviews': 17, 'id': 'AAA3F46FCEE39DAB599CCC05BE9C63A8', 'location': 'Houston'}, 'date_stayed': 'March 2012', 'offering_id': 1966350, 'num_helpful_votes': 2, 'date': 'March 8, 2012', 'id': 125810806, 'via_mobile': False}”.

Тут ключами та значеннями є відповідно:

- 'ratings': поле типу словник з оцінками кожного аспекту відгуку. Аспектами є відповідно: 'service' - обслуговування, 'cleanliness' - чистота, 'overall' - загальна оцінка, 'value' - співвідношення ціни та якості відпочинку, 'location' - місцезнаходження, 'sleep_quality' - якість сну, 'rooms' - оцінка стану кімнат;
- 'title': текстове поле з короткою назвою відгуку, що відображає основну думку автора;
- 'text': текстове поле з повним відгуком автора;

- 'author': поле типу словник з інформацією про автора відгуку. Це поле містить таку інформацію: 'username' - ім’я автора на сайті, 'num_cities' - кількість відвіданих міст, 'num_helpful_votes' - кількість голосів, що були відмічені як корисні, 'num_reviews' - загальна кількість залишених відгуків автором, 'num_type_reviews' - загальна кількість відгуків, залишена користувачем на певний вид туристичного місця (готелі, пам’ятки, тощо), 'id' - ідентифікаційний номер користувача, 'location' - район проживання користувача;
- 'date_stayed' - текстове поле, що містить дату, коли користувач відвідав готель;
- 'offering_id' - числове поле з ідентифікаційним номером готелю;
- 'num_helpful_votes' - числове поле з кількістю відміток, що цей відгук є корисним;
- 'date' - текстове поле, що містить дату коли автор залишив відгук на сайті;
- 'id' - числове поле з ідентифікаційним номером відгуку;
- 'via_mobile' - поле типу bool, що позначає чи був залишений відгук з мобільного пристрою.

3.2 Попередня обробка даних

В поданому форматі дані неможливо передати на опрацювання нейронній мережі, тому необхідно провести попередню обробку.

Для дослідження необхідні дані, що закріплені за ключами 'ratings', 'id' , 'text'. Для обробки даних в свою чергу створено класи Review Sentence ReadData та BootStrap. Структуру цих класів можна побачити на Рис. 3.1 - Рис. 3.4.

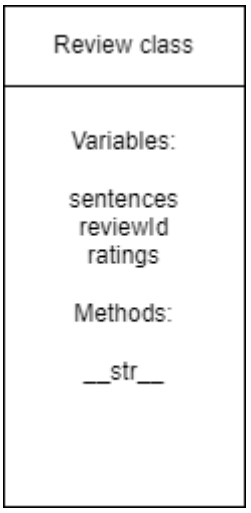


Рис. 3.1 - структура класу Review.

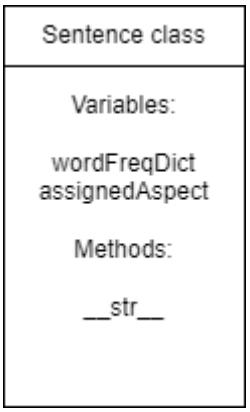


Рис. 3.2 - структура класу Sentence.

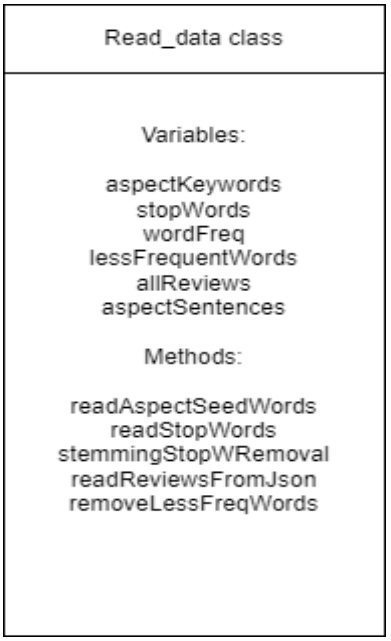


Рис. 3.3 - структура класу ReadData.

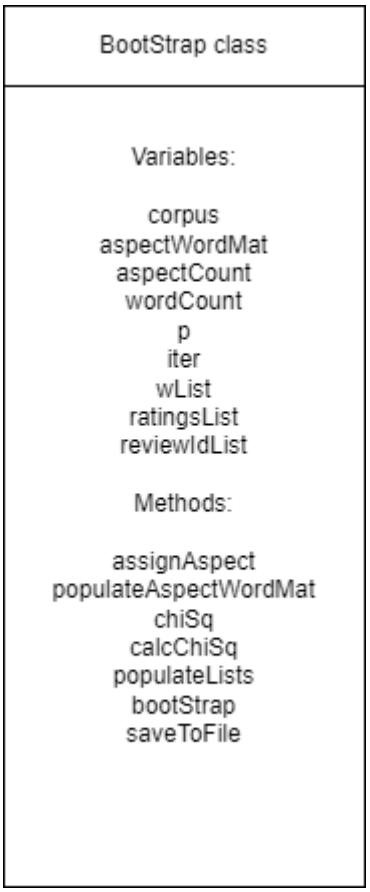


Рис. 3.4 - структура класу BootStrap.

В першу чергу було проведено парсинг файлу з відгуками з метою відокремити необхідні поля та створити для кожного відгуку змінну класу Review. Кожне речення в коментарі відповідно також записано в змінну класу Sentence.

Наступним кроком є очистка даних. В першу чергу видаляються всі знаки пунктуації, зайві символи, що можуть додати складності при роботі моделі, та всі числа. Одразу після цього видаляються так звані стоп слова. Маються на увазі артиклі та слова, що часто використовуються в англійській мові, але не несуть змістового навантаження. Список таких слів завантажено за допомогою бібліотеки nltk.

Після цього необхідно лематизувати коментарі, Лематизація передбачає групування відмінюваних форм слова так, щоб їх можна було проаналізувати як єдиний елемент, ідентифікований за лемою слова або словниковою формою.

Паралельно з цим, з усіх коментарів формується словник. Він включає в себе множину всіх слів, що використовувались в коментарях, не включаючи рідко використовувані слова, що виокремлені методом FreqDist.

Останнім кроком є збереження опрацьованих коментарів, оцінок та настрою в файли формату .csv (для LSTM моделі) та .json (для LARA моделі) для подальшого використання.

3.3 Використання моделі на базі LSTM

Як приклад класичного підходу до вирішення поставленої задачі побудована модель на основі LSTM. На Рис. 3.5 можемо побачити архітектуру моделі:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 128)          384000

 lstm (LSTM)                 (None, 64)                49408

 dense (Dense)               (None, 32)                2080

 dense_1 (Dense)             (None, 11)                363

=================================================================
Total params: 435,851
Trainable params: 435,851
Non-trainable params: 0
_________________________________________________________________
```

Рис. 3.5 - модель на основі LSTM

Метрики під час тренування моделі можна побачити в таблиці 3.1.

Табл. 3.1 - зміна точності та функції втрат моделі на основі LSTM під час тренування.

№ епохи	Тренувальні втрати	Перевірочні втрати	Тренувальна точність	Перевірочна точність
1	2.0435	2.0315	0.2019	0.2033
2	2.0276	2.0224	0.2049	0.2041
3	2.0161	2.0323	0.2060	0.2018
4	2.0002	2.0362	0.2207	0.2021
5	1.9810	2.0559	0.2243	0.2031

Також нижче на Рис.3.6 - Рис.3.9 вказано графіки зі змінами втрат та точності в процесі тренування.

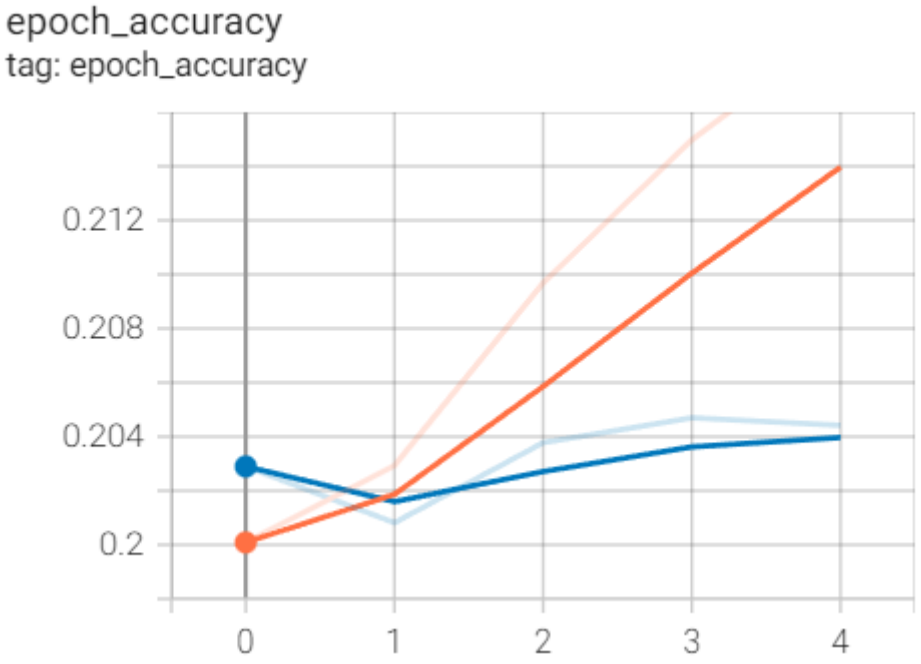


Рис. 3.6 - зміна точності по епохам

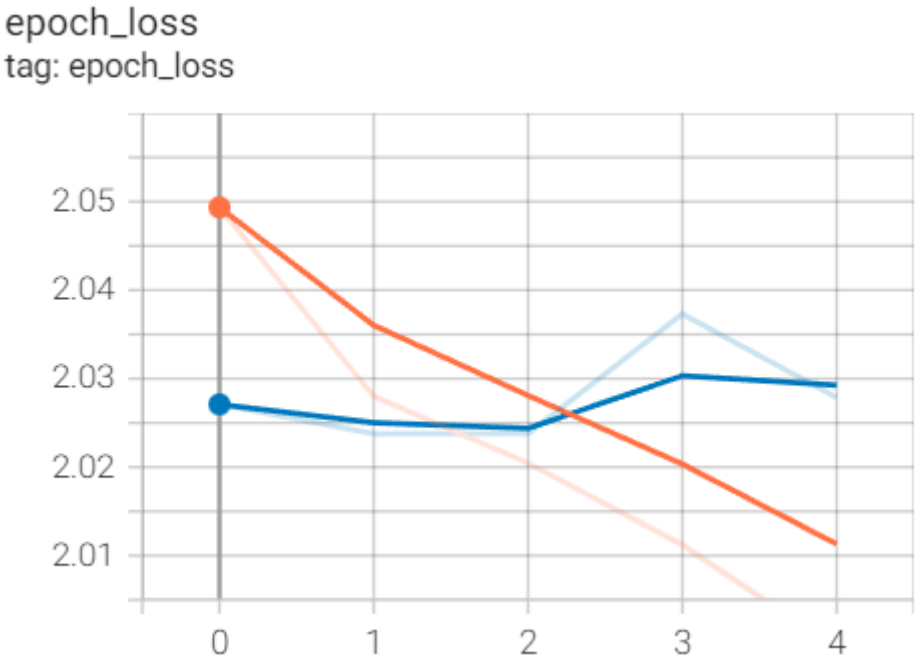


Рис. 3.7 - зміна втрат по епохам

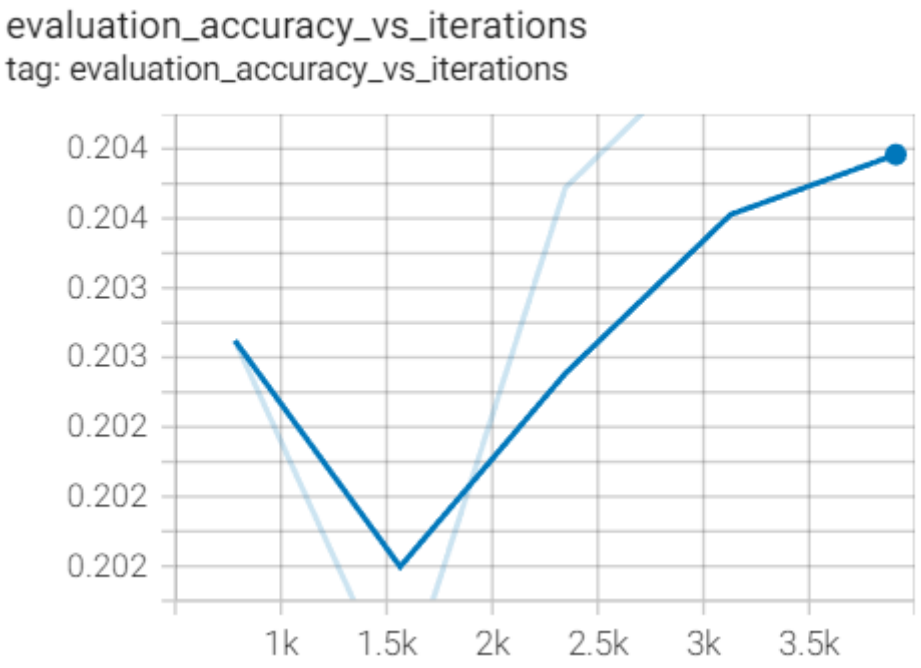


Рис.. 3.8 - оцінювання точності за ітераціями

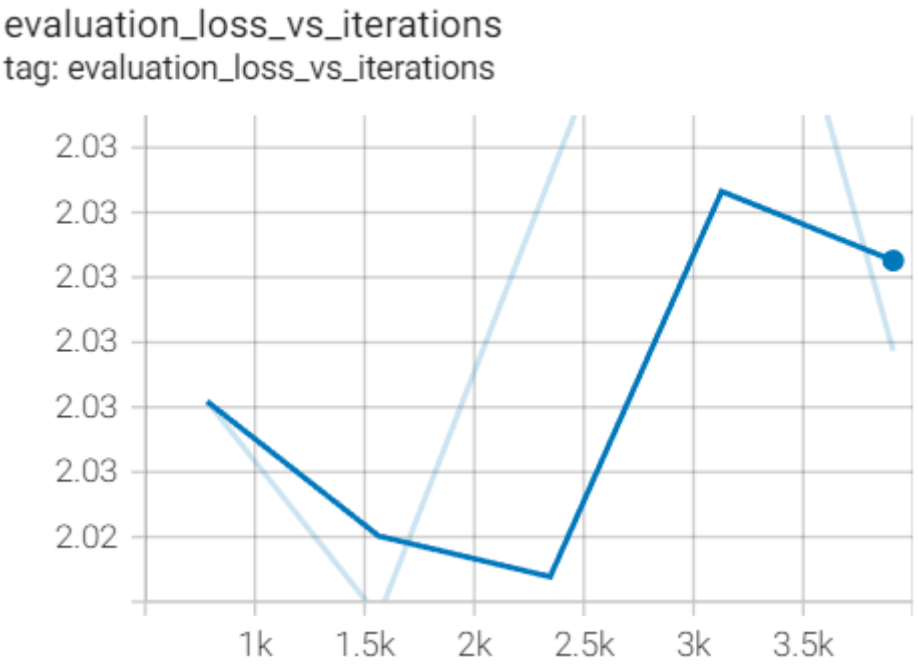


Рис. 3.9 - оцінювання втрат за ітераціями

3.4 Використання моделі LARA

Для побудови моделі на основі LARA використано клас LRR. На Рис. 3.10 зображено структуру класу LRR.

Рис. 3.10 - структура класу LRR

Метрики під час тренування моделі можна побачити в Табл 3.2 нижче.

Табл 3.2 - зміна точності та функції втрат моделі на основі LSTM під час тренування.

№ епохи	Тренувальні втрати	Перевірочні втрати	Тренувальна точність	Перевірочна точність
1	1.5591	1.6032	0.4020	0.4001
2	1.5417	1.5247	0.4197	0.4221
3	1.5120	1.4534	0.4260	0.4230
4	1.4763	1.4072	0.4307	0.4294
5	1.4077	1.3899	0.4387	0.4402

Також нижче на Рис. 3.11 - Рис. 3.14 вказано графіки зі змінами втрат та точності в процесі тренування.

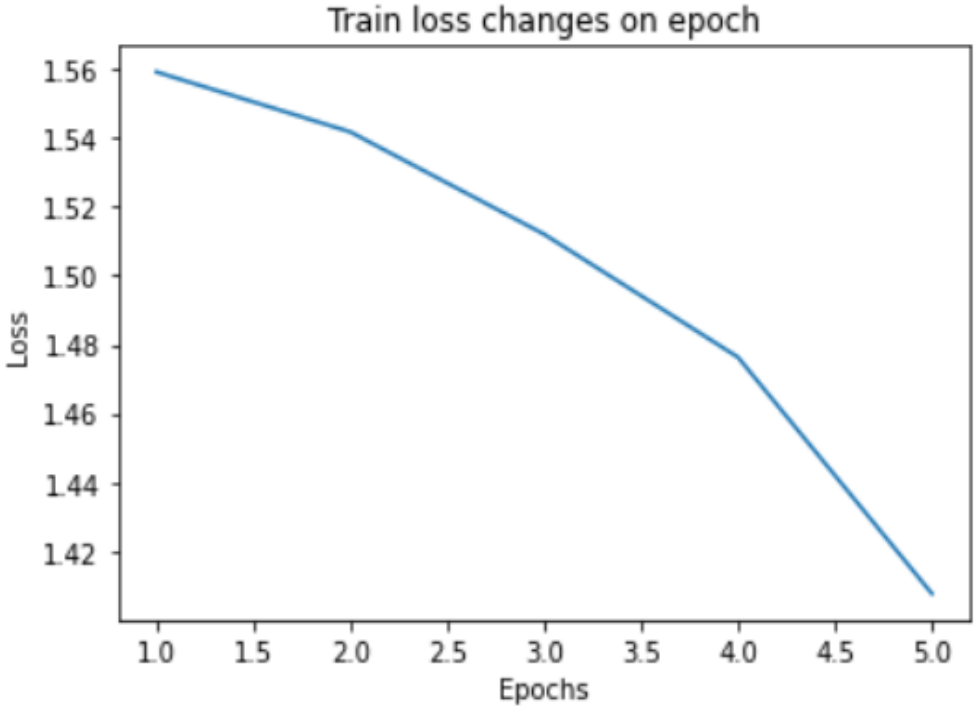


Рис. 3.11 - зміна функції втрат у процесі тренування.

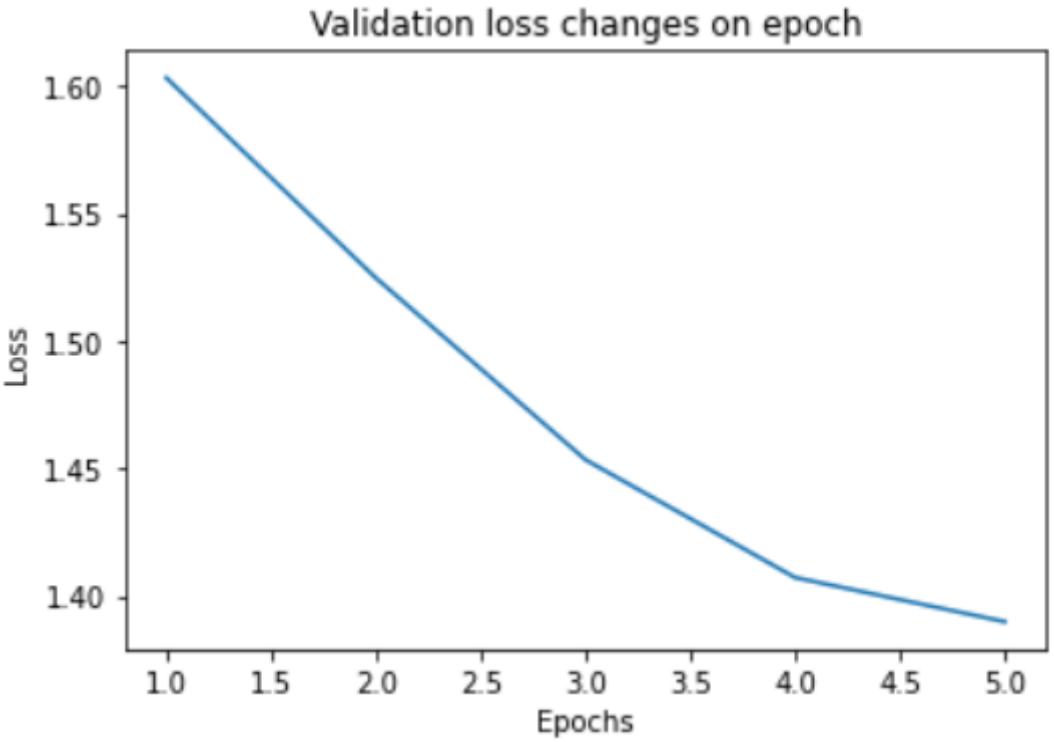


Рис. 3.12 - зміна функції втрат на перевірочному наборі.

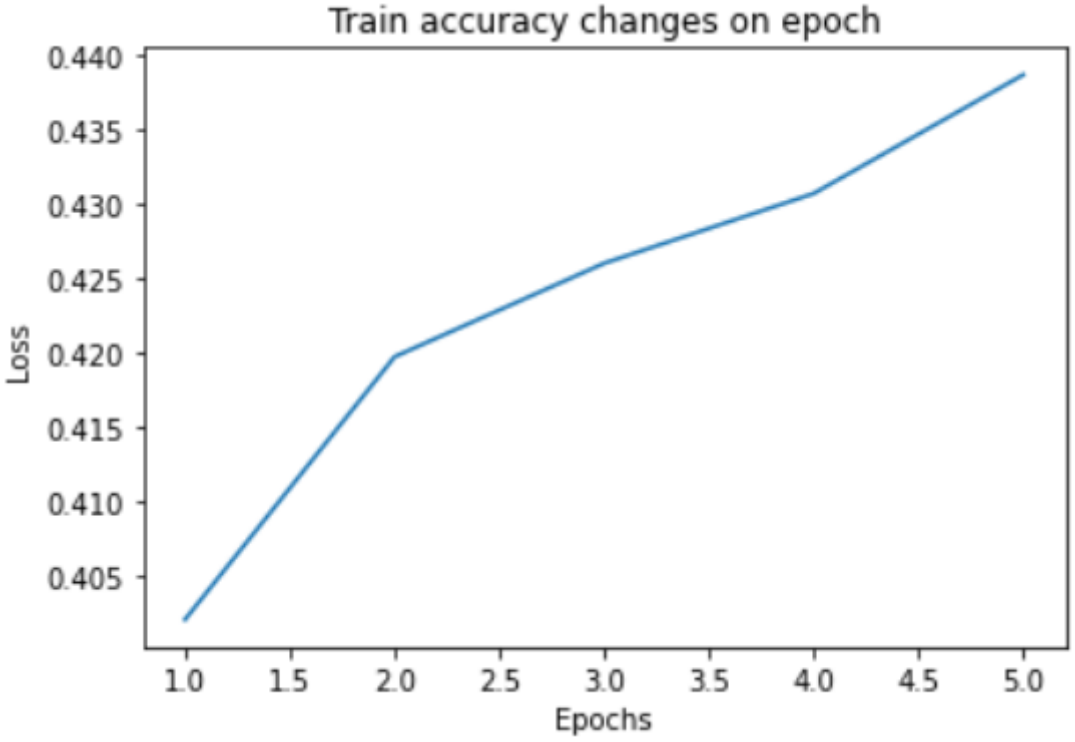


Рис. 3.13 - зміна точності моделі під час тренування.

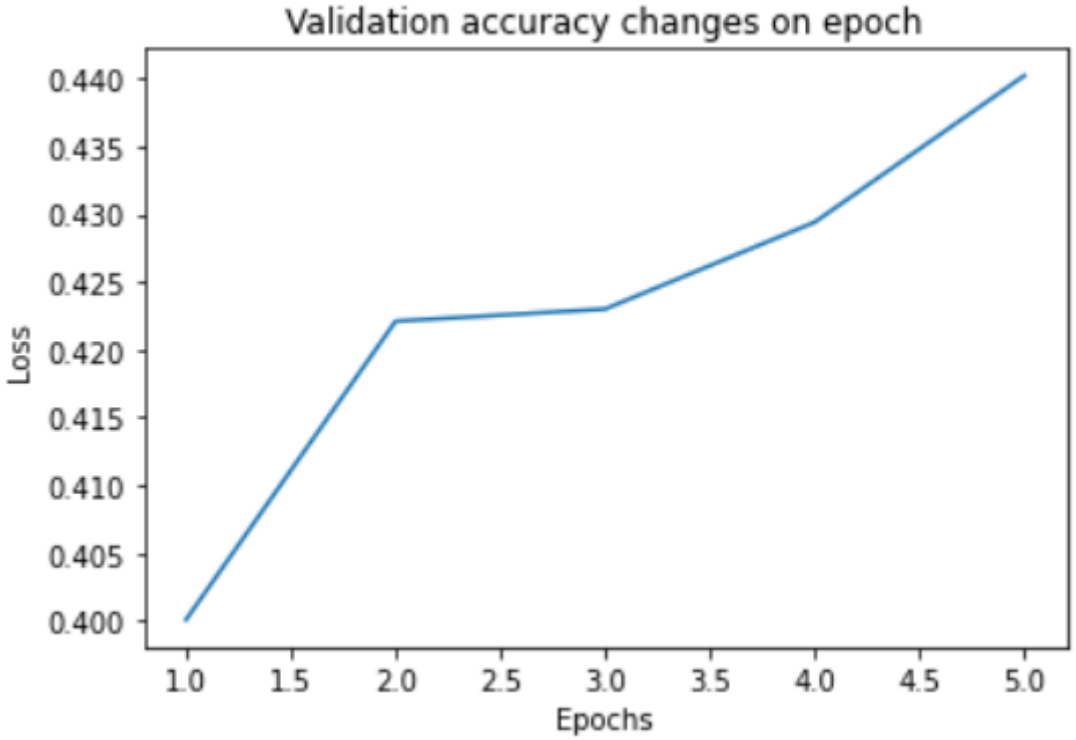


Рис. 3.14 - зміна точності на перевірочному наборі.

3.5 Висновки до розділу 3

В цьому розділі побудовано моделі машинного навчання на базі LSTM та LRR. Для тренування моделей обраний набір даних (набір даних Trip Advisor) було попередньо зчитано та попередньо оброблено використовуючи модуль “data_read”, та класи, розроблені в ньому.

Для безпосереднього порівняння результатів моделі демонстративно натреновані на невеликому фрагменті набору даних (1000 коментарів) для швидкості тренування моделей. З метрик тренування можна побачити, що підхід з використанням латентного аналізу виявився набагато кращим ніж кращий з класичних методів рішення задачі.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на аналізі настроїв за аспектами тексту.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити поставлене питання.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп’ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – якісний аналіз даних.

F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

- Python.
- C#.

Функція F_2.

- Застосування вбудованих функцій.
- Створення своїх обчислень значень.

Функція F_3:

- Використання шаблонних графіків.
- Створення своїх.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

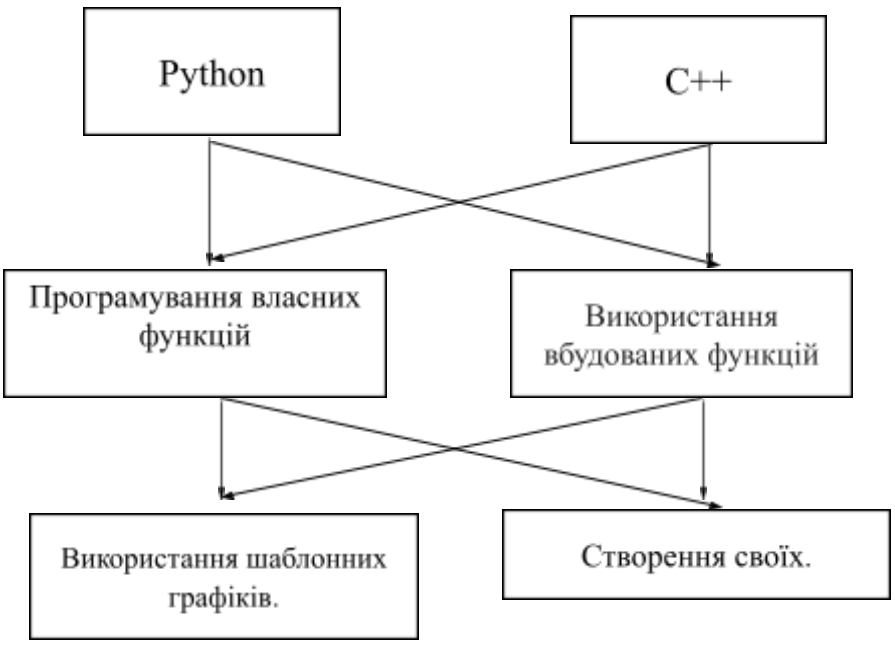


Рис. 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Проста у використанні, має широкий спектр інструментів для моделювання та тренування нейронних мереж.	Повільна в порівнянні з іншими варіантами. Потребує набагато більше ресурсів через відсутність динамічної типізації.
	Б	Швидка, різнорівнева, мова програмування з гарно побудованим ООП.	Складніший синтаксис, погана підтримка модульності, складніший процес усунення можливих помилок.
F_2	А	Висока гнучкість та функціональність.	Складніший синтаксис, погана підтримка модульності, складніший процес усунення можливих помилок.
	Б	Швидка реалізація необхідних задач без додаткової розробки коду.	Обмежена функціональність, менша гнучкість.
F_3	А	Швидкість та простота, відсутність необхідності додаткової розробки коду	Обмежена функціональність, менша гнучкість.
	Б	Висока гнучкість та функціональність, можливість вибору власного стилю	Повільний та працезатратний процесс, оскільки включає розробку коду.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто

відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Віддаємо перевагу варіанту А, оскільки цей варіант більш загально доступний та не вимагає додаткових зусиль при написанні коду, відкидаючи варіант Б для спрощення роботи та швидкості її виконання.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для обчислень та збереження даних;
- X3 – час навчання даних;

- X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	22	79	117
Об’єм пам’яті	X2	Мб	200	80	45
Час попередньої обробки даних	X3	мс	105	50	25
Потенційний об’єм програмного коду	X4	кількість рядків коду	3000	1700	1200

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

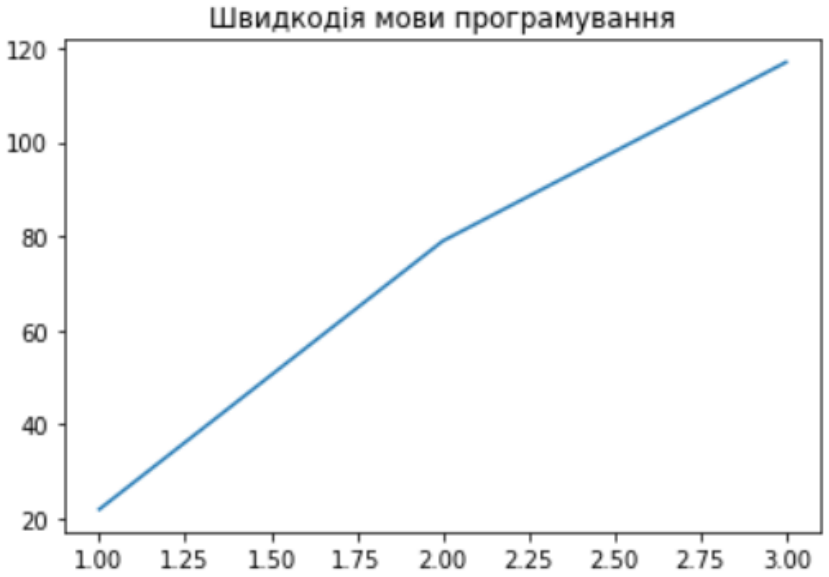


Рис. 4.2 – X1, швидкодія мови програмування

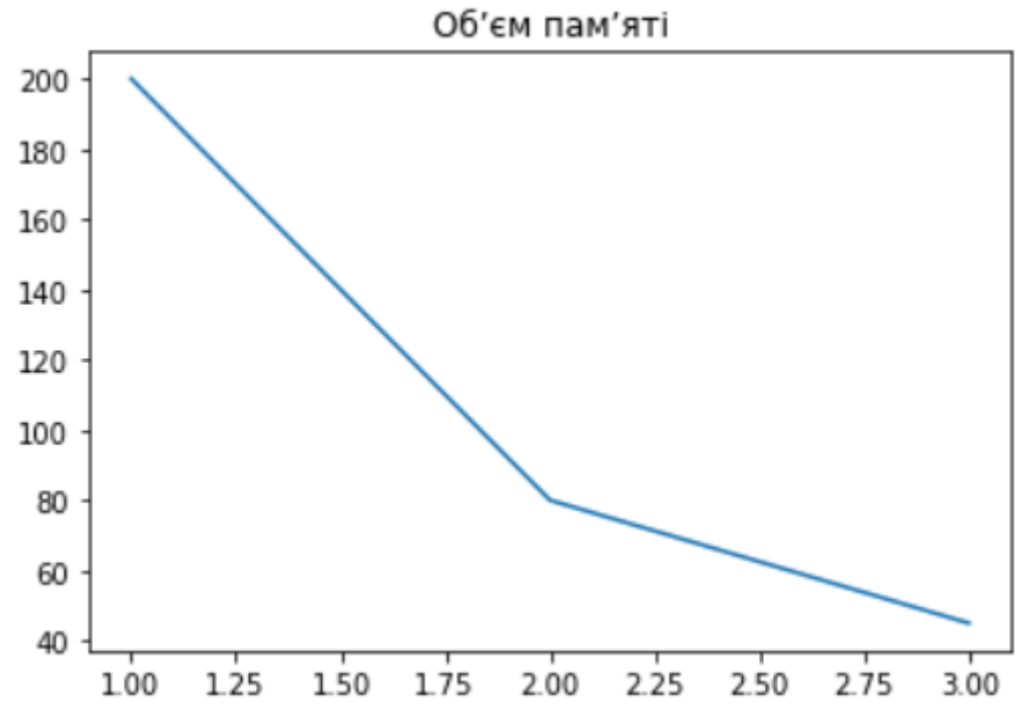


Рис. 4.3 – X2, об'єм пам'яті

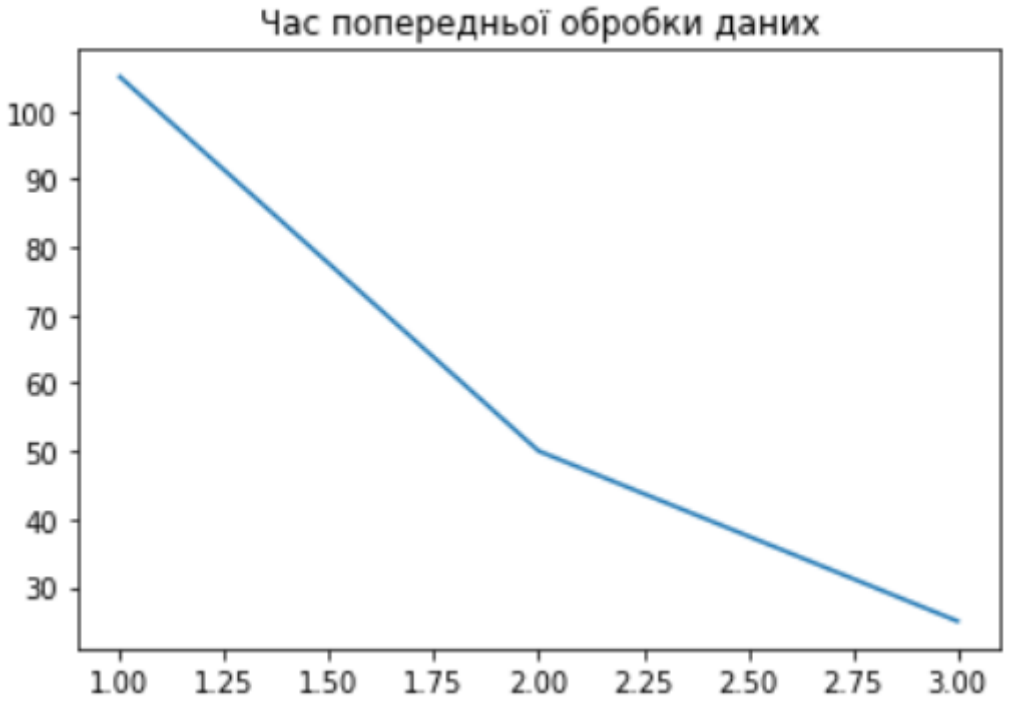


Рис. 4.4 – X3, час попередньої обробки даних

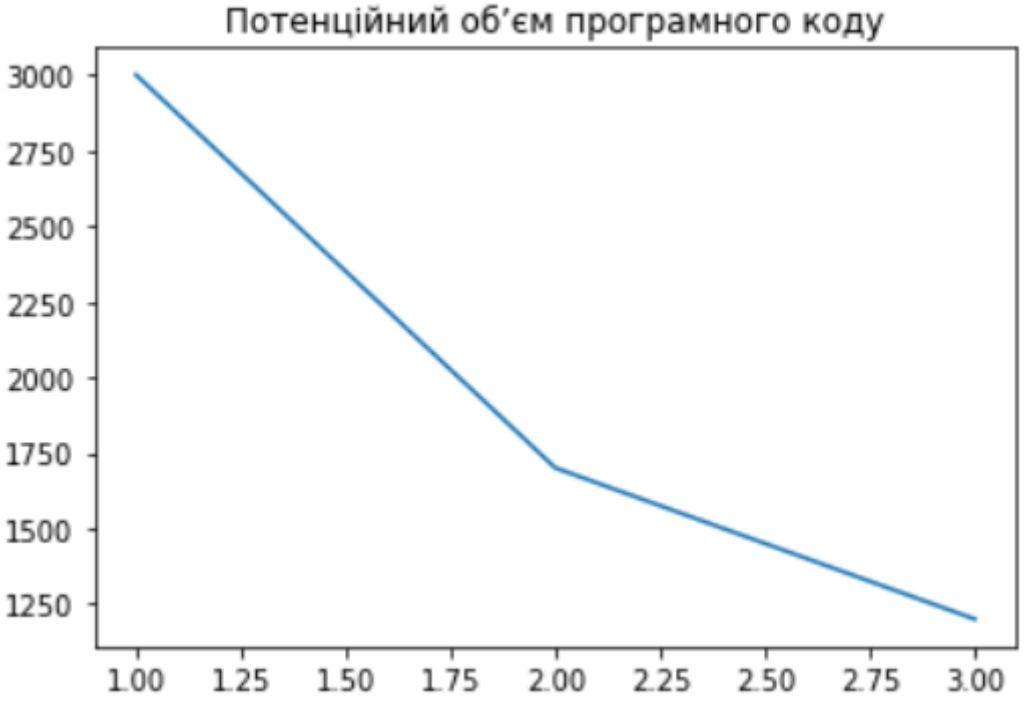


Рис. 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;

- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	2	1	1	1	2	2	1	10	-7,5	6,25
X2	Об’єм пам’яті	Mb	4	3	4	3	3	4	3	24	6,5	2,25
X3	Час попередньої обробки даних	мс	1	2	2	2	1	1	2	11	-6,5	2,25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	3	4	3	4	4	3	4	25	7,5	6,25
	Разом		10	10	10	10	10	10	10	70	0	97

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \ (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5 \ (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2\left(n^3-n\right)} = \frac{12 \cdot 197}{7^2\left(4^3-4\right)} = 0,754 > W_k = 0,67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати заносимо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	>	<	<	<	>	>	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	<	>	<	>	>	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги і–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j \; 1.0 \text{ при } X_i = X_j \; 0.5 \text{ при } X_i < X_j . \#(4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A=\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{ві} = \frac{b_i}{\sum_{i=1}^{n} b_i} \#(4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \#(4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{ві} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \ (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \ (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів.

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	$K_{ві}$	b_i^1	$K_{ві}^1$	b_i^2	$K_{ві}^2$
X1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
X3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (Об'єм пам'яті), X3 (час попередньої обробки даних) та X4 (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1 (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \quad (4.11)$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості і–го параметра;

B_i – оцінка і–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП.

Основні функції	Варіант реалізації функції	Пара-метри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	100	25	0,16	4
F3	А	X2	87	29	0,28	8,12
	Б	X3	27	19	0,2	3,8
F4	А	X4	25	23	0,36	8,28

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$К_{К1} = 4 + 8{,}12 + 8{,}28 = 20{,}4$;

$К_{К2} = 4 + 3{,}8 + 8{,}28 = 16{,}08$.

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ =37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$Т_1 = 37 \cdot 1.8 \cdot 0.9 = 59,94$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$Т_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I = (59,94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852$ людино-годин.

$Т_{II} = (59,94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868,88$ людино-годин.

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 35000 грн., один аналітик в області даних з окладом 30000. Визначаємо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$СЧ = \frac{35000+35000+30000}{3 \cdot 21 \cdot 8} = 198,41 \text{ грн.} \quad (4.15)$$

Тоді, розраховуємо заробітну плату за формулою:

$$СЗП = С_ч \cdot Т_i \cdot КД, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 198,41 \cdot 852 \cdot 1,2 = 202854,38$ грн.

II. $С_{ЗП} = 198,41 \cdot 868,88 \cdot 1,2 = 206873,37$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 202854,38 \cdot 0,22 = 44627,96$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 206873,37 \cdot 0,22 = 45512,14$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 35000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 35000 \cdot 0,2 = 84000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 84000 \cdot (1 + 0,2) = 100800$ грн.

Відрахування на соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0,22 = 100800 \cdot 0,22 = 22176$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,4 \cdot 0,25 \cdot 10000 = 3500$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,4 \cdot 10000 \cdot 0,08 = 1120$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0,35 =$

$= 627,2$ години,

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Втрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,25 \cdot 0,3 \cdot 4,87 = 229,08$ грн.,

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$С_{ЕКС} = 100800 + 22176 + 3500 + 1120 + 229,08 + 6700 = 134525,08$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 134525,08 / 627,2 = 214,48$ грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T, \quad (4.18)$$

I. $С_М = 214{,}48 \cdot 852 = 182736{,}96$ грн.

II. $С_М = 214{,}48 \cdot 868{,}88 = 186357{,}38$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0{,}67, \quad (4.19)$$

I. $С_Н = 202854{,}38 \cdot 0{,}67 = 135912{,}43$ грн.

II. $С_Н = 206873{,}37 \cdot 0{,}67 = 138605{,}15$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 202854{,}38 + 44627{,}96 + 182736{,}96 + 135912{,}43 = 566131{,}73$ грн.

II. $С_{ПП} = 206873{,}37 + 45512{,}14 + 186357{,}38 + 138605{,}15 = 577348{,}04$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{К_j} / С_{Ф_j}, \quad (4.21)$$

$К_{ТЕР1} = 22{,}38 / 566131{,}73 = 3{,}953 \cdot 10^{-5}$,

$К_{ТЕР2} = 16{,}88 / 577348{,}04 = 2{,}923 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 3{,}953 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, що

залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 3,953 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір програмного продукту –Python;
- програмування власних функцій для реалізації поставлених задач;
- використання стандартного інтерфейсу для побудови значень.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

4.8 Висновки до розділу 4

В даному розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розробляється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

Поява сучасної писемності, несумнівно, мала значний вплив на розвиток людства. Через тексти люди не лише передають інформацію між собою а й передають емоції та мають можливість самовираження.

В свою чергу текстова аналітика, як галузь науки, з’явилась зовсім недавно, разом з виникненням сучасних методів аналізу даних. З моменту виникнення галузі до нашого часу вже закарбувались певні класичні підходи до розв’язку задачі аналізу тексту, та це не відміняє факту, що нові та більш практичні методи з’являються впродовж останніх років. В першу чергу - це пов’язано з розвитком машинного навчання та області аналізу даних.

В роботі, безпосередньо, розглянуто один із напрямків аналізу тексту, а саме аналіз настроїв (або аналіз тональності тексту). Основна мета такого аналізу - виявлення в текстах емоційно забарвленої лексики і емоційної оцінки авторів (думок) по відношенню до об'єктів, мова про які йде в тексті.

Для аналізу настроїв в роботі розглянуто два підходи. Перший - класичний підхід використовуючи нейронну мережу на базі LSTM. Для цього створена та натренована модель, на наборі даних з сайту tripadvisor.com. Вкажемо переваги такого методу розв’язку поставленої задачі.

1. Послідовне моделювання: LSTM розроблено для фіксації довгострокових залежностей і послідовних шаблонів у даних. У аналізі настроїв, де важливі контекст і порядок слів, LSTM можуть ефективно вловлювати почуття, виражені в тексті.
2. Збереження пам’яті: LSTM мають комірку пам’яті, яка може зберігати інформацію протягом довгих послідовностей. Це дозволяє їм запам’ятовувати відповідну інформацію з попередніх частин тексту та використовувати її, щоб робити прогнози щодо настроїв пізніше в послідовності.

3. Обробка вхідних даних змінної довжини: речення або документи в аналізі настроїв можуть мати різну довжину. LSTM можуть обробляти вхідні дані змінної довжини, обробляючи їх одне слово за раз і відповідно динамічно оновлюючи їхні внутрішні стани.
4. Вилучення функцій: LSTM можуть автоматично вивчати значущі представлення тексту через їхні приховані стани. Ця здатність допомагає витягувати релевантні характеристики з вхідного тексту для аналізу настроїв.
5. Передача навчання: LSTM, навчені на великих текстових наборах даних, таких як загальні мовні моделі, можуть бути налаштовані на завдання аналізу настроїв. Цей підхід навчання передачі дозволяє використовувати попередньо навчені моделі для покращення продуктивності та зменшує потребу у великих обсягах позначених даних настрою.

В свою чергу вкажемо недоліки.

1. Ресурсозатратні розрахунки: LSTM можуть бути ресурсозатратними для навчання та оцінки, особливо на великих наборах даних. Вони мають велику кількість параметрів, що вимагає більших обчислювальних ресурсів і часу на навчання.
2. Перенавчання: LSTM схильні до перенавчання, особливо коли навчаються на невеликих наборах даних. Вони мають велику кількість параметрів, і без достатньої кількості навчальних даних вони можуть запам’ятовувати навчальні приклади замість того, щоб вивчати загальні шаблони настроїв.
3. Відсутність інтерпретації: LSTM є моделями чорної скриньки, тобто може бути важко зрозуміти, як вони приходять до своїх прогнозів. Інтерпретація вивчених репрезентацій і розуміння міркувань конкретних прогнозів настроїв може бути складним.
4. Робота зі словами поза словниковим запасом: LSTM обробляють текст на рівні слова, і вони можуть зустріти слова, яких не було в

їхніх навчальних даних. Робота зі словами поза словниковим запасом може бути проблемою, оскільки вони можуть бути неадекватно представлені або зрозумілі моделлю LSTM.

Як противага до класичних підходів у роботі розглянуто використання латентно аспектної регресії для досягнення аналогічних цілей. Це техніка, яка використовується в аналізі настрою для виявлення та аналізу різних аспектів або вимірів настрою в тексті. Вкажемо плюси латентного аспектного регресійного аналізу для аналізу настрою.

1. Аналіз настрою на рівні аспекту: LAR дає змогу детально аналізувати настрої на рівні аспекту. Це допомагає визначити та кількісно оцінити настрої, пов’язані з конкретними аспектами чи особливостями продукту, послуги чи теми. Це дає більш детальне розуміння настроїв і може бути цінним для бізнесу з точки зору визначення областей покращення або підкреслення сильних сторін.
2. Розкриття прихованих аспектів: LAR дозволяє виявити приховані або приховані аспекти, які можуть бути явно не згадані в тексті. Це особливо корисно під час роботи з великими обсягами неструктурованих даних, де може бути складно вручну визначити всі відповідні аспекти. LAR допомагає виявити глибинні аспекти настроїв, які впливають на загальну думку.
3. Результати, які можна інтерпретувати: LAR надає результати, які можна інтерпретувати, пов’язуючи оцінки настрою з певними аспектами. Це дозволяє чітко зрозуміти, які аспекти викликають позитивні чи негативні настрої. Це допомагає зацікавленим сторонам визначити сильні чи слабкі сторони продукту чи послуги та прийняти обґрунтовані рішення на основі аналізу.
4. Покращений вибір функцій: LAR може допомогти у виборі функцій, визначаючи найважливіші аспекти, які сприяють настрою. Це допомагає визначити пріоритетність аспектів, які

мають найбільший вплив на настрої, дозволяючи дослідникам зосередити свою увагу та ресурси на цих конкретних сферах.

Відповідно вкажемо наступні недоліки.

1. Залежність від попередньо визначених аспектів: LAR зазвичай вимагає попередньо визначеного набору аспектів або функцій для виконання аналізу. Якщо попередньо визначені аспекти не є вичерпними або точними, аналіз може пропустити важливі аспекти настроїв. Якість результатів значною мірою залежить від початкового вибору аспектів, що може вимагати ручних зусиль і суб’єктивності.
2. Обмежена можливість узагальнення: аналіз LAR стосується попередньо визначеного набору аспектів, які використовуються в аналізі. Це може бути погано узагальнено для нових або невідомих аспектів, які не були враховані під час аналізу. Отже, результати аналізу настроїв можуть бути неповністю застосовні до різних контекстів або доменів, що обмежує масштабованість методики.
3. Комплексна реалізація: впровадження LAR для аналізу настроїв вимагає досвіду статистичного моделювання та обробки природної мови. Це передбачає роботу зі складними алгоритмами, попередню обробку даних і навчання моделі. Технічна складність LAR може ускладнити роботу користувачів, які не мають необхідних знань і ресурсів для її ефективного застосування.
4. Доступність і якість даних: LAR значною мірою покладається на доступність анотованих даних, які пов’язують почуття з певними аспектами. Отримання високоякісних анотованих даних може бути тривалим і дорогим. Крім того, якщо анотовані дані не точно відображають цікаві аспекти та настрої, це може вплинути на надійність і валідність аналізу.

Після створення, тренування, оцінювання моделей та аналізу метрик, отриманих під час тренування можна відмітити, що підхід з використанням латентно аспектної регресії виявився помітно результативнішим за класичний. При однаковому часі та розміру набору даних для тренування цей підхід має значно кращі метрики ніж модель на базі LSTM. Це свідчить про те, що аналіз настроїв з використанням цієї техніки дає більш точні результати в порівнянні з прямим конкурентом.

Актуальність даної роботи заключається в дослідженні відносно нової техніки до розв’язку задач аналізу настроїв з метою оцінки результатів. У подальшому отримані результати можна використовувати при виборі методу для вирішення подібних задач. Крім того, виконана робота може бути підґрунтям до таких робіт як генерація тексту виходячи з оцінки аспектів, або класифікація текстів за настроями на основі оцінок аспектів. Ця робота є значною основою для майбутньої магістерської роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ренькас Л.Е. Віртуальна виставка “Писемність - душа кожної нації”, Бібліотека Житомирського державного університету, Житомир 2016. URL: http://library.zu.edu.ua/doc/book/doc/pysemnist.pdf (дата звернення: 01.05.2023)
2. Wang, H., Lu, Y., Li, H., & Chen, X. Latent aspect rating analysis on review text data: a rating regression approach. In Proceedings of ACM KDD 2010.
3. Liu B, Sentiment analysis and opinion mining, Morgan & Claypool Publishers, 2012.
4. Pang B., Lee L. Opinion mining and sentiment analysis, Foundations and Trends in Information Retrieval 2(1-2).
5. Wang, H., Lu, Y., Li, H., & Chen, X.. 2011. Latent aspect rating analysis without aspect keyword supervision. In Proceedings of ACM KDD 2011.
6. Staudemeyer R. C., Morris E. R., Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks.
7. Liu B. Sentiment Analysis and Opinion Mining, Morgan & Claypool Publishers, May 2012.
8. Лемехова І.В., Мацевко Н.В. Аналіз емоційних та настроєвих станів в текстах: проблеми та методи. Київ: Видавництво Київського національного університету імені Тараса Шевченка, 2018.
9. Потебенько В.В. Емоційний аналіз текстів з використанням машинного навчання". Київ: Видавничо-поліграфічний центр "Київський університет, 2015.
10. Семеног О.В. Комп'ютерний аналіз емоційних настроїв україномовних текстів. Київ: Видавництво Національного технічного університету України "КПІ", 2012.

11. Леус І.В., Помиткіна Т.М. Емоційний аналіз текстів: методи та алгоритми. Львів: Видавництво Львівської політехніки, 2017.

12. Левченко О.А. Комп'ютерний аналіз емоційних настроїв українською мовою. Київ: Видавництво Інституту мовознавства імені О.О. Потебні НАН України, 2013.

13. Cambria E., Hussain A., Havasi C. SenticNet 3: A Common and Common-Sense Knowledge Base for Cognition-Driven Sentiment Analysis. In: TALIP Journal of Natural Language Processing, Vol. 20, No. 2, 2013. p. 383-405.

14. Pang B., Lee L., Vaithyanathan S. Thumbs up? Sentiment Classification using Machine Learning Techniques. In: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing, 2002. p. 79-86.

15. Liu B. Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2012.

16. Agarwal A., Xie B., Vovsha I., Rambow O., Passonneau R. "Sentiment Analysis of Twitter Data". In: Proceedings of the Workshop on Language in Social Media, 2011. p. 30-38.

17. Pak A., Paroubek P. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In: Proceedings of the 7th Conference on International Language Resources and Evaluation (LREC'10), 2010. p. 1320-1326.

18. Brody, S., & Elhadad, N. An unsupervised aspect-sentiment model for online reviews. In Proceedings of the 14th Conference on Computational Natural Language Learning, 2010. p. 45-53.

19. Pontiki, M., Galanis, D., Pavlopoulos, J., Papageorgiou, H., & Androutsopoulos, I. SemEval-2014 Task 4: Aspect Based Sentiment Analysis. In Proceedings of the 8th International Workshop on Semantic Evaluation, 2014. p. 27-35.

20. Xu, H., Liu, B., Shu, L., & Yu, P. S. Aspect term extraction with structured learning. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013. p. 860-870.

21. Wang, H., Lu, Y., Li, H., & Chen, X. Aspect sentiment classification with aspect-specific dependency regularization. Knowledge-Based Systems, 2019. p. 9-17, 161.

22. Li, C., Li, S., Cao, Z., Wei, F., & Liu, Y. Deep latent variable models for sentiment analysis in social media. In Proceedings of the AAAI Conference on Artificial Intelligence, 2019. p 5567-5574.

23. Tang, D., Qin, B., & Liu, T. Aspect level sentiment classification with deep memory network. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016. p. 214-224.

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

Модуль data_read.ipynb

```
import json
import nltk
from nltk.corpus import stopwords
import os
import glob
from nltk.stem.porter import *
from collections import defaultdict
import string
import heapq
import numpy as np
from nltk import FreqDist

modelDataDir = "modelData/"
path="Reviews/"
utilits="utilits/"

class Review:
    def __init__(self):
        self.sentences = [] #list of objects of class Sentence
        self.reviewId = ""
        self.ratings = {} #true ratings provided by the user

    def __str__(self):
        retStr = ""
        for sentence in self.sentences:
            retStr += sentence.__str__() + '\n'
                                                    retStr      +=
"###"+self.reviewId+"###"+str(self.ratings)+"\n"
        return retStr

class Sentence:
    def __init__(self, wordList):
```

```
                self.wordFreqDict = FreqDist(wordList)#Dictionary of
words in the sentence and corres. frequency
                self.assignedAspect = [] #list of aspects assigned to
this sentence
          def __str__(self):
                  return self.wordFreqDict.pformat(10000) + '##' +
str(self.assignedAspect)

      class ReadData:
          def __init__(self):
                self.aspectKeywords = {} #aspect name <--> keywords
list
              self.stopWords = []
               self.wordFreq = {} #dict with of all words and their
freq in the corpus
                    self.lessFrequentWords=set() #words which have
frequency<5 in the corpus
                self.allReviews = [] #list of Review objects from the
whole corpus
                 self.aspectSentences = defaultdict(list) #aspect to
Sentences mapping

          def readAspectSeedWords(self):
              with open(utilits+"SeedWords.json") as fd:
                  seedWords = json.load(fd)
                  for aspect in seedWords["aspects"]:
                                self.aspectKeywords[aspect["name"]] =
aspect["keywords"]

          def readStopWords(self):
              with open(utilits+"stopwords.dat") as fd:
                  for stopWord in fd:
                      self.stopWords.append(stopWord.strip())
              for stopWord in stopwords.words('english'):
                  if stopWord not in self.stopWords:
                      self.stopWords.append(stopWord)
              #print(self.stopWords)

          def stemmingStopWRemoval(self, review, vocab):
```

```
            ''' Does Following things:
            1. Tokenize review into sentences, and then into words
            2. Remove stopwords, punctuation and stem each word
            3. Add words into vocab
                4. Make Sentence objects and corresponding Review
object
            '''
            reviewObj = Review()
            #copying ratings into reviewObj
            for key, value in review["ratings"].items():
                reviewObj.ratings[key] = value
            reviewObj.reviewId = str(review["id"])

            stemmer = PorterStemmer()
            reviewContent = review["text"]
            #TODO: Append title too!
            sentencesInReview = nltk.sent_tokenize(reviewContent)
            puncs = set(string.punctuation) #punctuation marks
            for sentence in sentencesInReview:
                wordList=[]
                words = nltk.word_tokenize(sentence)
                for word in words:
                      if not all(c.isdigit() or c in puncs for c in
word):
                        word = word.lower()
                        if word not in self.stopWords:
                            word=stemmer.stem(word.lower())
                            vocab.append(word)
                            wordList.append(word)
                if wordList:
                    sentenceObj=Sentence(wordList)
                    reviewObj.sentences.append(sentenceObj)
            if reviewObj.sentences:
                self.allReviews.append(reviewObj)
                # print(reviewObj)

        def readReviewsFromJson(self):
                    ''' Reads reviews frm the corpus, calls
stemmingStopWRemoval
```

```
            and creates list of lessFrequentWords (frequency<5)
            '''
            vocab=[]
            i = 0
                    for  filename  in  glob.glob(os.path.join(path,
'*.json')):
                with open(filename, 'r') as file:
                    # print(file)
                    for line in file:
                        try:
                            data=json.loads(line)
                            self.stemmingStopWRemoval(data,vocab)
                        except:
                            print('error')
                            continue
                        i+=1
                        if i > 1000:
                            break
                print(data)
                # for review in data:
                #     print(type(review))
                #     print(review)
                # self.stemmingStopWRemoval(data,vocab)
            self.wordFreq = FreqDist(vocab)
            for word,freq in self.wordFreq.items():
                if freq < 4:
                    self.lessFrequentWords.add(word)
            for word in self.lessFrequentWords:
                del self.wordFreq[word]

            print("Less Frequent Words ",self.lessFrequentWords)
            print("Vocab ", self.wordFreq.pformat(10000))

        def removeLessFreqWords(self):
            emptyReviews = set()
            for review in self.allReviews:
                emptySentences = set()
                for sentence in review.sentences:
                    deleteWords = set()
```

```
                        for word in sentence.wordFreqDict.keys():
                            if word in self.lessFrequentWords:
                                deleteWords.add(word)
                        for word in deleteWords:
                            del sentence.wordFreqDict[word]
                        if not sentence.wordFreqDict:
                            emptySentences.add(sentence)
                    review.sentences[:] = [x for x in review.sentences
if x not in emptySentences]
                    if not review.sentences:
                        emptyReviews.add(review)
                self.allReviews[:] = [x for x in self.allReviews if x
not in emptyReviews]

        class BootStrap:
            def __init__(self, readDataObj):
                self.corpus = readDataObj
                   #Aspect,Word -> freq matrix - frequency of word in
that aspect
                            self.aspectWordMat  =  defaultdict(lambda:
defaultdict(int))
                #Aspect --> total count of words tagged in that aspect
                 # = sum of all row elements in a row in aspectWordMat
matrix
                self.aspectCount = defaultdict(int)
                #Word --> frequency of jth tagged word(in all aspects)
                   # = sum of all elems in a column in aspectWordMat
matrix
                self.wordCount = defaultdict(int)

                 #Top p words from the corpus related to each aspect to
update aspect keyword list
                self.p=5
                self.iter=7

                #List of W matrix
                self.wList=[]
                #List of ratings Dictionary belonging to review class
                self.ratingsList=[]
```

```
            #List of Review IDs
            self.reviewIdList=[]

            '''def calcC1_C2_C3_C4(self):
                                    for aspect, sentence in
self.corpus.aspectSentences.items():
                    for sentence in sentences:
                            for word in self.corpus.wordFreq.keys()
and not in sentence.wordFreqDict.keys():
                            self.aspectNotWordMat[aspect][word]+=1
                                                for word,freq in
sentence.wordFreqDict.items():
                            self.aspectWordMat[aspect][word]+=freq
            '''

          def assignAspect(self, sentence): #assigns aspects to
sentence
            sentence.assignedAspect = []
                count = defaultdict(int) #count used for aspect
assignment as in paper
                                    #print("IN ASSIGN ASPECT
FUNCTION:",len(sentence.wordFreqDict))
            for word in sentence.wordFreqDict.keys():
                                    for aspect, keywords in
self.corpus.aspectKeywords.items():
                    if word in keywords:
                        count[aspect]+=1
            if count: #if count is not empty
                maxi = max(count.values())
                for aspect, cnt in count.items():
                    if cnt==maxi:
                        sentence.assignedAspect.append(aspect)
             if(len(sentence.assignedAspect)==1): #if only 1 aspect
assigned to it

self.corpus.aspectSentences[sentence.assignedAspect[0]].append(senten
ce)

        def populateAspectWordMat(self):
```

```
            self.aspectWordMat.clear()
                                        for    aspect,    sentences    in
self.corpus.aspectSentences.items():
                for sentence in sentences:
                                                    for   word,freq   in
sentence.wordFreqDict.items():
                        self.aspectWordMat[aspect][word]+=freq
                        self.aspectCount[aspect]+=freq
                        self.wordCount[word]+=freq

        def chiSq(self, aspect, word):
            #Total number of (tagged) word occurrences
            C = sum(self.aspectCount.values())

              #Frequency of word W in sentences tagged with aspect
Ai
            C1 = self.aspectWordMat[aspect][word]

               #Frequency of word W in sentences NOT tagged with
aspect Ai
            C2 = self.wordCount[word]-C1

            #Number of sentences of aspect A, NOT contain W
            C3 = self.aspectCount[aspect]-C1

            #Number of sentences of NOT aspect A, NOT contain W
            C4 = C-C1

            deno = (C1+C3)*(C2+C4)*(C1+C2)*(C3+C4)
            #print(aspect, word, C, C1, C2, C3, C4)
            if deno!=0:
                return (C*(C1*C4 - C2*C3)*(C1*C4 - C2*C3))/deno
            else:
                return 0.0

        def calcChiSq(self):
            topPwords = {}
            for aspect in self.corpus.aspectKeywords.keys():
                topPwords[aspect] = []
```

```
                for word in self.corpus.wordFreq.keys():
                    maxChi = 0.0 #max chi-sq value for this word
                    maxAspect = "" #corresponding aspect
                    for aspect in self.corpus.aspectKeywords.keys():
                                    self.aspectWordMat[aspect][word] =
self.chiSq(aspect,word)
                        if self.aspectWordMat[aspect][word] > maxChi:
                            maxChi = self.aspectWordMat[aspect][word]
                            maxAspect = aspect
                    if maxAspect!="":
                        topPwords[maxAspect].append((maxChi, word))

                changed=False
                for aspect in self.corpus.aspectKeywords.keys():
                    for t in heapq.nlargest(self.p,topPwords[aspect]):
                                                if t[1] not in
self.corpus.aspectKeywords[aspect]:
                            changed=True

self.corpus.aspectKeywords[aspect].append(t[1])
                return changed

            # Populate wList,ratingsList and reviewIdList
            def populateLists(self):
                for review in self.corpus.allReviews:
                    #Computing W matrix for each review
                    W = defaultdict(lambda: defaultdict(int))
                    for sentence in review.sentences:
                        if len(sentence.assignedAspect)==1:
                                                for word,freq in
sentence.wordFreqDict.items():

W[sentence.assignedAspect[0]][word]+=freq
                    if len(W)!=0:
                        self.wList.append(W)
                        self.ratingsList.append(review.ratings)
                        self.reviewIdList.append(review.reviewId)
```

```
            def bootStrap(self):
                changed=True
                while self.iter>0 and changed:
                    self.iter-=1
                    self.corpus.aspectSentences.clear()
                    for review in self.corpus.allReviews:
                        for sentence in review.sentences:
                            self.assignAspect(sentence)
                    self.populateAspectWordMat()
                    changed=self.calcChiSq()
                self.corpus.aspectSentences.clear()
                for review in self.corpus.allReviews:
                    for sentence in review.sentences:
                        self.assignAspect(sentence)
                print(self.corpus.aspectKeywords)

            # Saves the object into the given file
            def saveToFile(self,fileName,obj):
                with open(modelDataDir+fileName,'w') as fp:
                    json.dump(obj,fp)
                    fp.close()

        rd = ReadData()
        rd.readAspectSeedWords()
        rd.readStopWords()
        rd.readReviewsFromJson()
        rd.removeLessFreqWords()

        bootstrapObj = BootStrap(rd)
        bootstrapObj.bootStrap()
        bootstrapObj.populateLists()
        bootstrapObj.saveToFile("wList.json",bootstrapObj.wList)
        bootstrapObj.saveToFile("ratingsList.json",bootstrapObj.rating
sList)
        bootstrapObj.saveToFile("reviewIdList.json",bootstrapObj.revie
wIdList)
        bootstrapObj.saveToFile("vocab.json",list(bootstrapObj.corpus.
wordFreq.keys()))
```

```
bootstrapObj.saveToFile("aspectKeywords.json",bootstrapObj.cor
pus.aspectKeywords)
```

Модуль LRR.ipynb

```
import numpy as np
from scipy import optimize
import json
import random
import logging
modelDataDir = "modelData/"

class LRR:
    def __init__(self):
        self.vocab=[]
        self.vocab = self.loadDataFromFile("vocab.json")

        self.aspectKeywords={}
        self.aspectKeywords =
self.loadDataFromFile("aspectKeywords.json")

        #word to its index in the corpus mapping
        self.wordIndexMapping={}
        self.createWordIndexMapping()

        #aspect to its index in the corpus mapping
        self.aspectIndexMapping={}
        self.reverseAspIndexmapping={}
        self.createAspectIndexMapping()

        #list of Wd matrices of all reviews
        self.wList=[]
        self.wList = self.loadDataFromFile("wList.json")

        #List of ratings dictionaries belonging to review
class
        self.ratingsList=[]
```

```
                self.ratingsList =
self.loadDataFromFile("ratingsList.json")

                #List of Review IDs
                self.reviewIdList=[]
                self.reviewIdList =
self.loadDataFromFile("reviewIdList.json")

                #number of reviews in the corpus
                self.R = len(self.reviewIdList)

                #breaking dataset into 3:1 ratio, 3 parts for training
and 1 for testing
                self.trainIndex = random.sample(range(0, self.R),
int(0.75*self.R))
                self.testIndex = list(set(range(0, self.R)) -
set(self.trainIndex))

                #number of aspects
                self.k = len(self.aspectIndexMapping)

                #number of training reviews in the corpus
                self.Rn = len(self.trainIndex)

                #vocab size
                self.n = len(self.wordIndexMapping)

                #delta - is simply a number
                self.delta = 1.0

                #matrix of aspect rating vectors (Sd) of all reviews -
k*Rn
                self.S = np.empty(shape=(self.k, self.Rn),
dtype=np.float64)

                #matrix of alphas (Alpha-d) of all reviews - k*Rn
                #each column represents Aplha-d vector for a review
                self.alpha = np.random.dirichlet(np.ones(self.k),
size=1).reshape(self.k, 1)
```

```
                for i in range(self.Rn-1):
                    self.alpha = np.hstack((self.alpha,
np.random.dirichlet(np.ones(self.k), size=1).reshape(self.k, 1)))

                #vector mu - k*1 vector
                self.mu = np.random.dirichlet(np.ones(self.k),
size=1).reshape(self.k, 1)

                #matrix Beta for the whole corpus (for all aspects,
for all words) - k*n matrix
                self.beta = np.random.uniform(low=-0.1, high=0.1,
size=(self.k, self.n))

                #matrix sigma for the whole corpus - k*k matrix
                #Sigma needs to be positive definite, with diagonal
elems positive
                '''self.sigma = np.random.uniform(low=-1.0, high=1.0,
size=(self.k, self.k))
                self.sigma = np.dot(self.sigma,
self.sigma.transpose())
                print(self.sigma.shape, self.sigma)
                '''

                #Following is help taken from:
                #https://stats.stackexchange.com/questions/124538/
                W = np.random.randn(self.k, self.k-1)
                S = np.add(np.dot(W, W.transpose()),
np.diag(np.random.rand(self.k)))
                D = np.diag(np.reciprocal(np.sqrt(np.diagonal(S))))
                self.sigma = np.dot(D, np.dot(S, D))
                self.sigmaInv=np.linalg.inv(self.sigma)

                # testing for positive semi definite
                # if(np.all(np.linalg.eigvals(self.sigma) > 0)):
#whether is positive semi definite
                #     print("yes")
                # print(self.sigma)

                # setting up logger
```

```
                self.logger = logging.getLogger("LRR")
                self.logger.setLevel(logging.INFO)
                self.logger.setLevel(logging.DEBUG)
                fh = logging.FileHandler("lrr.log")
                formatter = logging.Formatter('%(asctime)s
%(message)s')
                fh.setFormatter(formatter)
                self.logger.addHandler(fh)

            def createWordIndexMapping(self):
                i=0
                for word in self.vocab:
                    self.wordIndexMapping[word]=i
                    i+=1
                #print(self.wordIndexMapping)

            def createAspectIndexMapping(self):
                i=0;
                for aspect in self.aspectKeywords.keys():
                    self.aspectIndexMapping[aspect]=i
                    self.reverseAspIndexmapping[i]=aspect
                    i+=1
                #print(self.aspectIndexMapping)

            def loadDataFromFile(self,fileName):
                with open(modelDataDir+fileName,'r') as fp:
                    obj=json.load(fp)
                    fp.close()
                    return obj

            #given a dictionary as in every index of self.wList,
            #creates a W matrix as was in the paper
            def createWMatrix(self, w):
                W = np.zeros(shape=(self.k, self.n))
                for aspect, Dict in w.items():
                    for word, freq in Dict.items():

W[self.aspectIndexMapping[aspect]][self.wordIndexMapping[word]]=freq
                return W
```

```
        #Computing aspectRating array for each review given Wd->W
matrix for review 'd'
        def calcAspectRatings(self,Wd):
            Sd =
np.einsum('ij,ij->i',self.beta,Wd).reshape((self.k,))
            try:
                Sd = np.exp(Sd)
            except Exception as inst:
                self.logger.info("Exception in calcAspectRatings :
%s", Sd)
            return Sd

        def calcMu(self): #calculates mu for (t+1)th iteration
            self.mu = np.sum(self.alpha, axis=1).reshape((self.k,
1))/self.Rn

        def calcSigma(self, updateDiagonalsOnly): #update diagonal
entries only
            self.sigma.fill(0)
            for i in range(self.Rn):
                columnVec = self.alpha[:, i].reshape((self.k, 1))
                columnVec = columnVec - self.mu
                if updateDiagonalsOnly:
                    for k in range(self.k):
                        self.sigma[k][k] +=
columnVec[k]*columnVec[k]
                else:
                    self.sigma = self.sigma + np.dot(columnVec,
columnVec.transpose())
            for i in range(self.k):
                    self.sigma[i][i] =
(1.0+self.sigma[i][i])/(1.0+self.Rn)
            self.sigmaInv=np.linalg.inv(self.sigma)

        def calcOverallRating(self,alphaD,Sd):
            return np.dot(alphaD.transpose(),Sd)[0][0]

        def calcDeltaSquare(self):
```

```
                self.delta=0.0
                for i in range(self.Rn):
                    alphaD=self.alpha[:,i].reshape((self.k, 1))
                    Sd=self.S[:,i].reshape((self.k, 1))

Rd=float(self.ratingsList[self.trainIndex[i]]["overall"])
                    temp=Rd-self.calcOverallRating(alphaD,Sd)
                    try:
                        self.delta+=(temp*temp)
                    except Exception:
                        self.logger.info("Exception in Delta calc")
                self.delta/=self.Rn

            def maximumLikelihoodBeta(self,x,*args):
                beta = x
                beta=beta.reshape((self.k,self.n))
                innerBracket = np.empty(shape=self.Rn)
                for d in range(self.Rn):
                    tmp = 0.0
                    rIdx = self.trainIndex[d] #review index in wList
                    for i in range(self.k):
                        W = self.createWMatrix(self.wList[rIdx])
                        tmp += self.alpha[i][d]*np.dot(beta[i,
:].reshape((1, self.n)), W[i, :].reshape((self.n, 1)))[0][0]
                    innerBracket[d] = tmp -
float(self.ratingsList[rIdx]["overall"])
                mlBeta=0.0
                for d in range(self.Rn):
                    mlBeta+=innerBracket[d] * innerBracket[d]
                return mlBeta/(2*self.delta)

            def gradBeta(self,x,*args):
                beta=x
                beta=beta.reshape((self.k,self.n))

gradBetaMat=np.empty(shape=((self.k,self.n)),dtype='float64')
                innerBracket = np.empty(shape=self.Rn)
                for d in range(self.Rn):
                    tmp = 0.0
```

```
                    rIdx = self.trainIndex[d] #review index in wList
                    for i in range(self.k):
                        W = self.createWMatrix(self.wList[rIdx])
                        tmp += self.alpha[i][d]*np.dot(beta[i,
:].reshape((1, self.n)), W[i, :].reshape((self.n, 1)))[0][0]
                    innerBracket[d] = tmp -
float(self.ratingsList[rIdx]["overall"])

                for i in range(self.k):
                    beta_i=np.zeros(shape=(1,self.n))
                    for d in range(self.Rn):
                        rIdx = self.trainIndex[d] #review index in
wList
                        W = self.createWMatrix(self.wList[rIdx])
                        beta_i += innerBracket[d] * self.alpha[i][d] *
W[i, :]
                    gradBetaMat[i,:]=beta_i
                return gradBetaMat.reshape((self.k*self.n, ))

            def calcBeta(self):
                beta, retVal,
flags=optimize.fmin_l_bfgs_b(func=self.maximumLikelihoodBeta,x0=self.
beta,fprime=self.gradBeta,args=(),m=5,maxiter=20000)
                converged = True
                if flags['warnflag']!=0:
                    converged = False
                self.logger.info("Beta converged : %d",
flags['warnflag'])
                return beta.reshape((self.k,self.n)), converged

            def maximumLikelihoodAlpha(self, x, *args):
                alphad=x
                alphad=alphad.reshape((self.k, 1))
                rd,Sd,deltasq,mu,sigmaInv=args
                temp1=(rd-np.dot(alphad.transpose(),Sd)[0][0])
                temp1*=temp1
                temp1/=(deltasq*2)
                temp2=(alphad-mu)
```

```
temp2=np.dot(np.dot(temp2.transpose(),sigmaInv),temp2)[0][0]
                temp2/=2
                return temp1+temp2

            def gradAlpha(self, x,*args):
                alphad=x
                alphad=alphad.reshape((self.k, 1))
                rd,Sd,deltasq,mu,sigmaInv=args
                temp1=(np.dot(alphad.transpose(),Sd)[0][0]-rd)*Sd
                temp1/=deltasq
                temp2=np.dot(sigmaInv,(alphad-mu))
                return (temp1+temp2).reshape((self.k,))

            def calcAlphaD(self,i):
                alphaD=self.alpha[:,i].reshape((self.k,1))
                rIdx = self.trainIndex[i]
                rd=float(self.ratingsList[rIdx]["overall"])
                Sd=self.S[:,i].reshape((self.k,1))
                Args=(rd,Sd,self.delta,self.mu,self.sigmaInv)
                bounds=[(0,1)]*self.k
                #self.gradf(alphaD, *Args)
                alphaD, retVal,
flags=optimize.fmin_l_bfgs_b(func=self.maximumLikelihoodAlpha,x0=alph
aD,fprime=self.gradAlpha,args=Args,bounds=bounds,m=5,maxiter=20000)
                converged = True
                if flags['warnflag']!=0:
                    converged = False
                self.logger.info("Alpha Converged : %d",
flags['warnflag'])
                #Normalizing alphaD so that it follows dirichlet
distribution
                alphaD=np.exp(alphaD)
                alphaD=alphaD/(np.sum(alphaD))
                return alphaD.reshape((self.k,)), converged

            '''
            def getBetaLikelihood(self):
                likelihood=0
```

```
                return
self.lambda*np.sum(np.einsum('ij,ij->i',self.beta,self.beta))
            '''

            def dataLikelihood(self):
                likelihood=0.0
                for d in range(self.Rn):
                    rIdx = self.trainIndex[d]
                    Rd=float(self.ratingsList[rIdx]["overall"])
                    W=self.createWMatrix(self.wList[rIdx])
                    Sd=self.calcAspectRatings(W).reshape((self.k, 1))
                    alphaD=self.alpha[:,d].reshape((self.k, 1))
                    temp=Rd-self.calcOverallRating(alphaD,Sd)
                    try:
                        likelihood+=(temp*temp)
                    except Exception:
                        self.logger.debug("Exception in
dataLikelihood")
                likelihood/=self.delta
                return likelihood

            def alphaLikelihood(self):
                likelihood=0.0
                for d in range(self.Rn):
                    alphad=self.alpha[:,d].reshape((self.k, 1))
                    temp2=(alphad-self.mu)

temp2=np.dot(np.dot(temp2.transpose(),self.sigmaInv),temp2)[0]
                    likelihood+=temp2
                try:
                    likelihood+=np.log(np.linalg.det(self.sigma))
                except FloatingPointError:
                    self.logger.debug("Exception in alphaLikelihood:
%f", np.linalg.det(self.sigma))
                return likelihood

            def calcLikelihood(self):
                likelihood=0.0
                likelihood+=np.log(self.delta) #delta likelihood
```

```
            likelihood+=self.dataLikelihood() #data likelihood -
will capture beta likelihood too
            likelihood+=self.alphaLikelihood() #alpha likelihood
            return likelihood

        def EStep(self):
            for i in range(self.Rn):
                rIdx = self.trainIndex[i]
                W=self.createWMatrix(self.wList[rIdx])
                self.S[:,i]=self.calcAspectRatings(W)
                alphaD, converged = self.calcAlphaD(i)
                if converged:
                    self.alpha[:,i]=alphaD
                self.logger.info("Alpha calculated")

        def MStep(self):
            likelihood=0.0
            self.calcMu()
            self.logger.info("Mu calculated")
            self.calcSigma(False)
            self.logger.info("Sigma calculated : %s " %
np.linalg.det(self.sigma))
            likelihood+=self.alphaLikelihood() #alpha likelihood
            self.logger.info("alphaLikelihood calculated")
            beta,converged=self.calcBeta()
            if converged:
                self.beta=beta
            self.logger.info("Beta calculated")
            likelihood+=self.dataLikelihood() #data likelihood -
will capture beta likelihood too
            self.logger.info("dataLikelihood calculated")
            self.calcDeltaSquare()
            self.logger.info("Deltasq calculated")
            likelihood+=np.log(self.delta) #delta likelihood
            return likelihood

        def EMAlgo(self, maxIter, coverge):
            self.logger.info("Training started")
            iteration = 0
```

```
            old_likelihood = self.calcLikelihood()
            self.logger.info("initial calcLikelihood calculated,
det(Sig): %s" % np.linalg.det(self.sigma))
            diff = 10.0
            while(iteration<min(8, maxIter) or (iteration<maxIter
and diff>coverge)):
                self.EStep()
                self.logger.info("EStep completed")
                likelihood = self.MStep()
                self.logger.info("MStep completed")
                diff = (old_likelihood-likelihood)/old_likelihood
                old_likelihood=likelihood
                iteration+=1
            self.logger.info("Training completed")

        def testing(self):
            for i in range(self.R-self.Rn):
                rIdx = self.testIndex[i]
                W = self.createWMatrix(self.wList[rIdx])
                Sd = self.calcAspectRatings(W).reshape((self.k,1))
                overallRating = self.calcOverallRating(self.mu,Sd)
                print("ReviewId-",self.reviewIdList[rIdx])
                print("Actual
OverallRating:",self.ratingsList[rIdx]["overall"])
                print("Predicted OverallRating:",overallRating)
                print("Actual vs Predicted Aspect Ratings:")
                for aspect, rating in
self.ratingsList[rIdx].items():
                    aspect_predict = []
                    aspect_value = []
                    if aspect != "overall" and aspect.lower() in
self.aspectIndexMapping.keys():
                        r =
self.aspectIndexMapping[aspect.lower()]
                        print("Aspect:",aspect," Rating:",rating,
"Predic:", Sd[r])
                if overallRating > 3.0:
                    print("Positive Review")
                else:
```

```
                    print("Negative Review")

np.seterr(all='raise')
lrrObj = LRR()
lrrObj.EMAlgo(maxIter=1, coverge=0.001)
lrrObj.testing()
```

Модуль lstm.ipynb

```
import tensorflow as tf
import pandas as pd
import numpy as np
import keras
import math
import os
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

train_data = pd.read_csv('train_data.csv')
valid_data = pd.read_csv('valid_data.csv')
test_data = pd.read_csv('test_data.csv')

train_data['Comment'][0]

def data_stats(dataframe):

    s = 0.0
    pos = 0

    for i in dataframe['Comment']:
        word_list = i.split()
        s = s + len(word_list)
    print("Total revievs: ", dataframe.shape[0])
    print("Average length of each review :
",s/train_data.shape[0])
```

```
            try:
                for i in range(dataframe.shape[0]):
                    if dataframe.iloc[i]['Sentiment'] == 1:
                        pos = pos + 1
                neg = dataframe.shape[0] - pos
                print("Percentage of reviews with positive sentiment:
"+str(pos/dataframe.shape[0]*100)+"%")
                print("Percentage of reviews with negative sentiment:
"+str(neg/dataframe.shape[0]*100)+"%")

            except:
                print("No sentiment in test data")

        data_stats(train_data)

        data_stats(valid_data)

        data_stats(test_data)

        X_train = train_data['Comment']
        X_valid = valid_data['Comment']
        X_test = test_data['Comment']

        # y_train = train_data['Sentiment']
        # y_valid = valid_data['Sentiment']

        y_train = train_data['Rating']
        y_valid = valid_data['Rating']

        num_classes = 11 #amount of numbers from 0 to 9

        y_train = tf.keras.utils.to_categorical(y_train, num_classes)
        y_valid = tf.keras.utils.to_categorical(y_valid, num_classes)

        vocab_size = 1000 # choose based on statistics
        oov_tok = ''
        embedding_dim = 128
```

```
      max_length = 300 # choose based on statistics, for example 150
to 200
      padding_type='post'
      trunc_type='post'
      # tokenize sentences
      tokenizer = Tokenizer(num_words = vocab_size,
oov_token=oov_tok)
      tokenizer.fit_on_texts(X_train)
      word_index = tokenizer.word_index
      # convert train dataset to sequence and pad sequences
      train_sequences = tokenizer.texts_to_sequences(X_train)
      train_padded = pad_sequences(train_sequences, padding='post',
maxlen=max_length)
      # convert Test dataset to sequence and pad sequences
      valid_sequences = tokenizer.texts_to_sequences(X_valid)
      valid_padded = pad_sequences(valid_sequences, padding='post',
maxlen=max_length)

      model = keras.Sequential([
          keras.layers.Embedding(vocab_size, embedding_dim,
input_length=max_length),
          # keras.layers.Bidirectional(keras.layers.LSTM(64,
return_sequences=True)),
          keras.layers.LSTM(64),
          keras.layers.Dense(32, activation='relu'),
          keras.layers.Dense(11, activation='sigmoid')
      ])
      # compile model
      optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
      model.compile(loss =
tf.keras.losses.CategoricalCrossentropy(),
                    optimizer=optimizer,
                    metrics=['accuracy'])
      # model summary
      model.summary()

      tb_callback = tf.keras.callbacks.TensorBoard(log_dir="logs/",
histogram_freq=1)
```

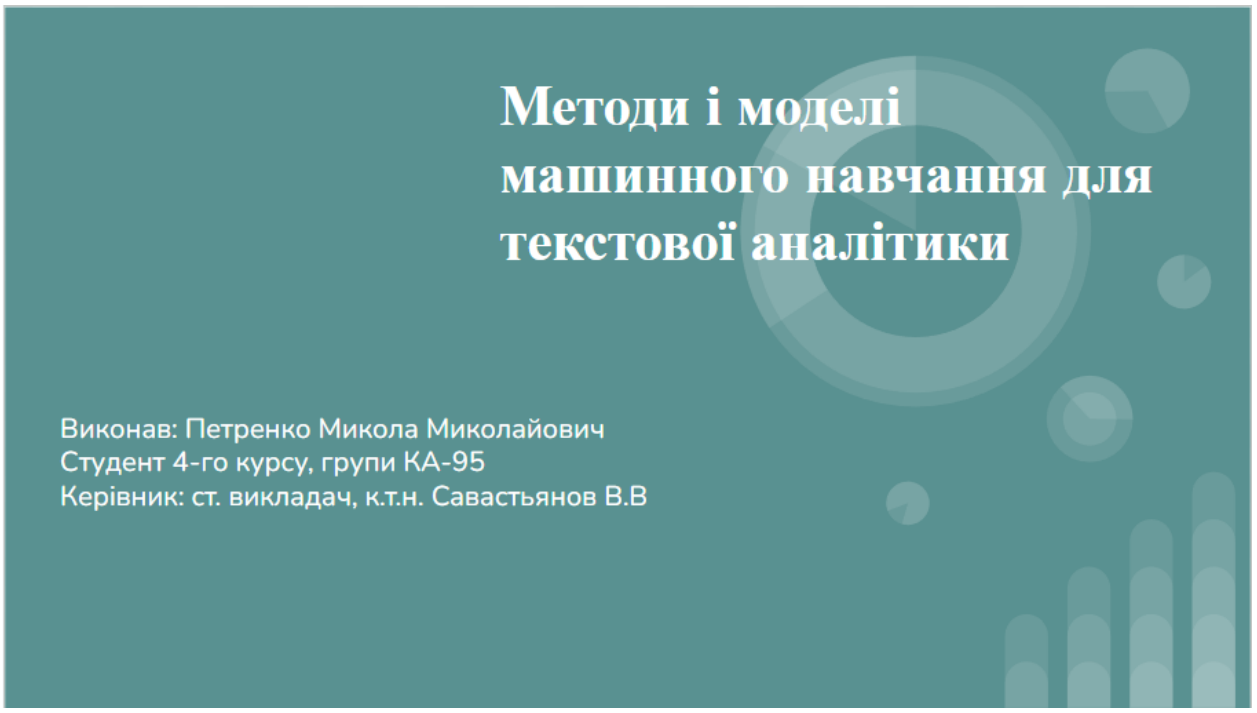
```
      checkpoint_path = "training/cp-{epoch:04d}.ckpt"
      checkpoint_dir = os.path.dirname(checkpoint_path)
      cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath =
checkpoint_path, save_weights_only=True, verbose=1)

      num_epochs = 5
      history = model.fit(train_padded, y_train,
                          epochs=num_epochs, verbose=1, callbacks =
[tb_callback, cp_callback],
                         validation_data = (valid_padded, y_valid))

      prediction = model.predict(valid_padded)
      for i in range(len(prediction)):
          prediction[i] = np.where(prediction[i] <
max(prediction[i]), 0, 1)
      print("Accuracy of validation on test set : ",
accuracy_score(y_valid,prediction))

      %load_ext tensorboard
      %tensorboard --logdir './logs' --host localhost
```

ДОДАТОК Б. ГРАФІЧНИЙ МАТЕРІАЛ



Мета та об’єкт роботи

Мета роботи - застосування сучасних методів машинного навчання для аналізу, замір метрик та перевірка точності. Ключова задача - підтвердити, що використання алгоритмів машинного навчання суттєво полегшує задачу аналізу текстів.

Об’єкт роботи - набір даних з оцінками та рецензіями на готелі від відомого сайту tripadvisor.com Цей набір даних містить огляди готелей у форматі .json.

Постановка задачі та методи, що використовуються в роботі

Поставлена задача - на прикладі даних з файлу “reviews.json” натренувати моделі машинного навчання для передбачення оцінки відгуку за аспектами.

В роботі використано один з класичних підходів до рішення задачі, а саме модель на основі LSTM. Як противага до нього використано латентно аспектну регресію.

Набір даних

TripAdvisor — популярна онлайн-платформа, де користувачі можуть ділитися своїм досвідом, оцінками та думками про різні готелі, ресторани, пам’ятки тощо. Набір даних містить таку інформацію.

1. Інформація про готель;
2. Текст відгуку;
3. Оцінки відгуків;
4. Дата відгуку;
5. Інформація про рецензента;

Модель на основі LSTM

LSTM є варіантом рекурентної нейронної мережі (RNN), яка здатна зберігати та використовувати інформацію про довготривалі залежності в послідовностях.

Переваги LSTM:

- послідовне моделювання;
- збереження пам'яті;
- обробка даних змінної довжини;
- вилучення функцій;
- передача навчання.

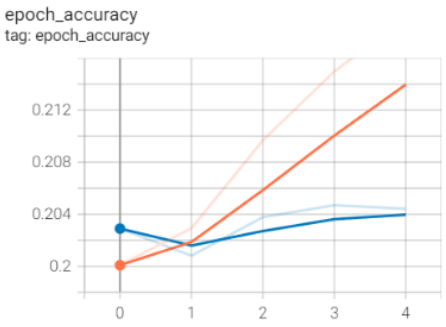
Недоліки LSTM:

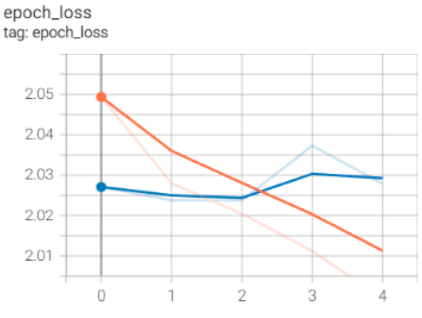
- ресурсозатратні розрахунки;
- перенавчання;
- відсутність інтерпретації;
- робота поза словником.

Метрики тренування LSTM

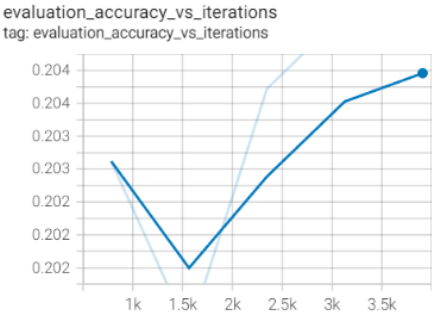
№ епохи	Тренувальні втрати	Перевірочні втрати	Тренувальна точність	Перевірочна точність
1	2.0435	2.0315	0.2019	0.2033
2	2.0276	2.0224	0.2049	0.2041
3	2.0161	2.0323	0.2060	0.2018
4	2.0002	2.0362	0.2207	0.2021
5	1.9810	2.0559	0.2243	0.2031

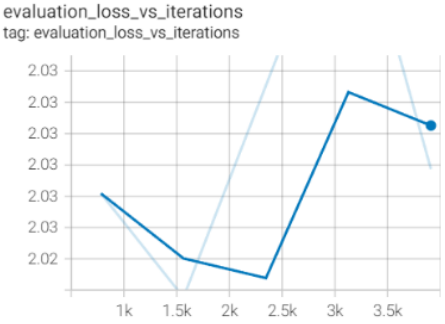
Графіки зміни метрик моделі за епохами





Графіки зміни метрик моделі за ітераціями





Латентно аспектна регресія

Латентна аспектна регресія (LAR) — це метод статистичного моделювання, який використовується в машинному навчанні та обробці природної мови (NLP) для виявлення прихованих аспектів у наборі спостережуваних змінних.

Переваги LAR:

- аналіз на рівні аспекту;
- розкриття прихованих аспектів;
- легка інтерпретація;
- покращений вибір функцій.

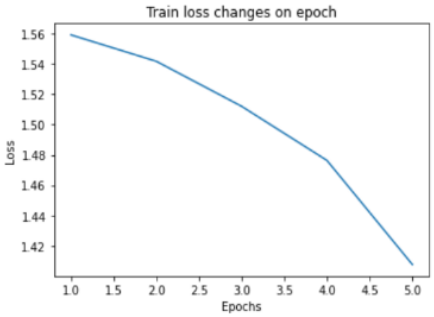
Недоліки LAR:

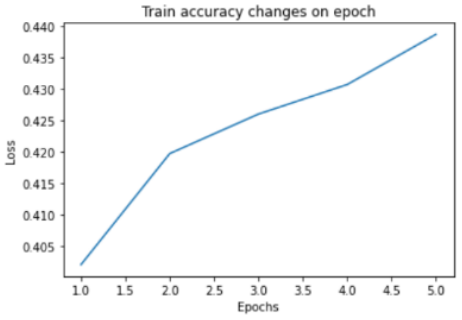
- залежність від аспектів;
- обмежене узагальнення;
- комплексна реалізація;
- якість даних.

Метрики тренування LAR

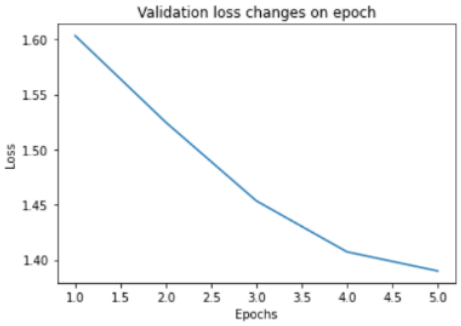
№ епохи	Тренувальні втрати	Перевірочні втрати	Тренувальна точність	Перевірочна точність
1	1.5591	1.6032	0.4020	0.4001
2	1.5417	1.5247	0.4197	0.4221
3	1.5120	1.4534	0.4260	0.4230
4	1.4763	1.4072	0.4307	0.4294
5	1.4077	1.3899	0.4387	0.4402

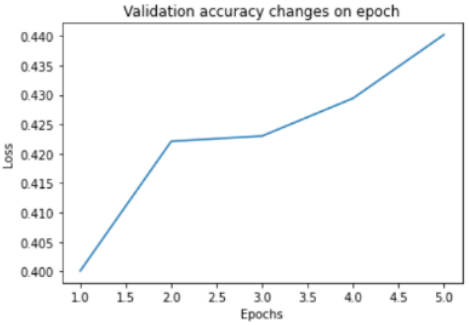
Графіки зміни тренувальних метрик моделі за епохами





Графіки зміни перевірочних метрик моделі за епохами





Висновки

В роботі, безпосередньо, розглянуто один із напрямків аналізу тексту, а саме аналіз настроїв (або аналіз тональності тексту). Основна мета такого аналізу - виявлення в текстах емоційно забарвленої лексики і емоційної оцінки авторів (думок) по відношенню до об'єктів, мова про які йде в тексті.

Після створення, тренування, оцінювання моделей та аналізу метрик, отриманих під час тренування можна відмітити, що підхід з використанням латентно аспектної регресії виявився помітно результативнішим за класичний.

Актуальність даної роботи заключається в дослідженні відносно нової техніки до розв'язку задач аналізу настроїв з метою оцінки результатів. У подальшому отримані результати можна використовувати при виборі методу для вирішення подібних задач.

Дякую за увагу