КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра прикладної статистики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 124 Системний аналіз
на тему:

АВТОМАТИЧНА КЛАСИФІКАЦІЯ ТА ТЕГІЗАЦІЯ РЕСТОРАННИХ ВІДГУКІВ

Виконав студент 4-го курсу
Таранюк Дмитро Юрійович

(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Махно Михайло Федорович

(підпис)

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

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри прикладної статистики
«05» червня 2023 р.,
протокол № 11
Завідувач кафедри
Розора І.В.

(підпис)

Київ – 2023

РЕФЕРАТ

Реферат до роботи "Автоматична класифікація та тегізація ресторанних відгуків" студента 4 курсу Таранюка Дмитра Юрійовича, наукового керівника асистента, кандидата технічних наук Махно М.Ф.

Робота складається з 59 сторінок, містить 7 ілюстрацій та 1 таблицю. Використано 9 джерел. Додатків до роботи 2.

Ключові слова: АВТОМАТИЧНА КЛАСИФІКАЦІЯ, ТЕГІЗАЦІЯ, РЕСТОРАННІ ВІДГУКИ, НЕЙРОННІ МЕРЕЖІ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, МОДЕЛЬ BERT, NLP, АНАЛІЗ ТЕКСТУ, ОПТИМІЗАЦІЯ.

Об'єкт дослідження: аналіз текстових відгуків за допомогою штучного інтелекту.

Мета роботи: розробка системи автоматичної класифікації та тегізації ресторанних відгуків.

Методи та інструменти дослідження: використання нейронних мереж та моделі BERT для аналізу тексту.

Результати та їх новизна: розроблено систему, яка дозволяє автоматично класифікувати та тегувати відгуки в ресторанному бізнесі, що значно покращує процес аналізу відгуків.

Інформація щодо впровадження: система може бути впроваджена в ресторанний бізнес для покращення процесу аналізу відгуків.

Взаємозв’язок з іншими роботами: робота базується на сучасних методах аналізу текстових відгуків та використанні нейронних мереж.

Рекомендації щодо використання результатів роботи: результати роботи можуть бути використані для покращення процесу аналізу відгуків в ресторанному бізнесі, що може покращити якість обслуговування та збільшити задоволеність клієнтів.

Сфера застосування: ресторанний бізнес, аналіз відгуків.

Значимість роботи: робота важлива для сучасного світу, де інформація стала основним товаром і важливим ресурсом. Здатність аналізувати та обробляти великі обсяги текстової інформації стала не просто потребою, а вирішальним фактором успіху.

Висновки та пропозиції щодо розвитку об’єкта дослідження (розроблення) та доцільності продовження досліджень або розробок: в майбутньому, такий підхід можна буде розглянути, наприклад, для написання Магістерської кваліфікаційної роботи.

ЗМІСТ

ВСТУП

Визначення проблеми і її актуальності

У сучасному світі, де інформація стала основним товаром і важливим ресурсом, здатність аналізувати та обробляти великі обсяги текстової інформації стала не просто потребою, а вирішальним фактором успіху. Саме той хто володіє інформацією – керує світом. Проте для керування світом замало звичайного зберігання інформації, вона потребує аналітики та обробки, й чим більший масив інформації – тим складніші й дорожчі методи її обробки. Сучасні технології дозволяють суттєво зменшити час обробки, при цьому зменшивши фізичні потужності, що дає змогу здешевити загальну вартість аналітики, й при цьому не тільки зберегти, а й навіть покращити її якість. Світ давно звик до живої робочої сили; це аналітики, фінансисти, банкіри, тощо. Але останні роки суттєво змінили суспільне бачення, та бачення бізнесу у питанні збору та обробки великих об’ємів даних. Світ вперше побачив глобальне використання нейронних мереж(CNN[1]) та штучного інтелекту(AI[2]), раніше такі методи впроваджували точково, загалом у великих корпораціях. Зараз же, можна виявити тенденцію, що популярність використання таких методів значно зросла, й все більше компаній довіряють аналітику своїх даних штучному інтелекту. У тому числі, й бізнес сфери обслуговування.

[1] CNN (Convolutional Neural Network) — це тип нейронної мережі для аналізу зображень, використовуючи згорткові операції для виявлення візуальних ознак.
[2]AI (Artificial Intelligence) — галузь, що створює системи, здатні до розумової активності, навчання, розуміння та прийняття рішень, наближених до людського інтелекту.

Важливість аналізу відгуків в інтернеті набуває все більшої значущості у сучасному світі. Користувачі активно залишають відгуки про різні товари, послуги та досвід, що їх сприйняття. Ці відгуки стають цінною джерелом інформації для компаній, оскільки вони можуть допомогти зрозуміти, що функціонує добре і що потребує покращення.

Зокрема, у сфері ресторанного бізнесу відгуки користувачів мають вирішальне значення. Потенційні клієнти перед походом до ресторану часто переглядають відгуки, щоб отримати враження про якість обслуговування, смак страв та загальну атмосферу. Аналіз ресторанних відгуків дозволяє власникам і менеджерам ресторанів не лише зрозуміти задоволення клієнтів, але й ідентифікувати сильні та слабкі сторони свого закладу.

Штучний інтелект (AI) та нейронні мережі (CNN) стають незамінними інструментами для аналізу великого обсягу ресторанних відгуків. Вони здатні автоматично обробляти текст та виявляти ключові ознаки, такі як згадки про конкретні страви, якість обслуговування, атмосферу тощо. Це дозволяє швидко здобути загальну картину задоволення клієнтів і виявити тенденції.

Ціль дослідження та завдання

У рамках дослідження ставиться завдання розробити ефективний метод для аналізу потоку ресторанних відгуків, що поступають в інтернеті. Оскільки відгуки можуть бути дуже різноманітними та великими за обсягом, важливо мати систему, яка здатна автоматично обробляти цей потік і виділяти корисну інформацію. Метою дослідження є з'ясування ефективних та точних методів аналізу відгуків, що

допоможуть ресторанам у розумінні задоволення клієнтів, виявленні слабких моментів та вдосконаленні своєї діяльності.

Завдання полягає у створенні програми, яка за допомогою штучного інтелекту класифікує та тегує відгуки. Одним з основних завдань дослідження є розробка програми, що використовує штучний інтелект для класифікації та тегування ресторанних відгуків. Ця програма має здатність автоматично аналізувати та розпізнавати ключові аспекти відгуків, такі як якість їжі, обслуговування, атмосфера тощо. Вона буде використовувати алгоритми машинного навчання та нейронні мережі, щоб класифікувати відгуки за певними категоріями та визначати їхню емоційну забарвленість.

Об'єкт і предмет дослідження

Об’єктом дослідження є масив даних, що містять у собі ресторанні відгуки. Об’єкт дослідження в цьому контексті означає реальний об’єкт або явище, яке ми намагаємося зрозуміти, оцінити або інтерпретувати. В даному випадку, це масив даних, що складається з відгуків про ресторани.

Предмет дослідження це обробка таких даних, починаючи від їх попередньої обробки, тобто очищення від “сміття”. Предмет дослідження в даному контексті є процес або методи, які ми використовуємо для вивчення об’єкта. Очищення даних - це перший крок в обробці даних, який включає видалення невідповідних, невірних, неповних або непотрібних даних з нашого масиву. "Сміття" може включати в себе шум, такий як нестатистичні відхилення, або помилки, такі як невірні значення або опечатки.

Дослідження також зосереджено на зведенні даних до однієї мови. Це важливий етап, особливо коли досліджуються відгуки з різних країн або регіонів, де відгуки можуть бути написані на різних мовах. Також це дозволяє полегшити роботу штучного інтелекту, адже йому доведеться тренуватися лише на одній конкретній мові.

Обробка даних також включає форматування. Це може означати перетворення даних в більш використовуваний формат або структуру, щоб полегшити їх аналіз, або ж для легшого сприйняття даних машиною, де, наприклад формат .txt перетворюється у формат .csv, а емоційні забарвлення перетворюються у одиниці та нулі.

Дослідження завершується аналізом штучного інтелекту. Штучний інтелект (AI) може використовувати різні методи для аналізу цих відгуків, включаючи машинне навчання та обробку природної мови. Ці методи можуть включати вивчення сентименту для визначення позитивного або негативного тону відгуку, класифікацію відгуків на основі різних аспектів ресторанного досвіду, або виявлення шаблонів або тенденцій у відгуках.

Методи дослідження

Ця робота базується на комплексному підході до дослідження, який включає в себе застосування різноманітних методів аналізу. У процесі розробки основного методу було проведено експерименти з різними техніками, включаючи поляризацію ключових слів на емоційний відтінок, яка, хоч і не дала очікуваних результатів, була важливим етапом під час вивчення характеристик даних.

Автоматичний переклад був включений в процес як інструмент для уніфікації даних, що дозволяло привести всі відгуки до однієї мови та полегшити подальшу їх обробку.

Основний акцент було зроблено на застосуванні штучного інтелекту для аналізу даних. Використовуючи модель BERT[3], було проведено глибокий аналіз відгуків, змодельовані на основі попередньо підготовлених даних. Цей підхід дозволив отримати робочу модель, яка розуміє контекст відгуку, та видає стабільний результат.

Наукова новизна та практична значимість

Наукова новизна цього дослідження полягає в комплексному підході до аналізу відгуків ресторанів, який включає використання передових технологій штучного інтелекту. Особливо цінною є інтеграція моделі BERT, яка відкриває нові можливості для глибокого аналізу природної мови. Також, дана робота пропонує унікальну систему попередньої обробки даних та автоматичного перекладу, що спрямовані на ефективне зведення відгуків до однієї мови.

[3] BERT (Bidirectional Encoder Representations from Transformers) - це модель природної мови, що використовується для розуміння тексту, заснована на трансформерній архітектурі та навчана задачами заповнення пропусків та передбачення наступного речення. Вона досягає високої якості завдяки урахуванню контексту та використовується в багатьох завданнях обробки мови.

1. НЕЙРОННІ МЕРЕЖІ ТА МОДЕЛЬ BERT

1.1 Огляд сучасних методів аналізу текстових відгуків

Сучасні методи аналізу текстових відгуків включають широкий спектр підходів, які змінюються від простих текстових обробок до більш складних і розширених технік, заснованих на штучному інтелекті.

Базові методики аналізу тексту включають сегментацію тексту (або токенізацію), видалення стоп-слів, стеммінг та лематизацію. Стеммінг - це процес скорочення слів до їх кореня, видаляючи суфікси, а лематизація перетворює слово на його базову (або словникову) форму. Такі методи є важливими для попередньої обробки тексту перед застосуванням складніших аналітичних методів.

З іншого боку, більш складні методи засновані на машинному навчанні та глибокому навчанні. Моделі машинного навчання, такі як наївний байєсівський класифікатор або метод опорних векторів (SVM), використовуються для класифікації тексту на основі наявності певних слів або фраз. Однак ці методи зазвичай не враховують контекст.

Цей дефіцит подолали моделі глибокого навчання, які використовують нейронні мережі для моделювання складних взаємозв'язків між словами в тексті. Одним з найбільш відомих прикладів є трансформатори, які використовують механізм уваги для кращого розуміння контексту слова на основі його відносин з іншими словами в тексті.

Особливою розробкою є модель BERT (Bidirectional Encoder Representations from Transformers), яка використовує двонаправлену структуру та може ефективно моделювати відносини між словами в контексті, що робить її особливо потужною для розуміння сутності тексту.

Також існують інші методи, які використовують знання з області лінгвістики та психології для аналізу тексту, наприклад, сентимент-аналіз, який намагається визначити емоційний відтінок тексту, чи то позитивний, нейтральний чи негативний.

1.2 Введення в теорію нейронних мереж

Нейронні мережі є фундаментальною частиною сучасного глибокого навчання та штучного інтелекту. Їх ідея походить від відображення структур та принципів роботи біологічних нейронних мереж, які знаходяться в мозку живих істот, та їх подальшого використання для моделювання аналогічних систем в комп'ютерних технологіях.

Центральною концепцією нейронних мереж є ідея "шарів", що складаються з множини вузлів або "нейронів". Кожний окремий нейрон у такому шарі відповідає за приймання вхідних даних, їх подальшу обробку за допомогою специфічної вагової функції та активацію за допомогою відповідної функції активації. Після цього результат передається наступному шару для подальшої обробки.

В структурі нейронних мереж можна виділити три основних типи шарів: вхідний шар, один або кілька прихованих шарів та вихідний шар. Вхідний шар приймає початкові дані для обробки, приховані шари виконують більшість обчислювальних операцій, а вихідний шар

представляє кінцевий результат обробки даних. У моделях глибокого навчання кількість прихованих шарів може досягати великих значень, що дозволяє моделі вивчити більш глибокі та складні взаємозв'язки у вхідних даних.

Основний механізм навчання нейронних мереж базується на алгоритмі зворотного розповсюдження помилки. В процесі його роботи відбувається оцінка різниці між поточним вихідним значенням мережі та бажаним вихідним значенням (відповідною міткою навчання). Потім ця "помилка" розповсюджується в зворотному напрямку через мережу, що призводить до корекції ваг нейронів.

У сучасному світі нейронні мережі знаходять застосування в найрізноманітніших областях, включаючи такі завдання як класифікація зображень, розпізнавання мови, аналіз тексту та багато інших. Вони стали невід'ємною частиною більшості систем штучного інтелекту, які використовуються сьогодні, та продовжують активно розвиватися і вдосконалюватися.

Перші нейронні мережі були створені у 1950-х і 1960-х роках як спроби моделювання біологічних процесів в мозку. Наприклад, "Перцептрон" Френка Розенблатта, розроблений у 1958 році, був одним з перших моделей, що використовували основні принципи нейронних мереж, які ми знаємо сьогодні. Однак нейронні мережі зазнали справжнього прориву з початком ери глибокого навчання в 2010-х роках. Існують численні відомі приклади успішного використання нейронних мереж:

а) Розпізнавання зображень: Це, мабуть, одне з найвідоміших використань нейронних мереж. Модель нейронної мережі "AlexNet", створена Олексом Крижевським, виграла змагання ImageNet Large Scale

Visual Recognition Challenge у 2012 році, що стало важливим моментом в розвитку глибокого навчання.

b) Google Translate: Google використовує нейронні мережі для вдосконалення своєї системи перекладу. Система нейронного машинного перекладу, вперше представлена в 2016 році, здатна аналізувати цілі речення, а не просто окремі слова, що поліпшує якість перекладу.

c) Генерація тексту: GPT-3/GPT-4, модель глибокого навчання, розроблена OpenAI, використовує нейронні мережі для генерації тексту, що здатний написати все від есе до віршів.

d) AlphaGo: Цей проект від Google DeepMind став першою штучною інтелектуальною системою, що перемогла світового чемпіона в грі Go. AlphaGo використовує нейронні мережі для аналізу можливих ходів та визначення найкращого стратегії.

Ці приклади показують, наскільки потужними і гнучкими можуть бути нейронні мережі, і як вони можуть бути використані для вирішення широкого спектра завдань.

1.3 Опис моделі BERT та її застосування в NLP

BERT (Bidirectional Encoder Representations from Transformers) використовує дві основні концепції: трансформери (Transformers) та бідирекційний контекстний аналіз (Bidirectional Contextual Analysis).

Трансформери - це моделі глибокого навчання, вперше представлені в роботі "Attention is All You Need" в 2017 році. Вони використовують механізм "самоуваги" (Self-Attention), який дозволяє моделі дивитися на всі слова в реченні одночасно та визначати важливість кожного слова відносно інших.

Бідирекційний контекстний аналіз означає, що BERT вивчає контекст слова, аналізуючи текст, що йде перед і після цього слова. Це відрізняється від традиційних однонаправлених моделей, які аналізують текст лише в одному напрямку.

Тепер розглянемо, як BERT навчається.

Навчання BERT відбувається в два етапи: предтренування (Pre-training) та тонка настройка (Fine-tuning):

а) Предтренування (Pre-training): На цьому етапі BERT вивчається на великому корпусі тексту (наприклад, Wikipedia). Це дає моделі можливість вивчити широку кількість залежностей між словами та контекстом, в якому вони використовуються. Це відбувається за допомогою двох основних методів:

1) Masked Language Modeling (MLM): Близько 15% слів у кожному реченні випадковим чином "маскуються" або приховуються, а BERT намагається передбачити ці слова на основі контексту.

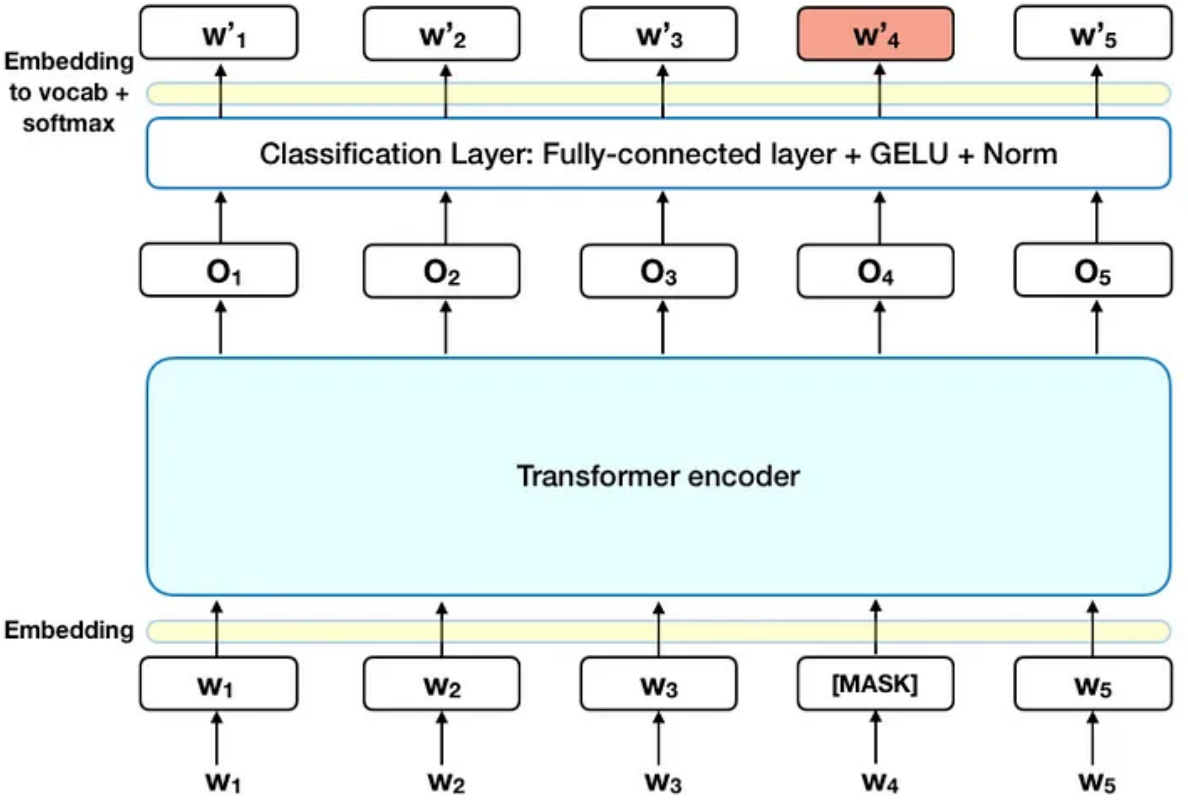

Рисунок 1 – Принцип роботи MLM

2) Next Sentence Prediction (NSP): BERT отримує два речення і навчається передбачати, чи є друге речення безпосереднім продовженням першого.

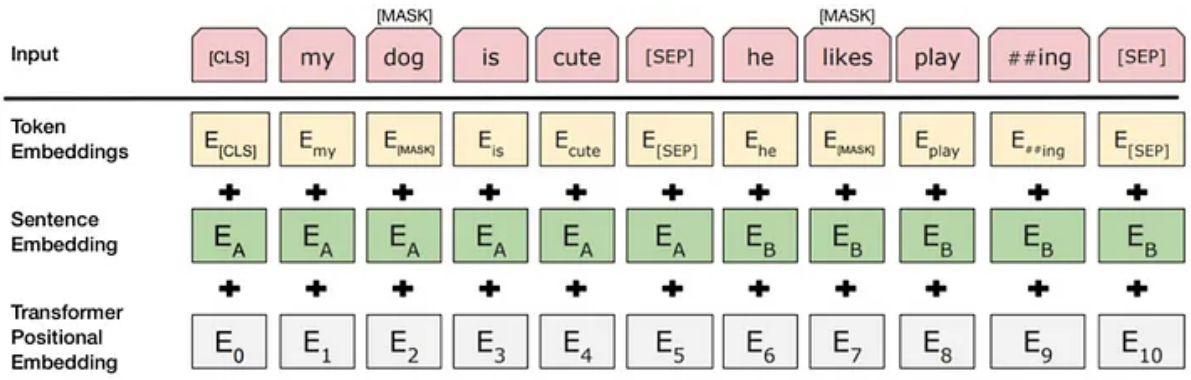

Рисунок 2 – Принцип роботи NSP

b) Тонке налаштування (Fine-tuning): Після предтренування BERT можна налаштувати на конкретну задачу NLP. Це включає навчання на певному наборі даних, специфічному для цієї задачі, із збереженням знань, які модель вже отримала під час предтренування. Ця тонка настройка дозволяє моделі BERT добре працювати на широкому спектрі завдань NLP.

Для того аби підкреслити ефективність використання BERT, можемо поглянути на Рисунок 3. Червоною лінією показана точність при використанні методики “Зліва направо”, коли кожне слово аналізується по черзі, а Синьою лінією показана точність при використанні “маскування”. Очевидна різниця у точності відповідно до використаних методик.

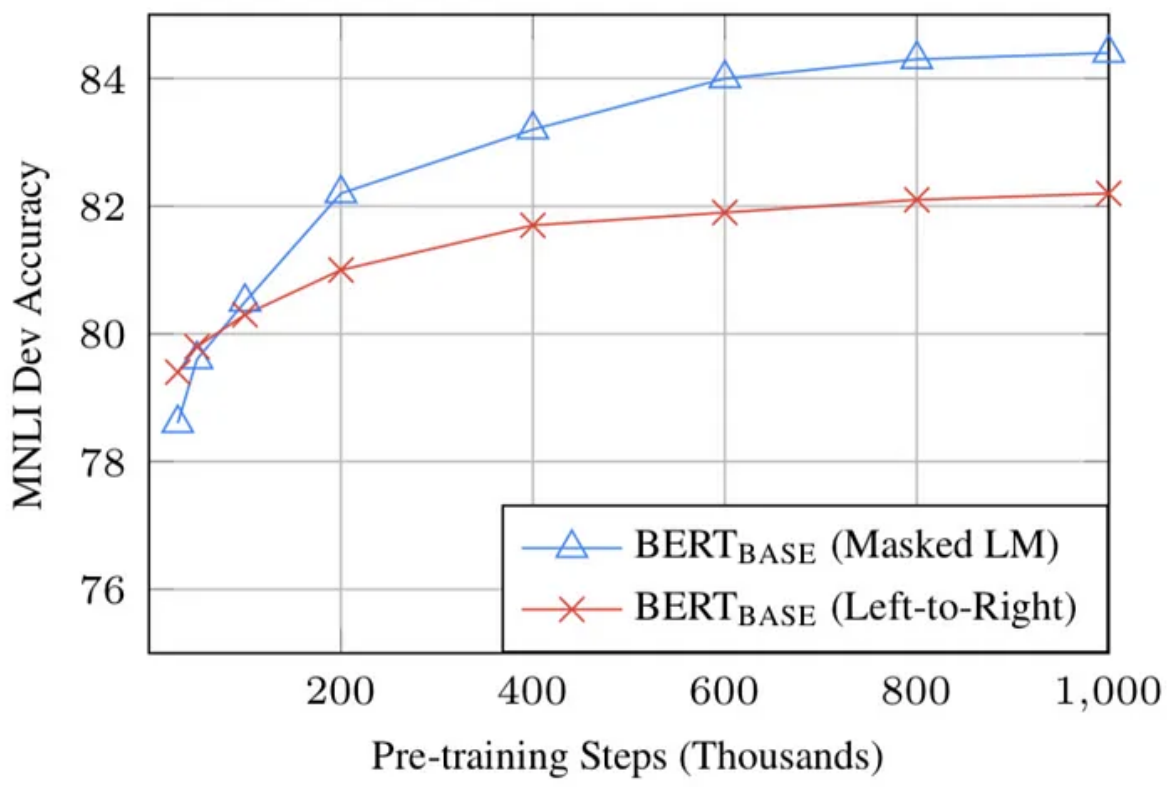

Рисунок 3

Відтоді як BERT був запущений, він знайшов широке застосування у багатьох задачах NLP, зокрема в класифікації тексту, генерації тексту, визначенні настрою, відповідях на запитання, тегуванні частин мови та багатьох інших задачах. Його вплив на галузь NLP був настільки значним, що він став новим стандартом для багатьох сучасних систем обробки мови, змінивши спосіб, яким ми розуміємо та використовуємо машинне навчання для розуміння природної мови.

1.4 Розгляд інших моделей, що використовуються для аналізу відгуків

Існує кілька готових моделей, які можна використовувати для аналізу відгуків. Вони мають свої переваги та недоліки у порівнянні з BERT, для початку розглянемо такі моделі детальніше, ти визначимо, чому вони не є найкращими у контексті цієї роботи.

VADER (Valence Aware Dictionary and sEntiment Reasoner) - це лексичний метод для аналізу настрою тексту, розроблений для англійської мови. Його основна ідея полягає в використанні словників з оцінками настрою та правил для визначення позитивного, негативного або нейтрального значень відгуку.

VADER враховує не тільки окремі слова, але й фрази, що можуть мати емоційне забарвлення. Він використовує словники, які містять попарні асоціації слів з оцінками настрою (наприклад, "добрий" - позитивний, "поганий" - негативний). Кожне слово відгуку оцінюється за його позитивністю, негативністю, об'єктивністю та інтенсивністю.

Окрім лексичного аналізу, VADER також використовує правила, які враховують контекст і синтаксичні особливості, щоб визначити точніше настрій тексту. Він, наприклад, враховує підсилення (наприклад, "дуже добрий"), заперечення (наприклад, "не поганий") та інші лінгвістичні особливості.

VADER повертає комплексний настрій тексту, включаючи загальну позитивність, негативність та нейтральність, а також компаунд-показник, який представляє сумарну оцінку настрою тексту від -1 (негативний) до +1 (позитивний).

Перевагою VADER є його простота в застосуванні та ефективність у розпізнаванні загального настрою тексту. Він може бути особливо корисним для швидкого аналізу великих обсягів тексту, таких як відгуки з соціальних медіа або веб-сторінок.

Однак, варто враховувати, що VADER був розроблений для англійської мови і може не давати такої точності для інших мов. Також він не враховує семантичні зв'язки між словами та не дозволяє виявляти відносно складні відтінки настрою.

TextBlob - це бібліотека для обробки тексту в мові Python, яка включає модуль для аналізу настрою тексту. Цей модуль використовує комбінацію методів машинного навчання та правил, щоб визначити позитивний, негативний або нейтральний настрій відгуку.

TextBlob використовує базу даних словних оцінок (словники з оцінками настрою) для призначення оцінок окремим словам у тексті. Він також

враховує контекст, використовуючи аналіз граматики та синтаксичних залежностей, щоб визначити настрій речень і фраз.

Модуль для аналізу настрою TextBlob повертає показники позитивності, негативності та об'єктивності тексту. Позитивність і негативність вимірюються на шкалі від 0 до 1, де 0 вказує на негативний настрій, 1 - на позитивний настрій, а значення навколо 0,5 вказують на нейтральний настрій. Об'єктивність вимірюється на шкалі від 0 до 1, де 0 вказує на суб'єктивний текст, а 1 - на об'єктивний текст.

TextBlob є простим у використанні і може бути швидким в аналізі великих обсягів тексту. Проте, враховуйте, що його точність може варіюватись в залежності від мови та контексту.

Обидві моделі доволі прості у використанні, особливо у порівнянні зі штучним інтелектом, який потребує тонкого налаштування. Але така простота оплачується нищою точністю. Очевидно, що розгляд окремих слів, чи навіть фраз не є настільки ж фундаментальним підходом, як розуміння контексту та настрою кожного окремого відгука.

Але варто зазначити, що впродовж розробки практичної частини даної роботи, на одному з етапів був експеримент з впровадження для аналізу моделі TextBlob. Ця модель хоч і не потребувала попереднього локального навчання на власних масивах даних, але потребувала ретельно підготовлених вхідних даних для кращої точності. Для використання TextBlob необхідно провести нормалізацію текста.

Нормалізація тексту - це процес перетворення текстового вмісту на стандартизований, уніфікований формат, що полегшує подальшу обробку

та аналіз. Цей процес включає в себе ряд кроків, спрямованих на усунення непотрібних чи несуттєвих елементів, стандартизацію форматування, редагування або видалення символів, а також нормалізацію різних видів текстових даних.

Основні кроки нормалізації тексту включають:

a) Приведення до одного регістру: Текст може бути перетворений у верхній, нижній або перша заголовна літера у кожному слові, для забезпечення уніфікації регістру.

b) Токенізація: Розбиття тексту на окремі токени або слова. Це дозволяє розібрати текст на окремі одиниці для подальшої обробки.

c) Видалення пунктуації: Видалення символів пунктуації, таких як крапки, коми, лапки тощо, які можуть нести обмежену семантичну інформацію.

d) Видалення зайвих символів: Видалення непотрібних символів, наприклад, символів нового рядка, зайвих пропусків, спеціальних символів, номерів телефонів тощо.

e) Лематизація або стемінг: Приведення слів до їх базової форми (леми) або відсічення зайвих суфіксів (стемінг). Це сприяє уніфікації слів з однаковим значенням та зменшенню розміру словника.

f) Видалення стоп-слів: Видалення загальних слів, які не несуть значущої інформації, таких як "я", "ти", "він", "та" тощо.

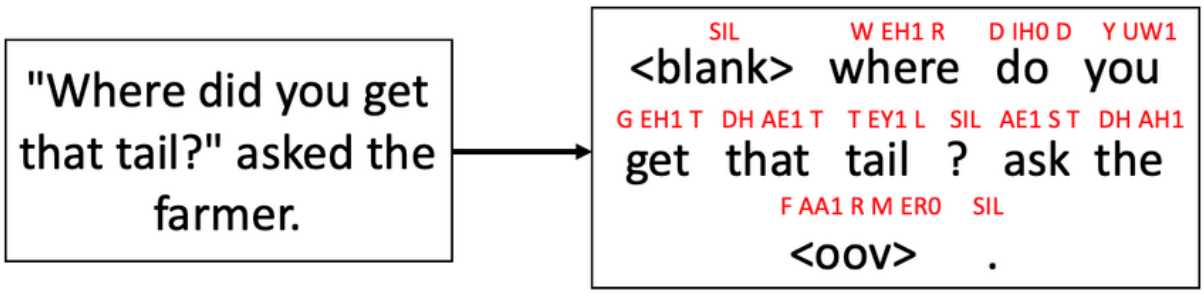

Рисунок 4 – Приклад Нормалізації тексту

Отже, простота імплементації простих методів потребує глибшої попередньої обробки даних. Але, навіть із попередньою обробкою, модель TextBlob показала точність на рівні 40-50% на тестовій вибірці даних, що не є задовільним показником, і оскільки модель за своєю простотою є обмеженою у масштабуванні, згодом при розробці було прийнято рішення відмовитись від використання даної моделі. Проте варто зазначити, що ці моделі не є безнадійними, та чудово підійдуть у більш простих задачах, де немає потреби у комплексному підході.

В ході вивчення моделей, я також проводив пошуки вже готових подібних проектів, та згодом знайшов статтю, в якій описано задачу, надзвичайно схожу до задачі у даній роботі.

Це проект, що має на меті класифікувати відгуки на Yelp за відповідними категоріями. Мотивацією цього проєкту стало спостереження, що хоча рейтинги Yelp дають загальне уявлення про досвід користувача, вони не надають контексту, який призвів до отримання тієї чи іншої оцінки. Наприклад, 4-зірковий відгук може не давати чіткого уявлення про те, чому користувач оцінив ресторан саме так. Тому команда вирішила класифікувати відгуки за категоріями, які можуть надати додатковий контекст. Категорії, які вони обрали на основі попереднього вивчення кількох сотень відгуків, такі: "Їжа", "Обслуговування", "Атмосфера", "Акції/знижки" та "Гідність" (яку можна узагальнити як співвідношення ціни та якості).

Класифікація відгуків за цими категоріями може допомогти іншим користувачам зрозуміти, чому ресторан отримав високу або низьку оцінку. Команда сформулювала це як навчальну задачу, а саме задачу класифікації з декількома етикетками, оскільки відгук може бути пов'язаний з декількома категоріями одночасно.

Команда використовувала набір даних Yelp, випущений для академічної задачі, який містить інформацію про 11 537 підприємств. Вони зосередилися лише на даних про ресторани, зменшивши кількість підприємств приблизно до 5 000. Вони відібрали всі відгуки про ці ресторани, які мали хоча б один корисний голос, і випадковим чином вибрали 10 000 з них. Вони розробили кодекс маркування для категорій шляхом початкового відкритого кодування випадкової вибірки з 400 відгуків. Після перевірки та уточнення кодексу на основі другої випадкової вибірки з 200 відгуків, вони розділили 10 000 відгуків на 5 кошиків і попросили аспірантів-дослідників анотувати кожен відгук у визначених категоріях. Результатом цього процесу стало 9019 оглядів, які вони розділили на 80% для навчання і 20% для тестування.

Для вилучення ознак вони використовували зіркові рейтинги і текстові ознаки, що складаються з уніграм, біграм і триграм. Вони створили бінарні ознаки для зіркових оцінок (1-2 зірки, 3 зірки і 4-5 зірок) і нормалізували текст відгуку для вилучення текстових ознак, зберігаючи стоп-слова через їхню важливість для розуміння настроїв користувачів. В результаті було виділено 375 одногрaммних ознак, 208 біграммних ознак і 120 триграммних ознак.

Що стосується класифікації, вони створили бінарний класифікатор для кожної категорії. Вони перетворили набір даних на 5 різних наборів

даних, кожен з яких містив інформацію лише про одну категорію. Отримавши новий відгук, бінарний класифікатор для кожної категорії прогнозує, чи належить відгук до тієї чи іншої категорії, а остаточний прогноз є об'єднанням усіх бінарних предикторів. Однак такий підхід ігнорує кореляції між категоріями. Щоб врахувати це, вони розглядали кожну окрему підмножин у категорій як одну категорію і вивчили багатокласовий класифікатор. Як результат, вони отримали модель, як надзвичайно точно передбачає категорії та оцінки цих категорій.

На жаль, у цій статті не зазначено нічого більше крім теорії, немає посилань чи згадок робочого репозиторію проєкту, немає посилань на датасет, що вони використовували, то ж можливість перевірити відсутня. Але це корисні знання, щодо інших підходів до такого роду задачі, в майбутньому, такий підхід можна буде розглянути наприклад, для написання Магістерської кваліфікаційної роботи.

2. РОЗРОБКА ТА ТРЕНУВАННЯ МОДЕЛІ BERT

2.1 Засоби Розроблення

В якості інструменту створення програмного засобу для виконання задачі даної роботи, було обрано PyCharm 2023.1 – інтегроване середовище розробки (IDE) мовою програмування Python.

2.2 Опис набору даних, що використовується для аналізу

У ході розробки проекту, виникла потреба в аналізі даних. Використовуючи сучасні технології штучного інтелекту, було обрано модель BERT для вирішення цього завдання. Цей метод вимагав детального налаштування моделі, що в свою чергу потребувало відповідного датасету для тренування.

Для ефективного аналізу, було вирішено, що найкращим датасетом для нашої моделі будуть відгуки користувачів, які вже були категоризовані та оцінені. Це дало можливість тренувати модель за конкретними принципами, що відображалися у вхідному наборі.

Однак, пришли до стадії, коли з'ясувалося, що знайти ідеальний датасет в Інтернеті, який б повністю відповідав би нашим потребам, було доволі складно. В результаті, виявилась потреба переглянути вимоги та спростити їх, обмежившись відгуками з "позитивними" та "негативними" загальними оцінками. Після додаткових пошуків, вдалий датасет був знайдений, що дозволило продовжити роботу.

Для реалізації цієї роботи було обрано датасет, що містить дві тисячі записів, у яких рівно поділені позитивні та негативні відгуки - по 1000

кожного типу. Це було зроблено з метою забезпечення балансу між позитивними та негативними емоціями в рамках даного датасету, що дозволяє забезпечити більш точну роботу моделі.

2.3 Підготовка даних для тренування моделі

Після збору необхідних записів, наступним етапом роботи стало виділення ключових категорій для аналізу відгуків. З метою оптимізації процесу було вирішено виділити наступні основні категорії: "Кухня", "Сервіс", "Атмосфера" та "Локація". Ці категорії були обрані, оскільки вони мають велике значення для споживачів і можуть бути важливими факторами, що впливають на загальний відгук.

Враховуючи, що існуючий датасет не містив явних позначок, що б дозволяли автоматично визначити, до якої категорії належить кожний відгук, було вирішено створити колекцію ключових слів для кожної з вибраних категорій (див. додаток 1). Ці ключові слова були обрані на основі їхнього зв'язку з певною категорією.

Після того, як були визначені категорії та ключові слова для них, кожна категорія отримувала емоційне забарвлення на основі загального емоційного тону відгуку. Так, наприклад, якщо відгук був позитивним, то всі категорії, що були ідентифіковані в цьому відгуку, отримували позитивне забарвлення.

Кінцевим результатом цього процесу стало створення фінального .csv файлу, що містить відформатований датасет з вказаними заголовками (headers):
"review_text","service_good","service_neutral","service_bad","kitchen_good","kitchen_neutral","kitchen_bad","ambience_good","ambience_neutral","ambience_bad","location_good","location_neutral","location_bad","overall_good","overall_neutral","overall_bad". Цей файл готовий до подальшого аналізу та обробки в рамках наступних етапів дослідницької роботи.

Програмний код для підготовки даних наведений у додатку Б №1

2.4 Опис процесу тренування моделі BERT

Як вже було зазначено раніше, модель BERT проходить два етапи налаштування або тренування. На моменті коли ми підключаємо бібліотеку до проєкту, модель вже навчена на величезних об’ємах даних, і тепер вже наша задача зробити тонке налаштування саме під наше завдання. Тренування моделі проходить у три етапи:

a) Підготовка до навчання
b) Процес навчання
c) Валідація

Тренування моделі(див. додаток Б п.2) відбувається у n-кількість так званих епох. Епоха — це один повний прохід через весь набір даних у процесі навчання моделі машинного навчання. Наприклад, візьмемо наш набір даних з 1000 прикладів, тоді одна епоха означає, що кожен з цих 1000 прикладів був використаний для оновлення ваг моделі.
Термін "епоха" важливий, тому що він визначає, скільки разів модель "бачила" кожен приклад у наборі даних. Якщо ви тренуєте модель

протягом 5 епох, це означає, що кожен приклад у вашому наборі даних використовувався 5 разів для оновлення ваг моделі.

Варто зазначити, що тренування моделі протягом більшої кількості епох не гарантує кращих результатів. У навчанні моделей машинного навчання важливо збалансувати кількість епох: занадто мало епох може призвести до недонавчання моделі, коли вона не зможе достатньо добре вивчити закономірності в даних, тоді як занадто багато епох може призвести до перенавчання, коли модель стає занадто специфічною для навчального набору даних і втрачає здатність узагальнювати на нових даних(див. Рисунок 5).

Вибір оптимальної кількості епох зазвичай вимагає експериментування та використання таких технік, як рання зупинка (early stopping), яка полягає в припиненні навчання, коли продуктивність на валідаційному наборі даних перестає покращуватися.

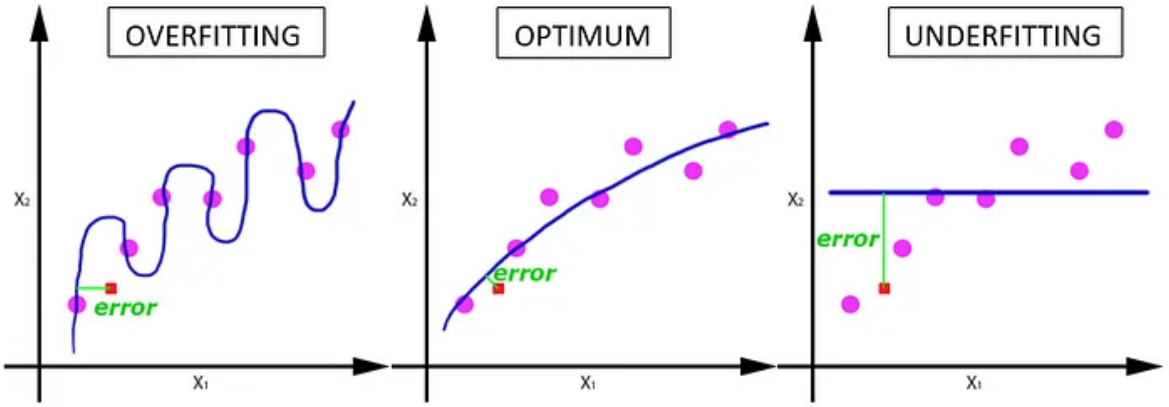

Рисунок 5 – Результати моделі від кількості епох

Отже, епохи це певного роду еволюція, де кожне покоління відрізняється від попереднього вагою, й модель росте(покращує точність) завдяки коригуванні циг ваг.

Ваги в машинному навчанні - це параметри, які модель використовує для визначення важливості конкретного вхідного значення або характеристики. Вони є ключовим компонентом багатьох моделей машинного навчання, включаючи нейронні мережі.

В контексті нейронних мереж, ваги використовуються для трансформації вхідних даних в вихідні значення. Це відбувається за допомогою векторних операцій, де вхідні дані (відомі як вектори вхідних даних) множаться на вектори ваг(див. Рисунок 6). Результатом цього процесу є вихідний вектор, який потім проходить через функцію активації, щоб отримати кінцевий вихід моделі.

$$Y = \sum (weight * input) + bias$$

Рисунок 6.

Ваги в моделі нейронної мережі зазвичай ініціалізуються випадковим чином і потім оптимізуються в процесі навчання.

На етапі підготовки до навчання, ініціалізується оптимізатор AdamW.
AdamW - це невелика модифікація алгоритму оптимізації Adam, яка включає ваговий регулятор, або вагову декомпозицію (Weight Decay).

Adam (Adaptive Moment Estimation) - це алгоритм оптимізації, що використовується в глибокому навчанні, зокрема, при тренуванні нейронних мереж. Він є покращенням інших алгоритмів оптимізації, таких як stochastic gradient descent (SGD) і RMSprop, і відомий своєю високою ефективністю і легкістю використання.

Основна ідея Adam полягає в адаптивному налаштуванні швидкості навчання для кожного параметра моделі на основі оцінок першого та

другого моментів градієнтів. Це означає, що Adam автоматично коригує свій крок навчання в залежності від того, як швидко чи повільно градієнти змінюються, що може прискорити процес навчання і поліпшити загальну точність моделі.

У традиційному Adam ваговий регулятор приміняється безпосередньо до градієнтів перед їх застосуванням, в той час як в AdamW він використовується безпосередньо до ваг, що змінюються під час оптимізації. Це робить регуляцію ваг більш явною та контрольованою, що може покращити результуючу модель. Пояснення роботи двух оптимізаторів продемонстровано на Рисунку 7.

Algorithm 2 Adam with L_2 regularization and Adam with weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{x}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{x}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{x}_{t-1}) + w\boldsymbol{x}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + w\boldsymbol{x}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{x}_t$

Рисунок 7 – Робота оптимізатора Adam та AdamW

Adam відслідковує середні значення (експоненційні ковзні) градієнта (називають першим моментом, відтепер позначатимемо як m) та квадрат градієнтів (називають сирий другий момент, відтепер позначатимемо як v).

На кожному часовому кроці обчислюється градієнт$g = \nabla f[x(t-1)]$, за чим вираховуються ковзні середні:

$$m(t) = \beta1 * m(t-1) + (1-\beta1) * g(t)$$

$$v(t) = \beta2 * v(t-1) + (1-\beta2) * g(t)^2$$

Параметри β1 (тобто 0.9) та β2 (тобто 0.999) контролюють, наскільки швидко середні значення зменшуються, тобто "наскільки далеко в минуле, береться середнє по градієнтам (в квадраті)". Інтерпретувати рівняння можна таким чином: "Нове середнє дорівнює 0.9 (або 0.999 для квадратів градієнтів) від старого середнього плюс 0.1 від поточного градієнта". З кожним часовим кроком старі градієнти множаться на 0.9 ще один раз, що означає, що вони все менше і менше вносять вклад в ковзне середнє.

Варто звернути увагу, що в рядках 9 і 10(див. Рисунок 7) середні значення масштабуються за допомогою (1-β^t), де t - це часовий крок. Щоб зрозуміти, чому це необхідно, розглянемо перший часовий крок, враховуючи, що m(0) та v(0) ініціалізуються як 0. Це означає, що середнє значення після першого часового кроку є $m(1) = 0.9 \cdot 0 + 0.1 \cdot g(1) = 0.1 \cdot g(1)$. Однак середнє значення після першого часового кроку має бути точно g(1), яке отримуємо, якщо поділити m(1) на $(1–0.9^1)=0.1$.

Ми встановлюємо η=1 для простоти (множник розкладу швидкості навчання) і об'єднуємо все в рядку 12(див. Рисунок 7):

При спуску "вниз по схилу" розмір кроку адаптується шляхом множення швидкості навчання α на m(t) і ділення на корінь з v(t) (наразі ігноруємо знак капелюха ^).

$$x(t) = x(t-1) — \alpha \cdot m(t) / [\, sqrt(\, v(t)\,) + \epsilon]$$

Пам'ятайтаємо, що дисперсія випадкової величини x визначається як $Var(x) = < x^2 > - < x >^2$, де $< >$ - очікуване значення. Експоненційне ковзне середнє квадрату градієнтів v(t) називається невицентрованою дисперсією, тому що ми не віднімали квадрат середнього значення градієнтів.

Дисперсія кількісно визначає, наскільки градієнти відрізняються від своїх середніх значень. Якщо градієнти залишаються приблизно постійними, оскільки ми "ходимо по луці", то дисперсія градієнтів приблизно 0, а невицентрована дисперсія v(t) приблизно дорівнює m(t)². Це означає, що $m(t) / sqrt(v(t))$ приблизно дорівнює 1, і крок "вниз по схилу" є порядку α.

Якщо ж градієнти швидко змінюються, sqrt(v(t)) значно більше, ніж m(t), тоді крок "вниз по схилу" значно менше, ніж α.

Підсумовуючи, це означає, що Adam може адаптувати розміри кроків для кожної окремої ваги, оцінюючи перший і другий моменти градієнтів. Коли градієнти не змінюються багато і "не потрібно бути обережними, ідучи вниз по схилу", розмір кроку є порядку α, якщо ж вони

змінюються, і "потрібно бути обережними, щоб не йти в неправильному напрямку" – розмір кроку набагато менший.

Ідея за L2-регуляризацією(див. Рисунок 7) або згасанням ваг полягає в тому, що мережі з меншими вагами (за умови, що всі інші речі рівні) спостерігаються як менш перенавчені та кращі в узагальненні.

Звичайно, великі ваги все ще можливі, але лише якщо вони значно зменшують втрати. Швидкість згасання ваги за крок w визначає відносну важливість мінімізації початкової функції втрат (проте більше важливо, якщо вибрано мале w) і знаходження малих ваг (проте більш важливо, якщо вибрано велике w). Якщо порівняти оновлення ваг, як пояснено раніше (нова вага дорівнює старій вазі мінус швидкість навчання помножена на градієнт)

$$x(t) = x(t-1) — \alpha \nabla f[x(t-1)]$$

з версією зі згасанням ваги

$$x(t) = (1-w)\, x(t-1) — \alpha \nabla f[x(t-1)]$$

то можна помітити додатковий член (1-w), який експоненціально зменшує ваги x і тим самим змушує мережу вчити менші ваги.

Часто, замість проведення згасання ваги, визначається регуляризована функція втрат (L2-регуляризація):

$$f\_reg[x(t-1)] = f[x(t-1)] + w'/2 \cdot x(t-1)^2$$

Якщо розрахувати градієнт цієї регуляризованої функції втрат:

$$\nabla f\_reg[x(t-1)] = \nabla f[x(t-1)] + w' \cdot x(t-1)$$

і онивити ваги:

$$x(t) = x(t-1) - \alpha \nabla f\_reg[x(t-1)]$$

$$x(t) = x(t-1) - \alpha \nabla f[x(t-1)] - \alpha \cdot w' \cdot x(t-1)$$

то побачимо, що це еквівалентно згасанню ваги, якщо позначити $w' = w/\alpha$.

Зазвичай бібліотеки глибокого навчання реалізують останню L2-регуляризацію.

Фіолетовий термін у рядку 6(див. Рисунок 7) показує L2-регуляризацію в Adam (не AdamW), як це зазвичай реалізовано в бібліотеках глибокого навчання. Термін регуляризації додається до функції вартості, яка потім виводиться для розрахунку градієнтів g. Однак, якщо в цей момент додати термін згасання ваги, то ковзні середні величини градієнта та його квадрату (m і v) відслідковують не лише градієнти функції втрат, але й термін регуляризації.

Якщо вставити рядки 6, 7 і 8 у рядок 12 (наразі ігноруємо знак ^, оскільки припускається, що t велике, і тому β^t=0), оновлення ваг виглядає наступним чином:

$$x_t \leftarrow x_{t-1} - \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)(\nabla f_t + \boxed{w x_{t-1}})}{\sqrt{v_t} + \epsilon}$$

AdamW також може мати кращу поведінку при зміні швидкості навчання, а також при роботі з великими батчами, які стають все більш популярними в сучасних архітектурах глибокого навчання.

Розглядаючи дані алгоритми оптимізації, ми зачепили ще одне важливе поняття – “Батч”.

"Батч" в контексті машинного та глибокого навчання відноситься до підмножини набору даних, що використовуються для навчання моделі в межах одного кроку оновлення. Кількість кроків на одну епоху залежить від розміру батчу, наприклад, якщо взяти 1000 прикладів у нашому наборі даних і використати розмір батчу 100, то потрібно 10 кроків, щоб пройти одну епоху. Батчі є важливою частиною процесу навчання, і вони мають вплив на ефективність та якість навчання моделі.

При навчанні моделі BERT, датасет звичайно великий і складається з великої кількості речень або фрагментів тексту. Оскільки весь набір даних зазвичай занадто великий, щоб обробити його за один раз (особливо в контексті обмеженої обчислювальної потужності), ми розділяємо його на менші підмножини, які називаються "батчами".

Батчі використовуються в процесі навчання наступним чином:

a) Модель передбачає вихід для всіх прикладів в батчі.

b) На основі передбаченого виходу та фактичного виходу розраховуються втрати.

c) Втрати використовуються для обчислення градієнтів, які потім використовуються для оновлення ваг моделі.

Оптимізатор AdamW ініціалізується з двома параметрами: швидкість навчання (lr) та eps:

a) Швидкість навчання (lr): Це гіперпараметр, що визначає, наскільки швидко модель навчається. Це впливає на те, наскільки сильно ваги моделі оновлюються на кожному кроці навчання. Значення 2e-5 є досить стандартним для BERT та інших трансформерних моделей. Якщо швидкість навчання занадто висока, модель може перескакувати оптимальне значення вагів і не збігатися до оптимального рішення. Якщо швидкість навчання занадто низька, модель може навчатися дуже повільно або застрягати в певній точці та не збігатися до оптимального рішення.

b) Epsilon (eps): Цей параметр використовується для підвищення стабільності обчислень. Він додається до деномінатора в рівнянні Adam для того, щоб уникнути ділення на нуль. Значення 1e-8 є типовим для багатьох застосувань і слугує для попередження числових проблем з обчисленням.

Важливо зазначити, що ці параметри можуть варіюватися в залежності від конкретної задачі та набору даних.

Разом з оптимізатором визначається загальна кількість кроків за формулою:

Загальна кількість кроків = Кількість батчів на одну епоху * Кількість епох

Після цього ініціалізується Планувальник (scheduler) – компонент, що керує швидкістю навчання в процесі оптимізації. У даній задачі планувальник починає з "періоду розігріву", під час якого швидкість навчання лінійно збільшується від 0 до початкової швидкості навчання. Після цього періоду, швидкість навчання лінійно зменшується до 0 протягом решти тренування.

Далі необхідно визначити, на якому процесорі буде відбуватись навчання: CPU чи GPU.

CPU – це загального призначення процесор, який ефективно виконує невеликі обчислювальні задачі, які мають різну структуру. CPU має менше ядер, але кожне ядро працює на вищій тактовій частоті, ніж ядра GPU.

GPU, з іншого боку, має більше ядер, які працюють на нижчій тактовій частоті. GPU був спеціально розроблений для виконання паралельних обчислень, що є особливо корисним для обчислювально вимогливих завдань, таких як обробка зображень та відео, і глибоке навчання.

Використання GPU для навчання моделей машинного навчання зазвичай дає значне прискорення порівняно з CPU, оскільки моделі машинного навчання зазвичай включають велику кількість паралельних обчислень (наприклад, матричні операції), які GPU може виконувати ефективніше.

Оскільки, на деяких машинах відсутні GPU, або ж ці GPU не підтримують
технології що використовуються у навчанні (Наприклад GPU від AMD не підтримують технологію CUDA і мають свій власний аналог – ROCm), то потрібно автоматично визначити на чому саме буде проходити навчання.

Після цього модель переноситься на відповідний пристрій і визначається функція витрат BCEWithLogitsLoss.
Функція втрат BCEWithLogitsLoss, надана PyTorch, об'єднує два компоненти: сигмоїдну активацію і бінарну кросс-ентропію втрати в одну одиничну функцію.

Вона використовується для бінарної класифікації і класифікації з багатьма мітками, коли класи не є взаємовиключними.

Втрата бінарної кросс-ентропії між цільовим та виведенням визначається наступним чином:

$$loss = -[y * log(\sigma(x)) - (1 - y) * log(1 - \sigma(x))]$$

де:
y — цільові значення, які мають бути 0 або 1,
x — прогнозовані значення (виведення моделі),
σ() — сигмоїдна функція.
Значення втрати, що вираховується функцією, використовується для оновлення ваг моделі в процесі зворотного розповсюдження помилки.

Після ініціалізації усіх потрібних функцій та визначення усіх необхідних параметрів, можна перейти до процесу навчання моделі. Процес навчання відбувається за наступними кроками:

a) Для кожної епохи модель переводиться в режим навчання.

b) Кожен пакет даних, що складається з вхідних ідентифікаторів, маски та міток, обробляється моделлю.

Маски - це вектори, що визначають, які елементи вхідних даних беруть участь в обчисленнях, маска вказує на те, які токени вважаються значущими для вивчення вхідного тексту.

Мітки - це "правильні" відповіді або цільові значення, які модель намагається передбачити під час навчання, вони використовуються для визначення втрати моделі.

c) Вивід моделі порівнюється з мітками, визначаючи втрати.

d) Втрати зворотно розповсюджуються через модель, оновлюються параметри моделі за допомогою оптимізатора, і планувальник оновлює швидкість навчання.

e) Після обробки всіх пакетів вираховується середній рівень втрат на епоху.

Наступним іде етап валідації моделі, він відбувається за такими кроками:

a) Після кожної епохи модель переводиться в режим валідації.

b) Кожен пакет валідаційного набору даних обробляється моделлю, при цьому втрати розраховуються для кожного пакета.

c) Середній рівень втрат валідації вираховується і виводиться.

Для демонстрації процесу навчання, продемонструємо вивід програми під час тренування моделі:

Epoch: 1 Training Loss: 0.5396037723934441 Validation Loss: 0.45765319040843416
Epoch: 2 Training Loss: 0.4200600427493714 Validation Loss: 0.3830463332789285
Epoch: 3 Training Loss: 0.3494508836353034 Validation Loss: 0.353766781943185
Epoch: 4 Training Loss: 0.315107147944601 Validation Loss: 0.3415019597326006

Легко бачити, що з кожною епохою, втрата на тренувальній та валідаційний вибірках поступово зменшується – це гарний признак того, що усе йде за планом та модель покращується з кожною епохою.

2.5 Оцінка результатів моделі

Результат програми - це відповідь у форматі json, у якому вказується:

a) Відгук який надійшов на аналіз.

b) Кількість очок та емоційна оцінка по кожній з категорій.

c) Кількість очок та емоційна оцінка загального настрою відгука.

Нижче наведено приклади деяких запитів.

Приклад №1 – повне співпадіння:


```
{
  "review": "A pleasant place. It's a pity that the terrace is not
covered, but everything else was perfect. Delicious pizza and excellent
service.",
  "kitchen": {
    "score": "0.3326112",
    "label": "good"
  },
  "service": {
    "score": "0.13249122",
    "label": "good"
  },
  "ambience": {
    "score": "0.053859822",
    "label": "neutral"
  },
  "location": {
    "score": "0.08384087",
    "label": "neutral"
  },
  "overall": {
    "score": "0.31393403",
    "label": "good"
  }
}
```

Цей приклад показав зразкову поведінку моделі, оскільки він точно відобразив ті категорії, та їх емоційне забарвлення, що присутні у відгуку. Прочитавши відгук, розуміємо, що клієнт звертає увагу на некрату терасу, але не надає цьому багато уваги, при цьому зазначаючи смачну піцу та прекрасне обслуговування. Модель дуже чітко відобразила нейтральне ставлення до атмосфери та до локації ресторану, при цьому зазначивши, що їжа та сервіс були оцінені добре. Загальна оцінка теж була виставлена вірно.

Приклад №2 – часткове співпадіння:


```
{
  "review": "Potato was to dry, I just cant eat it, why kitchen cant make it good, its just potato",
  "kitchen": {
    "score": "0.2293233",
    "label": "bad"
  },
  "service": {
    "score": "0.16830379",
    "label": "bad"
  },
  "ambience": {
    "score": "0.05052023",
    "label": "neutral"
  },
  "location": {
    "score": "0.09814048",
    "label": "neutral"
  },
  "overall": {
    "score": "0.3009841",
    "label": "bad"
```

```
  }
}
```


У цьому випадку, модель вірно визначила погану оцінку “Кухні”, а також нейтральне ставлення до атмосфери та локації, але при цьому помилково визначила сервіс поганим, при тому, що у відгуку сервіс не згадувався. При цьому - загальна оцінка виставлена вірно.

Приклад №3 – повне неспівпадіння:


```
{
  "review": "I like your design, and location is nice, a lot of parking
lots",
  "kitchen": {
    "score": "0.30991426",
    "label": "good"
  },
  "service": {
    "score": "0.13976756",
    "label": "good"
  },
  "ambience": {
    "score": "0.05419155",
    "label": "neutral"
  },
  "location": {
    "score": "0.09986834",
    "label": "neutral"
  },
  "overall": {
    "score": "0.326245",
    "label": "good"
  }
}
```

Незважаючи на те, що загальна оцінка виставлена вірно, усі інші категорії виставлені помилково. У відгуку були згадки про розташування та атмосферу, та не було згадок про їжу та сервіс, проте модель визначила усе навпаки.

Загальна точність моделі варіюється на рівні 60-70%, що хоч і далеко від ідеалу, але вже значно вище, аніж раніше згадана модель TextBlob.

ВИСНОВКИ

Підведення підсумків дослідження

Сучасний світ, з кожним днем генерує все більше інформації, тож виникає велика потреба в її обробці. Зі значним збільшенням об’єму інформації, людські ресурси втрачають здатність ефективно її опрацьовувати, це спонукає до пошуків та розробки методів та програм, щоб дозволити зменшити час обробки та збільшити її якість. Дослідження спрямоване на знаходження оптимального методу аналізу потокових даних. Для конкретизації задачі дане дослідження виділило серед потоку даних – ресторанні відгуки, одну з найпоширеніших галузей людства.

В ході дослідження було розглянуто ряд готових проектів, що пропонують рішення даної проблеми. Розглянуто різні моделі роботи з текстом, на кшталт VADER, TextBlob, BERT. Дана робота надає фундамент для розробки аналітичного проєкту, заснованого на штучному інтелекті та машинному навчанні на власних підготовлених даних.

Обговорення можливих шляхів покращення результатів

Перш за все, важливо зазначити, що результати даної роботи напряму залежать від якості та характеристик даних, на основі яких відбувається тонке налаштування моделі BERT. У цьому дослідженні виявлено декілька проблем, пов'язаних з підготовкою даних:

а) По-перше, виявлено, що тренувальний набір даних має відносно невеликий об'єм. Якісна модель, зазвичай, вимагає значного обсягу

даних, що налічується десятками або навіть сотнями тисяч. Відомо, що принцип "чим більше - тим краще" застосовується в цьому випадку.

b) По-друге, наявні дані були протегезовані прямолінійно. Наприклад, негативний відгук призводив до відзначення всіх категорій як негативних. Однак на великому обсязі даних такий підхід може призвести до формування неправильних маркерів для моделі, оскільки очевидно, що навіть у загально негативному відгуку можуть міститися позитивні відомості щодо окремих категорій. Для значного покращення результатів необхідно провести ретельну роботу, включаючи залучення додаткової людської робочої сили для обробки та тегування великого обсягу даних. Звичайно, це вимагатиме значних витрат часу та ресурсів, проте це дозволить моделі краще адаптуватися до конкретної задачі та суттєво поліпшити результати.

Незважаючи на вищевказані проблеми з даними, варто відмітити, що концепція представлення відгуків з розбиттям на 15 різних маркерів є дуже точним підходом до навчання моделі, що допомагає їй краще зрозуміти контекст кожного відгука тим самим збільшивши її точність.

Покращення даних є лише одним аспектом оптимізації моделі, існує також важливий аспект, пов'язаний з визначенням параметрів навчання моделі. Ці параметри включають розмір батчу, кількість епох, швидкість навчання та потужність обчислювальної машини, на якій виконується тренування. Кожен з цих параметрів має значний вплив на ефективність навчання та результати моделі.

Розмір батчу визначає кількість прикладів даних, які обробляються моделлю перед оновленням ваг моделі під час навчання. Великий розмір

батчу може призвести до більш стійкого оновлення ваг моделі, але вимагатиме більше обчислювальних ресурсів. З іншого боку, малий розмір батчу може призвести до більшої варіативності під час оновлення ваг моделі, але зменшить обчислювальну навантаженість. Вибір оптимального розміру батчу залежить від розміру даних, обчислювальних ресурсів та особливостей конкретної задачі.

Кількість епох визначає, скільки разів модель повторно пройде через усі дані навчання. Занадто низька кількість епох може призвести до недофітингу моделі, коли вона не встигає вивчити всі особливості даних. З іншого боку, занадто велика кількість епох може призвести до перенавчання моделі, коли вона "запам'ятовує" дані навчання і не здатна добре узагальнювати на нові дані. Оптимальна кількість епох також залежить від складності задачі та розміру даних.

Швидкість навчання визначає крок оновлення ваг моделі під час навчання. Велика швидкість навчання може призвести до швидкого збіжності моделі, але при надмірній величині ваги можуть рухатися "заразливо", пропускаючи оптимальні рішення. Занадто мала швидкість навчання може призвести до повільної збіжності або навіть застрягання в локальних мінімумах. Вибір оптимальної швидкості навчання вимагає експериментів та налаштування.

Нарешті, потужність обчислювальної машини, на якій виконується тренування, також має значний вплив на ефективність навчання. Більш потужна машина може обробляти більші обсяги даних та складніші моделі швидше. Оптимальний рівень потужності залежить від розміру даних, складності моделі та обчислювальних ресурсів, доступних для тренування.

Таким чином, визначення параметрів навчання моделі, включаючи розмір батчу, кількість епох, швидкість навчання та потужність машини, є надзвичайно важливим для досягнення оптимальних результатів навчання та ефективної роботи моделі.

Рекомендації щодо подальшого використання моделі BERT для аналізу відгуків

Для забезпечення якісного використання моделі BERT необхідно провести значний обсяг підготовки даних. Важливим етапом є імплементація додаткової моделі для поляризації слів, можливо, заснованої на моделі TextBlob. Цей підхід дозволить навчати модель на нормалізованих відгуках, попередньо оброблених за допомогою TextBlob. Ймовірно, такий підхід значно покращить точність моделі. Проте, варто врахувати, що швидкість обробки кожного відгуку буде значно знижуватись. Тому рекомендується використовувати потужну обчислювальну машину для обробки даних.

Для ефективного аналізу такого потоку даних необхідні серйозні обчислювальні ресурси. В таких випадках вже не обходяться простими ноутбуками. Для досягнення ефективності рекомендується розглянути можливість оренди потужностей, оскільки це зазвичай є вигіднішим з точки зору вартості та масштабованості.

Таким чином, для досягнення якісного використання моделі BERT важливо провести комплексну підготовку даних, включаючи імплементацію моделі для поляризації слів та використання потужних обчислювальних ресурсів для ефективної обробки даних.

При цьому, мушу зауважити, результати даної роботи є гарним початком для створення власного проєкту з аналітичної підтримки ресторанного бізнесу, де дана робота може виступати як незалежним аудитором, так і постачальником ліцензійного продукту, який буде ключовою ланкою між бізнесом та, наприклад, відділом маркетингу та іміджу.

Перспективи подальших досліджень

В подальшому дослідженні маються перспективи використання інших моделей, аналогічних BERT, а також комбінацій різних моделей і кардинально нових підходів, розроблених самостійно. Аналіз тексту є надзвичайно творчим та дослідницьким простором, оскільки текст є основним засобом комунікації між людьми від давніх часів. Нині, завдяки потужностям сучасних обчислювальних систем, практично кожен має можливість проводити власні дослідження, експерименти з аналізу тексту, виявлення закономірностей, виявлення аномалій, а також побудову власних аналітичних проектів, текстових нейронних моделей та іншого.

Це відкриває нові горизонти для розвитку та вдосконалення методів аналізу тексту. Наприклад, можна використовувати альтернативні моделі, що базуються на глибокому навчанні та природній мові, такі як GPT, XLNet, або RoBERTa, для досягнення кращих результатів в аналізі тексту. Крім того, комбінація різних моделей може привести до синергічного ефекту та покращення результатів.

Додатково, власноруч розроблені підходи можуть принести інновації у галузі аналізу тексту. Розробка нових алгоритмів, використання нестандартних методів аналізу, а також пошук нових підходів до

розуміння та обробки тексту відкривають широкі можливості для подальших досліджень.

В цілому, розвиток інструментів та методів аналізу тексту відкриває дорогу до нових можливостей дослідження та розуміння мови. Завдяки доступності потужних обчислювальних ресурсів і широкому спектру інструментів, стає можливим втілення власних ідей, розробка нових аналітичних проектів та внесення власного вагомого внеску у галузь аналізу тексту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. (2017). Attention Is All You Need. Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS 2017).

[2] Rani Horev. (2018). BERT Explained: State of the art language model for NLP.

[3] Diego Lopez Yse. (2021). Text Normalization for Natural Language Processing (NLP)

[4] RABI KUMAR SINGH. (2018). Restaurant dataset

[5] SAGAR SHARMA. (2017). Epoch vs Batch Size vs Iterations

[6] Paperspace . (2022). Weights and Biases

[7] Fabio M. Graetz . (2018). Why AdamW matters

[8] OpenAI . (2023). About OpenAI

[9] Aryan Bajaj . (2021). Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

ДОДАТОК

Додаток А

Додаток 1. Таблиця 1 – ключові слова до категорій ресторанних відгуків.

Kitchen	Service	Ambience	Location
tasty	wait	atmosphere	location
delicious	service	ambience	place
fresh	time	mood	spot
good	fast	vibe	area
great	slow	feeling	neighborhood
nice	quick	setting	district
yummy	long	decor	region
flavorful	short	furniture	local
spicy	minute	light	near
savory	hour	music	far
terrible	friendly	sound	close
sweet	rude	noise	distance
sour	polite	quiet	drive
bitter	courteous	loud	parking
umami	smile	crowd	lot
hot	frown	busy	street
cold	greet	empty	road
warm	serve	full	highway
cool	server	clean	route
food	waiter	dirty	direction
meal	waitress	neat	map
dish	staff	tidy	gps
cuisine	employee	messy	landmark

Таблиця 1 – продовження.

Kitchen	Service	Ambience	Location
menu	crew	smell	sign
recipe	team	scent	view

ingredient	help	aroma	scene
seasoning	assist	fragrance	scenery
sauce	customer	odor	sight
drink	client	stink	landscape
beverage	patron	air	building
appetizer	tip	conditioning	architecture
dessert	bill	temperature	design
snack	check	ventilation	style
breakfast	cash	comfort	structure
lunch	credit	space	
dinner	pay		
supper	charge		
pizza	price		
burger			
salad			
sushi			
steak			
fries			
chicken			
fish			
vegan			
caesar			
vegetarian			
gluten-free			
dairy-free			
spicy			

Таблиця 1 – продовження.

Kitchen	**Service**	**Ambience**	**Location**
sour			
bitter			

salty			
sweet			

Додаток Б

Даний проект і його опис доступні у вільному доступі для перегляду на гіт-репозиторії за наведеним посиланням.

https://gitlab.com/dima_tar/bert-restaurant-review-analysis

Додаток 1 – програмний код підготовки вхідних даних

```
import os
from loguru import logger
from typing import Optional

current_dir = os.path.dirname(os.path.realpath(__file__))

positive_folder = os.path.join(current_dir, 'data',
'restaurant_reviews_pos_and_neg', 'positive')
negative_folder = os.path.join(current_dir, 'data',
'restaurant_reviews_pos_and_neg', 'negative')

category_words = {
   'kitchen': ['tasty', 'delicious', 'fresh', 'good', 'great', 'nice',
'yummy', 'flavorful', 'spicy',
               'savory', 'terrible', 'sweet', 'sour', 'bitter', 'umami',
'hot', 'cold', 'warm', 'cool', 'food',
               'meal', 'dish', 'cuisine', 'menu', 'recipe', 'ingredient',
'seasoning', 'sauce', 'drink',
               'beverage', 'appetizer', 'dessert', 'snack', 'breakfast',
'lunch', 'dinner', 'supper',
               'pizza', 'burger', 'salad', 'sushi', 'steak', 'fries',
'chicken', 'fish', 'vegan', 'caesar',
               'vegetarian', 'gluten-free', 'dairy-free', 'spicy', 'sweet',
'sour', 'bitter', 'salty'],
   'service': ['wait', 'service', 'time', 'fast', 'slow', 'quick', 'long',
'short', 'minute', 'hour',
               'friendly', 'rude', 'polite', 'courteous', 'smile', 'frown',
```

```
'greet', 'serve', 'server',
              'waiter', 'waitress', 'staff', 'employee', 'crew', 'team',
'help', 'assist', 'customer',
              'client', 'patron', 'tip', 'bill', 'check', 'cash', 'credit',
'pay', 'charge', 'price'],
   'ambience': ['atmosphere', 'ambience', 'mood', 'vibe', 'feeling',
'setting', 'decor', 'furniture',
               'light', 'music', 'sound', 'noise', 'quiet', 'loud', 'crowd',
'busy', 'empty', 'full',
               'clean', 'dirty', 'neat', 'tidy', 'messy', 'smell', 'scent',
'aroma', 'fragrance',
               'odor', 'stink', 'air', 'conditioning', 'temperature',
'ventilation', 'comfort', 'space'],
   'location': ['location', 'place', 'spot', 'area', 'neighborhood',
'district', 'region', 'local',
               'near', 'far', 'close', 'distance', 'drive', 'parking', 'lot',
'street', 'road', 'highway',
               'route', 'direction', 'map', 'gps', 'landmark', 'sign',
'view', 'scene', 'scenery',
               'sight', 'landscape', 'building', 'architecture', 'design',
'style', 'structure']
}


def make_dataset(tag: str) -> Optional[None]:
   if tag == 'positive':
       logger.info('Making positive dataset')
       input_folder = positive_folder
       output_folder = os.path.join(current_dir, 'dataset',
'positive_with_categories')
       overall = 1
   elif tag == 'negative':
       logger.info('Making negative dataset')
       input_folder = negative_folder
       output_folder = os.path.join(current_dir, 'dataset',
'negative_with_categories')
       overall = 1
   else:
       logger.error('Invalid tag')
```

```
        return None

    for file in os.listdir(input_folder):

        is_kitchen = False
        is_service = False
        is_ambience = False
        is_location = False

        with open(os.path.join(input_folder, file), 'r') as f:
            review = f.readline().lower()
            review_1 = review[:len(review) // 2]
            review_2 = review[len(review) // 2:]
            if review_1 == review_2:
                review = review_1
            for category in category_words:
                for word in category_words[category]:
                    if word in review:
                        if category == 'kitchen':
                            is_kitchen = True
                        elif category == 'service':
                            is_service = True
                        elif category == 'ambience':
                            is_ambience = True
                        elif category == 'location':
                            is_location = True
                        break

            if tag == 'positive':
                res = f'"{review}",{1 if is_service else 0},0,0,{1 if
is_kitchen else 0},' \
                      f'0,0,{1 if is_ambience else 0},0,0,' \
                      f'{1 if is_location else 0},0,0,1,0,0'

            elif tag == 'negative':
                res = f'"{review}",0,0,{1 if is_service else 0},0,0,{1 if
is_kitchen else 0},' \
                      f'0,0,{1 if is_ambience else 0},0,0,' \
                      f'{1 if is_location else 0},0,0,1'
```

```
            with open(os.path.join(output_folder, file), 'w') as write_file:
                write_file.write(res + '\n')


def main():
    logger.info("Start...")
    make_dataset('positive')
    make_dataset('negative')
    logger.info("Separated datasets done")
    logger.info("Making final dataset")

    with open(os.path.join(current_dir, 'dataset', 'dataset.csv'), 'w') as
dataset:

        dataset.write(

'"review_text","service_good","service_neutral","service_bad","kitchen_good","
kitchen_neutral",'

'"kitchen_bad","ambience_good","ambience_neutral","ambience_bad","location_goo
d","location_neutral",'
            '"location_bad","overall_good","overall_neutral","overall_bad"\n')

        for file in os.listdir(os.path.join(current_dir, 'dataset',
'positive_with_categories')):
            with open(os.path.join(current_dir, 'dataset',
'positive_with_categories', file), 'r') as f:
                dataset.write(f.readline())
        for file in os.listdir(os.path.join(current_dir, 'dataset',
'negative_with_categories')):
            with open(os.path.join(current_dir, 'dataset',
'negative_with_categories', file), 'r') as f:
                dataset.write(f.readline())

    logger.info("Final dataset done")


if __name__ == '__main__':
```

```
    main()
```

Додаток 2 – Програмний код для тренування моделі BERT

```
from transformers import get_linear_schedule_with_warmup
from torch.nn import BCEWithLogitsLoss
from torch.optim import AdamW
from loguru import logger
import torch


def train_model(model, train_dataloader, validation_dataloader, epochs):
    logger.info('Preparing for training...')
    optimizer = AdamW(model.parameters(),
                      lr=2e-5,
                      eps=1e-8)

    total_steps = len(train_dataloader) * epochs
    scheduler = get_linear_schedule_with_warmup(optimizer,
                                                num_warmup_steps=0,
                                                num_training_steps=total_steps)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    logger.info(f'Using device {torch.cuda.get_device_name(0)}')
    model = model.to(device)
    loss_func = BCEWithLogitsLoss()

    logger.info("Training preparing complete. Starting training...")
    for epoch_i in range(0, epochs):
        total_train_loss = 0

        model.train()

        for step, batch in enumerate(train_dataloader):
            b_input_ids = batch[0].to(device)
            b_input_mask = batch[1].to(device)
```

```
            b_labels = batch[2].to(device)

            model.zero_grad()

            outputs = model(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)

            loss = loss_func(outputs.logits, b_labels.type_as(outputs.logits))
            total_train_loss += loss.item()
            loss.backward()

            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

            optimizer.step()
            scheduler.step()

        avg_train_loss = total_train_loss / len(train_dataloader)

        model.eval()

        total_eval_accuracy = 0
        total_eval_loss = 0
        nb_eval_steps = 0

        for batch in validation_dataloader:
            b_input_ids = batch[0].to(device)
            b_input_mask = batch[1].to(device)
            b_labels = batch[2].to(device)

            with torch.no_grad():
                outputs = model(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)

            loss = loss_func(outputs.logits, b_labels.type_as(outputs.logits))
            total_eval_loss += loss.item()

        avg_val_loss = total_eval_loss / len(validation_dataloader)

        logger.info(f"Epoch: {epoch_i + 1} Training Loss: {avg_train_loss}
```

```
Validation Loss: {avg_val_loss}")

    logger.info("Training complete!")


    return model
```