НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису» «До захисту допущено»

УДК _____ В.о. завідувача кафедри

_____ М.В.Грайворонський

“___”_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 113 Прикладна математика

на тему: Класифікація контенту Twitter-стрічки на основі алгоритмів машинного навчання

Виконав (-ла): студент (-ка) 2 курсу, групи ФІ-81мп
(шифр групи)

Топчій Данило Анатолійович
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доц. Родіонов А. М.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітньо-професійна програма) – 113 Прикладна математика («Математичні методи моделювання, розпізнавання образів та безпеки даних»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«____»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Топчію Данилу Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації: Класифікація контенту Twitter-стрічки на основі алгоритмів машинного навчання

науковий керівник дисертації к.т.н., доц. Родіонов Андрій Миколайович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15» листопада 2019 р. № 3927-с

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження

4. Вихідні дані

5. Перелік завдань, які потрібно розробити

6. Орієнтовний перелік ілюстративного матеріалу

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації[1*]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

[1*] Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Обсяг роботи 71 сторінка, 28 ілюстрацій, 1 таблиця, 3 додатки, 16 джерел літератури.

Об’єктом дослідження є обробка природних мов з використанням методів машинного навчання.

Предметом дослідження є класифікація тексту повідомлень користувачів соціальної мережі Твітер за допомогою машинного навчання.

Обсяг інформації що з’являється в мережі інтернет росте з кожним днем. Останнім часом більша частина контенту в мережі складає контент, згенерований користувачами в соціальних мережах. Однією із важливих функцій соціальних мереж, є класифікація контенту користувачів і рекомендаційна стрічка на основі вподобань користувачів, яка в тій чи іншій мірі реалізована в кожній з популярних сьогодні мереж.

Класифікацією текстової інформації займається такий напрям інформатики, як обробка природної мови. Людська мова транслюється в векторні представлення і за допомогою математичної лінгвістики, машинного навчання та штучного інтелекту природний текст стає зрозумілим для машин.

Результатом роботи є розроблена розподілена система з мікросервісною архітектурою, яка в режимі реального часу може отримувати повідомлення з Твітеру за вказаними фільтрами та класифікувати їх вміст за різними показниками.

Обробка природної мови, Твітер, машинне навчання, класифікатор, розподілені системи.

ABSTRACT

Volume of work 71 pages, 28 illustrations, 1 table, 3 supplements, 16 sources of literature.

The object of the study is natural language processing using machine learning methods.

The subject of the study is classification and processing users posts in Twitter using natural language processing and machine learning.

The amount of information that appears on the Internet is growing every day. Most of the content in the network is user generated content in social networks. One of the important features of social networks is classification of content, filtering, and personalized recommendation feed based on users preferences, which is implemented in every popular network.

Classification of textual information deals with such area of computer science as natural language processing. Human language is translated into vector representations, and using mathematical linguistics, machine learning and artificial intelligence, natural text is understood by machines.

Result of the work is a distributed system with a microservice architecture, which in real time can receive messages from Twitter by the specified filters and classify their contents according to different indicators.

Natural language processing, Twitter, machine learning, classifier, distributed systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP - natural language processing (обробка природних мов)

ML - machine learning (машинне навчання)

AI - artificial intelligence (штучний інтелект)

RNN - recurrent neural network (рекурентні нейронні мережі)

CNN - convolutional neural network (згорткові нейронні мережі)

DL - deep learning (глибоке навчання)

НМ - нейронні мережі

Мб - мегабайт

CBOW - неперервна мультимножина зі словами

FNN - нейронна мережа прямого поширення

ANN - Artificial neural networks

LM - Language models (мовні моделі)

BPTT - back-propagation through time

ВСТУП

Технології змінюють спілкування сучасного суспільства. Міжнародна корпорація даних (IDC) підраховує, що до 2025 року пов'язана людина буде взаємодіяти з технологіями, керованими даними, в середньому кожні 18 секунд.

Природна мова, яка є мовою, яка зазвичай використовується для розмови та письма, визначає популярний спосіб спілкування людей із машинами та людьми. Тексти зберігають сказане і вже давно є важливим форматом збереження людських знань. На ній базуються не лише книги чи газети, тексти - це загальний інформаційний ресурс у всесвітній мережі Інтернет. Особливо в соціальних мережах користувачі генерують величезну кількість текстових даних щосекунди. У 2018 році щомісяця у Twitter публікується близько 500 000 коротких текстових повідомлень, що дорівнює близько 6000 повідомлень в секунду, а Reddit отримує приблизно 2000 нових коментарів кожні 60 секунд. Велика кількість даних зберігається в такому неструктурованому форматі, який легко зрозуміти людям, але дуже важко для машин.

Останнім часом був досягнутий великий прогрес у впровадженні технологій обробки природних мов, таких як вилучення інформації, семантичний або граматичний аналіз, ці рішення часто базуються на різних моделях машинного навчання.

З метою зробити людські мови зрозумілими для машин, дослідження в галузі обробки природних мов проводилися з кінця 40-х років. На початку НЛП машинний переклад був у центрі уваги дослідження. Тим часом проводиться все більше досліджень у сферах NLP та машинного навчання, спричинених бумом глибокого навчання (DL), який зумовлений поліпшенням роботи комп’ютерних ресурсів, збільшенням доступності даних та прогресом у дослідження.

Майже кожна взаємодія між людиною та машиною має щось спільне із сферою НЛП. Через велику кількість інформації, що міститься в текстах, і частого контакту з ними, їх автоматичний аналіз має вирішальне значення для подолання перевантаження інформацією. Деякі програми, які покладаються на використання НЛП, такі як пошук інформації, узагальнення тексту та категоризація, агрегують інформаційний вміст, щоб зробити його більш доступним для людей та машин. Наприклад, NLP використовується для аналізу публікацій новин та соціальних медіа

Важливим підходом в розробці та впровадженні NLP рішень є розробка програмного забезпечення, яке дозволить безперервно, надійно та в режимі реального часу виконувати обробку великих об’ємів даних та не заважати роботі основної системи. Одним з підходів є розробка розподілених систем для обчислення з мікросервісною архітектурою, які можуть працювати самостійно та забезпечувати систему необхідним функціоналом. Цей архітектурний стиль, який структурує додаток як сукупність служб, які є високорентабельними і перевіреними, нещільно з'єднаними, незалежно розгортаними, організованими навколо можливостей бізнесу. Архітектура мікросервісів дозволяє швидко, часто та надійно доставляти великі, складні програми. Це також дозволяє організації розвивати свій набір технологій.

Метою даної роботи є розробка розподіленої обчислюваної системи для обробки, аналізу та класифікації повідомлень з мережі Twitter в режими реального часу з використанням моделей машинного навчання та нейронних мереж.

Завданнями дослідження:

- проаналізувати існуючі задачі та методи NLP та штучного інтелекту, які використовуються для аналізу та класифікації природних мов

- розробити алгоритми для класифікації повідомлень за різними класами
- побудувати архітектуру системи для отримання данних з Твітеру, яка буде класифікувати ці дані за визначеними класами, фільтрувати неякісний контент

Об’єктом дослідження є штучний інтелект та розподілені системи для обробки та класифікації природних мов.

Предметом дослідження є класифікація контенту згенерованого користувачами Твітера за допомогою методів обробки природних мов та машинного навчання.

Методи досліджень. В основі досліджень лежить мікросервісна архітектура розподіленої системи яка використовує методи машинного навчання та штучного інтелекту для класифікації та обробки контенту користувачів.

Наукова новизна полягає в розробці розподіленої системи з мікросервісної архітектури яка використовує поетапну обробку тексту з використанням різних моделей машинного навчання та нейронних мереж і може обробляти великі об’єми даних в режимі реального часу.

Практичне значення одержаних результатів. Результати дослідження дозволяють впроваджувати розроблену програму в різні існуючі системи та робити аналіз і класифікацію даних не втручаючись в процес роботи основної систему.

1 ЗАДАЧІ ТА МЕТОДИ ОБРОБКИ ПРИРОДНОЇ МОВИ

1.1 Постановка задачі обробки природних мов

Обробка природної мови (NLP) пов'язана з іншими областями, такими як штучний інтелект, машинне навчання. У цьому розділі розглядаються різні методи і описується як методи штучного інтелекту, обробки природної мови, машинного навчання і глибокого навчання об'єднані в одному контексті. Штучний інтелект - це дуже широкий термін і спосіб описати системи, які здатні «мислити». AI складається з чотирьох основних частин: машинного навчання, міркування, планування і NLP. Міркування дозволяє машині висувати гіпотези на основі даних, в той час як планування дає системам можливість діяти автономно при інтерпретації даних.

NLP займається «використанням людських мов комп'ютером». У нього є багато різних додатків, які відносяться до неструктурованої природної мови людини. Наприклад, його областями застосування є машинний переклад, розпізнавання мови, діалогові системи, розпізнавання іменованих об'єктів, пошук інформації і класифікація тексту. Таким чином, область NLP охоплює всі взаємодії між комп'ютером і людиною з використанням письмової або усної природної мови. Область досліджень і застосування пов'язані з маніпулюванням і розумінням природних мов. Обробка людської мови заснована на розумінні передбачуваного значення повідомлення, що складно навіть для людей, наприклад, коли використовується іронія. Всі компоненти природної мови, такі як фонетика, фонологія, морфологія, синтаксис, семантика і прагматика, повинні бути прийняті до уваги, щоб отримати повне розуміння повідомлення. Фонетика - це акустичні властивості звуку, виробленого людським голосом. Він досліджує, як звуки сконструйовані фізично,

наприклад, мовою або губами. Звук конкретного людського мови вивчається фонологією.

Наприклад, англійська мова має 45 помітних звуків, які називаються фонемами. Фонетика і фонологія є особливо важливими аспектами в розпізнаванні мови при перетворенні звуків в реальні слова, які можуть бути оброблені комп'ютером. Морфологія стосується значення і архітектури слів. Стемінг і лематизації засновані на цьому компоненті шляхом перетворення слів, таких як «going», до слова «go». Порядок слів і побудова граматичних правильних пропозицій досліджується синтаксисом. На відміну від цього, семантика досліджує значення пропозицій, які побудовані з використанням синтаксису і морфологічних форм слова. Щоб отримати передбачуваний загальний зміст повідомлення, прагматика використовує контекст ситуації. Наприклад, припустимо, хтось запитує: «Не могли б ви передати сіль?». З огляду на контекст ситуації, питання - це фактично прохання передати сіль, а не питання, чи може хтось це зробити. Тому комп'ютер повинен враховувати всі частини природної мови, щоб використовувати його.

Одне відоме визначення машинного навчання засноване на ідеї досвіду і ілюструє навчальну частину в ML: "Кажуть, що комп'ютерна програма вчиться на досвіді E щодо деякого класу задач T і показника ефективності P, якщо його ефективність в задачах в T, виміряна P , поліпшується з досвідом E.

Наступний приклад повинен допомогти пояснити це визначення. Припустимо, у нас є система прогнозування дощу, яка передбачає, чи буде дощ сьогодні чи ні (T). Система є бінарним класифікатором, і її продуктивність буде вимірюватися за її точності (P). Він вчиться на часі і історичних даних про погоду (E), щоб передбачити правильний результат. Крім того, ML є одним з центральних предметів в штучному інтелекті і ділиться на чотири підкатегорії: контрольоване навчання, неконтрольоване навчання, посилене навчання і глибоке навчання. Контрольоване та неконтрольоване навчання - два основних

типи проблем інтелектуального аналізу даних. Різниця між цими завданнями полягає в структурі систему адаптації. У контрольованому наборі даних цільова змінна відома і може використовуватися для навчання моделі. Неконтрольовані проблеми не мають відомого результату, і рішення часто ґрунтується на схожості примірників, моделях і групах. Посилене навчання схоже на контрольоване навчання, але не враховує фіксований набір даних. Він часто використовується в іграх або в автомобілях з автоматичним управлінням. Алгоритми навчання з підкріпленням використовують метод проб і помилок, щоб витягти уроки з заданої мети. Вони взаємодіють з навколишнім середовищем і використовують зворотний зв'язок для поліпшення свого досвіду.

На відміну від згаданих вище методів, глибоке навчання - це метод їх вирішення, що дозволяє комп'ютеру виводити складні шаблони з простіших. Глибокі нейронні мережі є основними частинами DL. Відповідно до наведеного вище визначення, важко сказати, чи використовується DL чи ні, тому що складним шаблоном також може бути нейронна мережа з одним прихованим шаром і безліччю нейронів. Мережа не повинна бути глибокою у всіх вимірах. Терміни AI, NLP, ML і DL не повинні розумітися тільки самі по собі. Межі між темами не є строгими. Зокрема, ML широко використовується в NLP для вирішення різних типів завдань.

1.2 Вибір функцій і попередня обробка

Текст - це просто послідовність слів або, точніше, послідовність символів. Але коли ми зазвичай маємо справу з мовним моделюванням або обробкою природної мови, ми більше дбаємо про слова в цілому, а не просто турбуємося про глибину текстових даних на рівні символів. Одна з причин

цього полягає в тому, що в мовних моделях окремі символи не мають великого «контексту». Такі символи, як «к», «н», «и», «г», «а», не містять окремого контексту, але при перестановці в формі слова вони можуть генерувати слово “книга”.

Вибір функцій і попередня обробка є важливими завданнями в AI і в основному являють етап підготовки даних в CRISP-DM. Особливо в NLP, це завдання має великий вплив на успіх аналізу тексту. В основному це викликано неструктурованою і довільною природою текстових даних. Крім того, машинам потрібна структура і числові дані. Існує кілька підходів для цього завдання перетворення, наприклад, вкладення слів або модель векторного простору.

Векторизація - це метод перетворення слів в довгі масиви чисел, в яких може міститися якась складна структура, зрозуміла тільки комп'ютеру, що використовує будь-який спосіб машинного навчання або алгоритм інтелектуального аналізу даних. Але навіть до цього нам потрібно виконати послідовність операцій над текстом, щоб наш текст можна було «очистити». Процес «очищення» даних може варіюватися в залежності від джерела даних. Нижче перераховані основні етапи очищення текстових даних з поясненнями.

1.3 Токенізація

Для обробки письмової природної мови необхідно розбити текст на більш дрібні одиниці, які називаються токенами. Комп'ютери повинні розрізняти окремі об'єкти тексту, і для їх створення використовується токенізація. Зазвичай маркери являють собою прості слова, які є найменшими незалежними одиницями природної мови. Крім того, токени можуть складатися з ідіом або дефісів, наприклад, «створених користувачем». Токенізація розбиває тексти, що знаходяться в обробці, на короткі текстові об'єкти і є найпершою задачею в

будь-якому циклі попередньої обробки тексту. Крім розбиття маленьких блоків, цілі речення також можуть бути результатом токенізації. Простий токенізатор слів може бути реалізований на багатьох мовах. Цей простий базовий підхід має пару недоліків через відсутність ідентифікуючих слів, які семантично пов'язані одне з одним. Однак простий токенізатор ділить фразу “the best fox is running” на наступні маркери (рис. 1.1):

Рисунок 1.1 – Результат роботи простого токенізатора

Використовуючи маркери, можна створити так звані n-грами, які вказують набір токенів з довжиною n. «Грама» - це грецьке слово, що позначає букву або знак. Коли мова йде про набір з n букв в словах, мова йде про символи n грамів.

1.4 Видалення стоп-слів

Дуже важливим підходом до скорочення величезного необробленого простору введення в NLP є видалення стоп-слів. У більшості мов є специфічні слова, які з'являються частіше за інші або не містять багато інформації про зміст тексту, наприклад, допоміжні дієслова або артикули. У зв'язку з цим часто має сенс виключати ці так звані стоп-слова в подальшому аналізі. В англійській мові такі слова можуть бути «the», «a» або «an». Виняток можна зробити, звіряючи слова зі стандартним списком стоп-слів. Ці списки доступні в літературі і часто реалізуються в різних програмних пакетах. У нашому прикладі «the» і «is» виключені. Видалення стоп-слів слід використовувати з

обережністю, особливо в аналізі настроїв, який намагається передбачити позитивний або негативний намір тексту. Видалення виключатиме слова, які можуть змінити ціле твердження, наприклад, «not» чи «none», як зображено на рисунку 1.2.

Рисунок 1.2 – Результат видалення стоп-слів з токенів

1.5 Стемінг

Крім виключення стоп-слів, стемінг є корисним методом для зіставлення слів з основами їх слів і подальшого зменшення вхідного вимірювання. Це допомагає витягти реальний зміст тексту і робить неструктуровані дані більш доступними для комп'ютера. Перший алгоритм визначення основ, заснований на видаленні найдовших суксов і правописі, був розроблений в 1968 році. До теперішнього моменту алгоритм визначення походження портеру являє собою сучасний підхід і видаляє всі суфікси зі слів для збереження основи слова. Хоча цей метод добре працює на англійській мові, у німецької мови є деякі недоліки, пов'язані з тим, що німецькі слова зазвичай не створюються шляхом додавання суфіксів. Проте, існує німецький еквівалент, заснований на ідеї Портера і мовою обробки рядків Snowball. За допомогою англійської Porter Stemmer слова “best”, “fox” і “running” присвоюються таким словами (рис. 1.3):

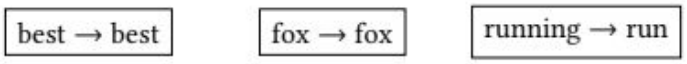

Рисунок 1.3 – Приклад стеммінгу

1.6 Лематизація

Лематизація - це процес зіставлення кожного слова в тексті до їх базової форми. Дієслова перетворюються в свою початкову форму, іменник відновлюється в однині, а прислівники або прикметники передбачають їх позитивний формат. Цей метод заснований на морфологічному аналізі і часто використовує словник, наприклад WordNet, де можна знайти лемму для кожного зміненого слова. Цей етап попередньої обробки скорочує векторний простір шляхом зіставлення різних форм слів з їхнім спільним представленням. Оскільки лематизація підтримується словниковими значенням, вона може зіставити «best» з його леммой «good» (рис. 1.4):

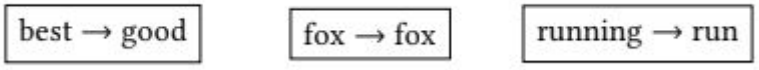

Рисунок 1.4 – Приклад роботи лематизації

1.7 Представлення тексту в векторній формі

Вкладення слів (word embeddings) - це загальний підхід до відображення багатовимірних векторів слів, таких як закодовані унітарними векторами, в низьковимірних представленні. Основна ідея була вперше згадана в 2001 році і

була мотивована подолати проблеми розмірності. Векторні представлення слів обчислюються для набору текстів зі словником розміру N. Для ілюстрації припустимо, що ми хочемо створити вкладення для одного документа: «сьогодні на вулиці гарна погода». Речення складається з п'яти окремих слів, що призводить до N = 5. Слово w_i в цьому словнику з i = 1 , … , N представлене вектором з визначеним виміром. Якщо використовується розмір вкладення (ED), рівний 4, тоді зберігає вкладення слів для всіх N = 5 слів в документі. E - матриця вкладень N × ED, яка нагадує словник. Тут кожне слово в словарі репрезентується одним рядком. Крім того, матриця E - це бажаний вихідний результатом кожного завдання побудування вкладень слів.

Embedding слова:

$$v3 = (0, 0, 1, 0, 0) \times E = (0.21, 0.18, 0.67, 0.89) \quad (1.1)$$

третє слово, яке позначено як w_3 = «вулиці», зберігається в третьому рядку E. Його можна отримати, помноживши унітарне кодування слова на E. Для подальшого використання вкладень, проблема може бути з обробкою слів, які не містяться в E. Вони називаються Out of vocabulary (OOV) words - слова, які не включені до словаря.

Більшість статистичних та основаних на правилах алгоритмів для обробки природної мови розглядають слова як атомарні сутності. Це транслюється в векторне представлення розміром словника і з одиницями на індексі даного слова. Але цей підхід має проблему, яка полягає в тому, що він не визначає схожості між двома словами. Тому, якщо модель бачить «готель» в контексті, вона не може використовувати цю інформацію, коли вона бачить «мотель» в тому ж місці під час тестування.

Замість цього спрощеного підходу у 1957 році була представлена потужна ідея представлення кожного слова за допомогою його сусідів. Це одна

з найбільш успішних ідей сучасної статистичної обробки природної мови, яка широко використовувалася в NLP, наприклад, кластеризація Брауна. Іншим прикладом є м'яка кластеризація, викликана такими методами, як прихований розподіл Діріхле. Споріднена ідея моделей нейронної мови полягає в тому, щоб спільно вивчати такі контекстно-захоплюючі векторні представлення слів і використовувати ці вектори, щоб передбачити, наскільки ймовірним є поява слова з урахуванням його контексту.

Тим часом, існує безліч різних способів створення вкладень слів. Word2Vec, GloVe і fastText є найбільш популярними підходами і будуть описані в наступних розділах.

1.8 Нейронні мережі прямого поширення

У цьому пункті основна увага приділяється концепції ANN і нейронних мереж прямого поширення (FNN), оскільки це перший розроблений тип мережі. ANN взяли людський мозок як приклад і засновані на нейронах Маккаллоха і Пітса, які були створені в 1943 році.

На рисунку 1.5 показана математична модель одного нейрона. ANN можуть вирішувати різні типи завдань класифікації і безперервного прогнозування (регресії). ANN - це загальний термін для нейронних мереж, таких як FNN, які використовуються в AI. FNN є мережами без циклів і засновані на одноріневому і багаторівневому персептронах.

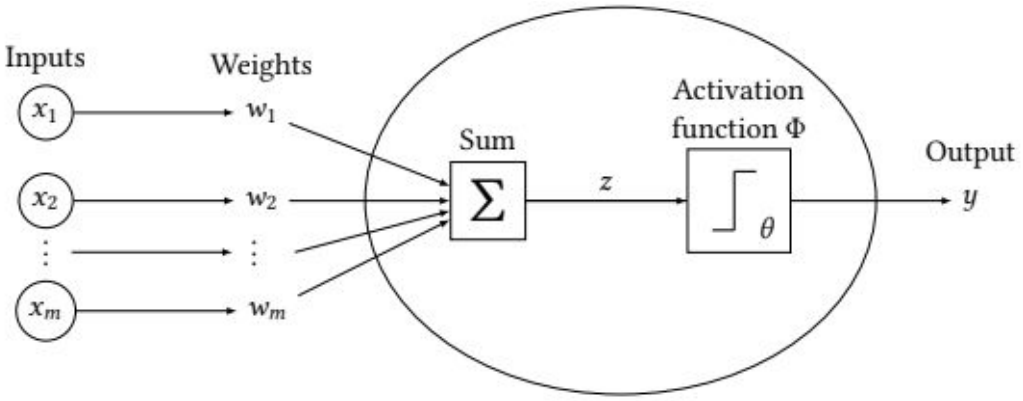

Рисунок 1.5 – Нейрон Маккаллоха і Пітса зіставляє m вхідних значень з одним виходом y

Нейрон Маккаллоха і Пітса приймає вектори з довжиною m в якості вхідних даних і примножує кожне значення x_1, x_2,..., x_m на відповідний вагу w_1, w_2,..., w_m. Нейрон має здатність активуватися, коли загальна сума перевищує заданий поріг θ. Активація в даному конкретному випадку означає, що нейрон видає 1, в іншому випадку - 0.

Спочатку нейрон обчислює логіт-регресію z вхідного вектора x_1, x_2,..., x_m:

$$z = \sum_{i=1}^{m} w_i x_i \tag{1.2}$$

Тоді результат y визначається функцією активації Φ, яка в цьому випадку є функцією Хевісайда.

1.9 Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) є подальшим розвитком в області ANN. На практиці RNN часто використовуються для послідовних входів. Особливо в мові, слова повинні бути оброблені крок за кроком, щоб відновити

їх контекст. Також, RNN виявилися дуже хорошими в задачах перекладу, так як вони можуть передбачити наступне слово по більш раннім словам. Щоб контролювати процес аналізу зразка з іншим, у RNN є приховані блоки для зберігання векторів стану. Ці вектори містять інформацію про історію послідовності. Таким чином, вихід $\widehat{y}$ для RNN залежить від фактичного входу x_t і попередніх входів (x_1, ..., x_{t-1}). Крім того, приховані стани представлені прихованою векторною послідовністю h = (h_1, ..., h_t). Вихідні дані RNN складають послідовність y = (y_1, ..., y_t), але не обов'язково представляють вихідні дані для загальної задачі. RNN також можуть використовуватися для прогнозування єдиної мети для вхідної послідовності. Прихований стан h_t розраховується за формулою:

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{1.3}$$

H позначає приховану функцію шару, яка часто є гіперболічним тангенсом (tanh). Крім того, W позначає є вагову матрицю між шарами, а b представляє вектор зміщення. Вихід розраховується аналогічно результату багатошарового перцептрона шляхом множення ваги на результат прихованого стану і додавання зсуву.

Тому прихований стан схожий на прихований шар. Результат розраховується аналогічно:

$$y_t = W_{hy}h_t + b_y \tag{1.4}$$

Вихід y_t і прихований стан h_t пов'язані з ваговою матрицею W_{hy}. Ця структура шарів порівняна з одношаровим перцептроном. RNN можна розглядати як складений багатошаровий перцептрон з циклами, оскільки результат попереднього введення також є входом для поточного екземпляра.

Модель RNN показана на рисунку 1.6. Тут вагові матриці W представлені в загальному форматі для поліпшення читабельності представлення. Зазвичай вони повинні бути адаптовані до базової схеми: $W_{h_{t-2}h_{t-1}}$, $W_{h_{t-1}h_t}$, $W_{h_th_{t+1}}$,...

Процедура навчання також аналогічна багатошаровому перцептрону. Спочатку мережа ініціалізується, і початковий прихований стан h_0 зазвичай встановлюється рівним 0. Потім прогнозування $\widehat{y}$ для вхідного x_t обчислюється в прямому поширенні. Після цього відбувається оцінка істинного результату y. При зворотному поширенні похідна помилки по вагам обчислюється з використанням методу зворотного поширення з розгортанням мережі в часі (BPTT).

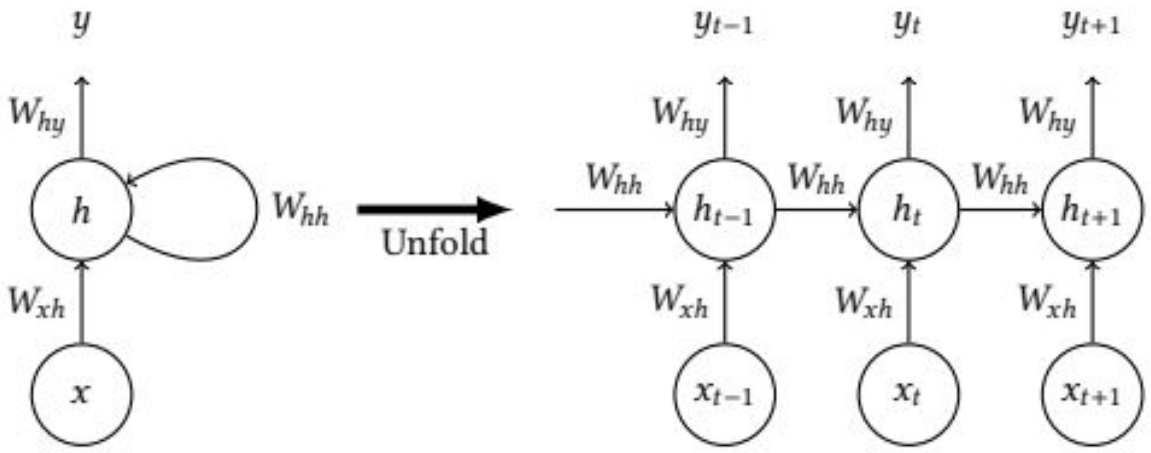

Рисунок 1.6 – Рекурентна нейронна мережа прогнозує результат у послідовності x_1 , ..., x_{t-1}

Одним з недоліків стандартних RNN є проблема зникнення і вибуху градієнтів. Перша проблема змушує RNN відстежувати залежності в більш довгих вхідних послідовностях. Це пов'язано з двома причинами. По-перше, гіперболічний тангенс і сигмовидна функція, які часто використовуються в RNN для активації, насичуються дуже швидко, і тому їх градієнт стає ближче

до 0. По-друге, застосовуючи BPTT, градієнт експоненціально зменшується шляхом множення його на повторювані вагові матриці. Це також змушує градієнт прагнути до нуля дуже швидко. Феномен вибуху градієнта приводить до значних коливань ваги мережі і збільшує час навчання, що може привести до відмови мережі.

Висновки до розділу 1

У даному розділі була сформована задача даної роботи, а також були розглянуті базові принципи обробки природної мови (Natural language processing), та сучасні методі використання машинного навчання та нейронних мереж для обробки природної мови.

Задача обробки та класифікації природної мови була сформована вже досить давно і за останні 5 років інженерами і науковцями був зроблений великий крок вперед в даній області. Аналіз даних з повідомлень в соцмережах є одним з найбільш яскравих прикладів використання методів NLP, оскільки можна працювати з великими масивами даних в реальному часі, та приносити користь соціуму за допомогою фільтрації небажаного контенту, рекомендаційних сіток.

Однак це є не єдиним напрямом використання NLP. NLP широко використовується в голосових помічниках, автовідповідачах, оцифровуванні мови, створення субтитрів, знайдення взаємозв’язків у текстах, виведення головної ідеї з тексту, аналіз дискурсу, прогнозування та підказки при друці, виправлення граматичних, синтаксичних, лексичних помилок.

2 МЕТОДИ МАШИННОГО НАВЧАННЯ ДЛЯ ОБРОБКИ ПРИРОДНИХ МОВ

2.1 Методі представлення тексту у векторі

2.1.1 Word2Vec

Модель word2vec була представлена Міколовим в 2013 році. Ідея полягає в тому, щоб перетворити слово в безперервний вектор, який також представляє локальний контекст слова. Вкладення word2vec не мають недоліку унітарно кодованих векторів слів, які можуть тільки розпізнавати, чи є слово точно таким же чи ні. Word2Vec робить ці відмінності більш помітними, додаючи контекст слова.

Представлення слів включають в себе семантичну і синтаксичну інформацію, яка витягується з контекстних слів. Наприклад, косинусна відстань є підходящою функцією для обчислення різниці між векторами слів. У просторі впровадження можливо, що однакові зв’язки між словами можуть бути представлені однаковим відстанню і напрямком. Наприклад, англійські слова “Man” і “Woman”мають однакову відстань до “King” і “Queen”.

Це означає, що контекстуальна відмінність між “Man” і “Woman” можна порівняти з “King” і “Queen”. Слова, що відповідають чоловічій статі, використовуються в контексті “Man” і “King”, тоді як “Woman” та “Queen” оточені більш жіночими словоформами. Таким чином, синтаксис і семантика слова включені в представлення слова. Це реальна сила вкладення слів або word2vec. Це полегшує пошук синонімів і більш легке визначення передбачуваного значення промовця.

У початковому релізі word2vec пропонуються два різних підходи до вивчення векторних слів з текстових корпусів з використанням лексики N.

Перший підхід - це неперервна мультимножина зі словами (CBOW), який передбачає слово на основі його контексту. Другий підхід називається Skip-gram і прогнозує контекст слова. Обидва способи мінімізують обчислювальну складність і засновані на FNN. Рисунок 2.1 зображує обидві архітектури більш детально - CBOW використовує контекстні слова для прогнозування цільового слова. Skip-gram вимагає слово для моделювання контексту.

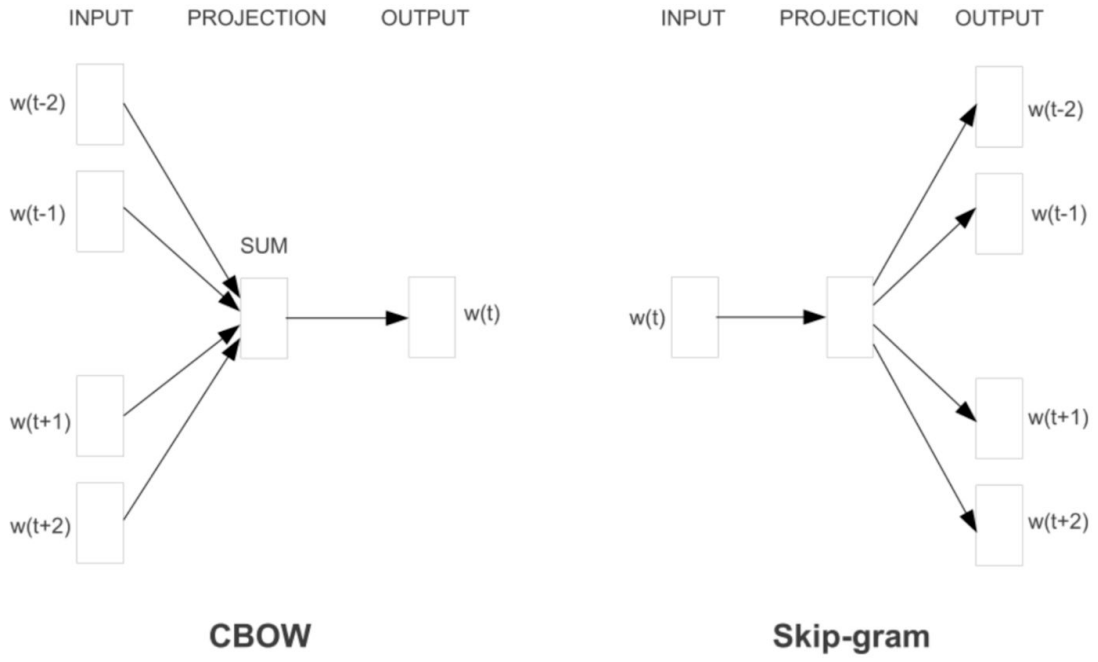

Рисунок 2.1 - FNN-архітектура моделі безперервного набору слів (CBOW) і моделі скіп-грами

Щоб обчислити векторне представлення слова, нейронна мережа CBOW використовує контекстні терміни зліва w_{t-2}, w_{t-1} і праворуч w_{t+1}, w_{t+2} слова w_t в якості вхідних даних. Це вікно слова може бути відрегульовано для збільшення або зменшення розміру локального контексту, пов'язаного зі векторного представлення v_t. На практиці перевагу надають контекстному

вікну 5 для CBOW і 10 для Skip-gram. CBOW приймає one-hot кодування кожного вхідного слова та передає їх в нейронну мережу для прогнозування one-hot кодування цільового слова w_t. Нейронна мережа прямого поширення (FNN) моделює місце 1 в вихідному векторі, що можна розглядати як задачу класифікації.

Мережа, яка використовується в Skip-gram, змінює місцями вхід і вихід. Цільове слово тепер є вхідним, і FNN прогнозує контекст слова. Skip-gram призводить до кращих результатів для невеликих корпусів, тоді як CBOW більш ефективний і рекомендується для великих текстових наборів. Для навчання векторів слова обидва підходи використовують багатошаровий перцептрон з одним прихованим шаром.

Отже, мережа має дві вагові матриці. Перший $w^{(1)}$ знаходиться між вхідним шаром і прихованим шаром. $W^{(2)}$ - друга матриця, яка з'єднує прихований та вихідний шаром. Кількість нейронів у прихованому шарі встановлено на 300, і активація softmax в вихідному шарі використовується в обох підходах.

Щоб витягти вектор слова v_t для слова w_t в архітектурі Skip-gram, береться вектор t-го рядка з матриці $w^{(1)}$ = E. CBOW бере вектори рядків з $w^{(1)}$ для своїх вхідних слів і усереднює їх. Число прихованих нейронів рівне вимірам embedding, яке визначається довжиною рядка $w^{(1)}$. CBOW моделює ймовірність p (wt | C) для контексту C = $\{w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}\}$. Навпаки, Skip-gram обчислює для кожного контекстного слова $w_c \in$ C, щоб створити word embedding v_t.

Термін $v_t^T u_c$ вказує, як мережа перетворює вхідне слово з one-hot кодуванням в вихідне слово з one-hot кодуванням в контексті. V_t множиться на базове контекстне представлення u_c для w_c, яке виходить в стовпці с матриці $w^{(2)}$. У зв'язку з тим, що N вказує довжину загального словника, обчислення

softmax може бути дорогим для великих N. Ієрархічний softmax використовується для подолання цього.

2.1.2 Глобальні вектори для представлення слів (GloVe)

Глобальні вектори для представлення слів (GloVe) дійсно мають інший підхід для побудови векторів слів. Спочатку вони були запропоновані в 2014 році Пенінгтоном і були зконцентровані не тільки на місцевому контексті слова. Навчання векторів GloVe здійснюється на основі глобальної міжсловісної матриці X_{ij} . Ця матриця зберігає частоту, з якою два слова w_i і w_j зустрічаються в одному і тому ж контексті в корпусі. Отже, враховується не тільки локальний контекст, який визначається матрицею спільного використання, а й статистичні аспекти корпусу. По-перше, матриця створюється на етапі ініціалізації, коли кожен текст проглядається один раз. Оскільки матриця спільного використання вимагає великого обсягу пам'яті при аналізі великої кількості документів, зберігаються тільки словосполучення, по крайній мірі, з одним збігом.

Потім, ймовірності співвідношення входжень розширюються з $P_{ij} = P(i|j) = \frac{X_{ij}}{X_i}$, як показано на рисунку 2.2. Рисунок ілюструє вірогідність для слів “ice” і “steam”, викладених в реальному корпусі. Наприклад, слова “ice” і “solid” зустрічаються з вірогідністю $1{,}9 \times 10^{-4}$, тоді як “steam” і “solid” зустрічаються рідше. Напротив, “steam” і “gas” зустрічаються частіше, ніж “ice” і “gas”, на що вказує відношення P (k | ice) / P (k | steam). Ділення на нуль неможливо, тому що матриця X_{ij} зберігає тільки ті частоти збігів, які хоча б дорівнюють одиниці. Якщо обидва слова або жодне з слів не відносяться до слова k, відношення близьке до одиниці, тому що обидві ймовірності мають однакове значення. Ця ситуація показана “water” і “fashion”. “Ice” і “steam”

часто використовуються в контексті “water”, але рідше зустрічаються в “fashion”.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Рисунок 2.2 - Приклад ймовірностей спільного виникнення слово-слово, які розраховані на корпус 6 мільярдів слів. Умовні ймовірності чотирьох контекстних слів k дають терміни „ice” та „steam”

Після обчислення ймовірностей збігів починається етап навчання. GloVe-підхід вивчає вектори слів відповідно до коефіцієнта збігу контекстного слова k.

Автори використовують лог-білінійну модель для навчання векторів слів v_i і v_j. Вони використовують скалярний добуток між різницею v_i - v_j і вектором контексту $\overline{v_k}$, щоб фактично передбачити коефіцієнт збігів. Подальше перетворення рівняння, таке як переписування дробу $\frac{P_{ik}}{P_{jk}}$ в різницю логарифмів, призводить до задачі регресії, яка намагається мінімізувати функцію витрат J.

$$J = \sum_{i,k=1}^{N} f(X_{ik}) * (v_i^T \overline{v_k} + b_i + \overline{b_k} - log(X_{ik})) \quad (2.1)$$

Залежить від вектора v_i, вектора контексту $\overline{v_k}$, їх зсувів b_i, b_k, кількості всього словника N і вагової функції f (X_{ik}). Функція f (X_{ik}) використовується,

щоб уникнути переваги рідкісних і частих збігів. Після навчання можна отримати вектор слова v_i для кожного слова в корпусі. Автори стверджують, що GloVe перевершує word2vec в різних задачах NLP, таких як аналогія слів, розпізнавання іменованих об'єктів або схожість слів. Це пов'язано з тим, що GloVe також включає в себе статистичну інформацію про слова в корпусі.

2.1.3 FastText

FastText - це бібліотека для створення векторних представлень слів, розроблена дослідницькою групою Facebook AI.

Цю бібліотеку можна порівняти з word2vec, але вона використовує символьні n-грами в якості вхідних даних, а не тільки саме слово. Це покращує здатність створювати вектори навіть для вигаданих або рідкісних слів. Крім того, цей метод підходить для мов, які мають багато перегинів і засновані на одних і тих же невеликих частинах слова, таких як фінська або турецька. FastText використовують CBOW або Skip-грами, які обговорювалися в розділі.

У word2vec Skip-gram прогнозує вектор контекстних слів (вихідний), отримавши фактичне слово w, використовуючи закодовані вектори слів. Ці представлення не мають спільних рис між словами, тому що вони можуть тільки вказати, чи є слово тим же самим чи ні. FastText навпаки розбиває свої вхідні слова на набори символів n-грам і передає їх в мережу. Символьні n-грами можуть бути розділені між різними словами. Наприклад, слово «дослідження» ділиться на “<до, дос, осл, осл, слі, лід, ідж, дже, жен, енн, ня>” якщо n дорівнює 3. Таким чином, «<до» і «ня>» містять спеціальні символи «<» і «>» що вказують початок і кінець слова. Це зроблено для того, щоб проілюструвати різницю між словами, які містяться у вигляді n-грами символів в іншому слові, наприклад «слід» в «дослідженні».

Для отримання вектора слова для слова w_t в даному наборі документів вводиться функція оцінки s(w_t, w_c). Він обчислює суму символів n-грам, помножених на навколишнє слово $w_c \in$ {..., wt - 2, wt - 1, wt + 1, wt + 2, ...} слова w_t. Кількість контекстних слів w_t задається як довжина вікна виведення Skip-грам. Skip-грам-версія fastText розраховує умовну ймовірність

$$p(w_c \mid w_t) = \frac{e^{s(w_t, w_c)}}{\sum_{i=1}^{N} e^{s(w_t, w_c)}} \tag{2.2}$$

замінивши результат скалярного множення між $u_c v_t^T$ на s(w_t, w_c). Нехай G - словник n-грам і $G_t \subset$ {1, ..., G} це масив символьних n-грам для слова w_t. Тоді результат

$$\mathrm{s}(w_t, w_c) = \sum_{g \in G_t} z_g^T c_c \tag{2.3}$$

визначається для кожного контекстного слова w_c. Замість унітарно-кодованого слова вектора набір символів n-грамів прогнозує контекстне слово у FastText. Усі вектори z_d де g $\in$ G_t є символьними вкладеннями n-грам слова w_t. Їх набір будує ембедінг слова v_t. Через те, що загальний набір n-грам для великого корпусу дуже великий, хешована версія символів n-грам використовується як вхідна інформація. Перевагою FastText є продуктивність. Час навчання представлень слів FastText короткий, і навіть можна навчити вектори на стандартному багатоядерному процесорі в реальні терміни. Більше того, точність FastText у завданнях NLP, таких як семантичний аналіз або класифікація за тегами, одна з найвищих серед найсучасніших методів.

2.2. Глибокі контекстуалізовані представлення слів

Ембедінги слів, як показано вище, являють собою сучасне представлення слів в числовому форматі. Проте, через їх поверхневу мережеву структуру, вони можуть включати тільки довколишній локальний контекст, семантику і синтаксис слова. Наприклад, CBOW або скіп-грами на практиці включають від 5 до 10 контекстних слів, які впливають на ембедінг слів. Тому їм не вистачає даних, щоб проектувати все глобальні зв'язки слів. Одні і ті ж слова часто використовуються в багатьох документах в різних лінгвістичних контекстах, наприклах. У 2018 році було розроблено «представлення слова з глибоким контекстом», яке долає цей недолік. Ці представлення слів створюються за допомогою глибокої двобічної мовної моделі (biLM). Крім того, модель попередньо навчена на великому текстовому корпусі.

По-перше, представлю короткий опис мовних моделей (LM). Для набору з N слів (w_1, w_2, ..., w_N) LM обчислює ймовірність всієї послідовності. Вони використовують умовну ймовірність наступного слова w_i, з огляду на історію попередніх слів (w_1, w_2, ..., w_{i-1}):

$$\mathrm{p}(w_1, w_2, ..., w_N) = \prod_{i=1}^{N} p(w_i | w_1, w_2, ..., w_{i-1}) \quad (2.4)$$

спочатку ембедінг слова v_i, передається в LSTM з K шарами, що обробляє послідовність в прямому напрямку. Кожен шар k = 1 , … , K виводить контекстно-залежне представлення слова $h_{i,k}$ для слова w_i. Верхній рівень в LSTM навчений прогнозувати наступне слово w_{i+1} і використовує FNN з рівнем softmax зверху для обчислення вихідних даних. Інший LSTM обробляє послідовність у зворотному напрямку, як в двунаправленному LSTM. Він прогнозує попереднє слово w_{i-1}. Щоб об'єднати обидва напрямки в biLM, вони максимізують спільну логарифмічну ймовірність як критерій оптимізації:

$$\sum_{i=1}^{N}(log\, p\,(w_i|\, w_1\,,\, w_2\,,\, ...,\, w_N;\, \theta_x,\, \theta_{LSTM}\,,\, \theta_s\,)) + log\, p\,(w_i|\, w_{i+1}\,,\, w_{i+2}\,, \\ ...,\, w_N;\, \theta_x,\, \theta_{LSTM}\,,\, \theta_s\,)) \tag{2.5}$$

θ_x представляє параметри для ембедінгов слів E, а θ_s відзначають параметри для softmax-FNN. У LSTM є окремі параметри для прямого і зворотного напрямку. Шар softmax приймає стан LSTM для прогнозування слова w_i по заданій вхідній послідовності.

2.3 Наївний Байєсовський класифікатор

У машинному навчанні наївні байєсовські класифікатори представляють собою сімейство простих «імовірнісних класифікаторів», заснованих на застосуванні теореми Байеса з сильними (наївними) припущеннями про незалежність між ознаками. Вони є одними з найпростіших моделей байєсівської мережі.

Наївний Байес широко вивчався з 1960-х років. Він був введений (хоча і не під цією назвою) в співтоваристві з пошуком тексту на початку 1960-х років і залишається популярним (базовим) методом категоризації тексту - проблема оцінки документів як що належать до тієї чи іншої категорії (наприклад, як спам, спорт чи політика і т. д.) з частотою слів в якості ознак. З відповідною попередньою обробкою, вона конкурентоспроможна в цій області з більш просунутими методами, включаючи машини опорних векторів.

Це один з методів математичного виведення, коли у нас є суспільство або явище, що йде за розподілом ймовірностей на основі невідомого фіксованого параметра. Ми хочемо оцінити період цього параметра або перевірити

конкретну гіпотезу за допомогою даних випадкової вибірки, взятих з цієї спільноти, де у нас є попередня інформація про ймовірність (до вибірки) для цього параметра, який приймає різні значення в якості випадкової величини з розподілом ймовірності, отримані з використанням цієї теорії, а розподіл ймовірностей, отриманий після вибірки, відомий як апостеріорний розподіл, який представляє собою зведення даних, отриманих з вибірки.

$$P\,(C / F) = P\,(C / F) * P\,(C) / P\,(F) \qquad (2.6)$$

У формулі 2.6, F є вектором атрибутів користувача, а C є класом (спамер / не спамер) користувача.

2.4 Алгоритм випадкового лісу

Випадкові ліса або ліса випадкових рішень - це метод навчання ансамблю для класифікації, регресії і інших задач, який працює шляхом побудови безлічі дерев рішень під час навчання і виведення класу, який є режимом класів (класифікація) або середнім прогнозом (регресія) окремих дерев. Випадкові ліса рішень коригують звичку дерев рішень відповідати їх тренувального набору.

Перший алгоритм для випадкових лісів рішень був створений Тін Кам Хо з використанням методу випадкових підпросторів, який у формулюванні Хо є способом реалізації підходу «стохастичної дискримінації» до класифікації, запропонованої Юджином Кляйнберг.

Він майже не потребує жодної підготовки даних або будь-якого досвіду моделювання, і дозволяє аналітикам отримувати ефективні моделі.

При використанні випадкового лісу створюється багато дерев рішень. Кожна частина даних подається в кожне дерево рішень. Превалюючий

загальний результат використовується для кожного меморандуму і остаточного результату. Новий елемент управління подається в усі дерева, і для кожної моделі оцінки приймається більшість голосів, помилка оцінюється в випадках, які не використовуються при побудові дерева. OOB (Out-of-bag) називається помилкою прогнозування, одержується в процентах.

2.5 Мультиноміальна логістична регресія

При обробці природної мови класифікатори, що базуються на Мультиноміальній логістичній регресії, зазвичай використовується в якості альтернативи Наївному Баєсовому класифікатору, оскільки вони не передбачають статистичну незалежність випадкових величин (зазвичай званих ознаками), які служать предикторами. Однак навчання в такій моделі повільніше, ніж для наївного байєсівського класифікатора, і, отже, може бути недоречним, враховуючи дуже велику кількість класів для навчання. Зокрема, навчання в наївному байєсівської класифікаторі - це простий підрахунок кількості одночасних появ об'єктів і класів, в той час як в мультиномальних ЛР класифікаторах ваги, які зазвичай максимізуються з використанням максимальної апостеріорної оцінки, повинні навчатися з використанням ітераційної процедури.

У статистиці поліноміальна логістична регресія - це метод класифікації, який узагальнює логістичну регресію для мультикласових задач, тобто з більш ніж двома можливими дискретними результатами. Тобто це модель, яка використовується для прогнозування ймовірностей різних можливих результатів категоріально розподіленої залежною змінною, враховуючи набір незалежних змінних, які можуть бути речовими, двійковими, категоріальним і т.д.

Поліноміальна логістична регресія відома під безліччю інших імен, у тому числі багаточленною логістичною регресією, мультіклассовою, регресією softmax, mlogit, класифікатором максимальної ентропії (MaxEnt) і умовної моделлю максимальної ентропії.

2.6 Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) були спочатку розроблені для комп'ютерного зору, наприклад для аналізу зображень. З 2011 року вони стають все більш популярними в NLP. У 2014 році CNN була успішно використана в задачах НЛП, таких як класифікація пропозицій і моделювання пропозицій.

CNN складається з згортаючого і об'єднуючого шарів. Вхідні дані передаються в ланцюжок з одного або декількох згортальних шарів, за якими слід шар пулу. Ця структура шару зі сверточним пулом може повторюватися для створення більш глибоких змін, які утворюють глибокі CNN. Згорточний шар застосовує шаблон фільтра до вхідних даних. Це допомагає зіставити вхідні дані з базовою картою об'єктів і знайти локальні зв'язки між ними. Шаблон фільтра зменшує кількість ваг в мережі, зіставляючи частини вхідних даних однієї функції. Крім того, шар об'єднання скорочує карту об'єктів і об'єднує семантично подібні об'єкти разом. Згорточний шар може складатися з більш ніж одного шаблону фільтра, що показано накладеної структурою на рисунку 2.3.

Слова передаються в мережу і фільтруються згорточними шарами. Після об'єднання FNN з шаром softmax зверху прогнозує клас.

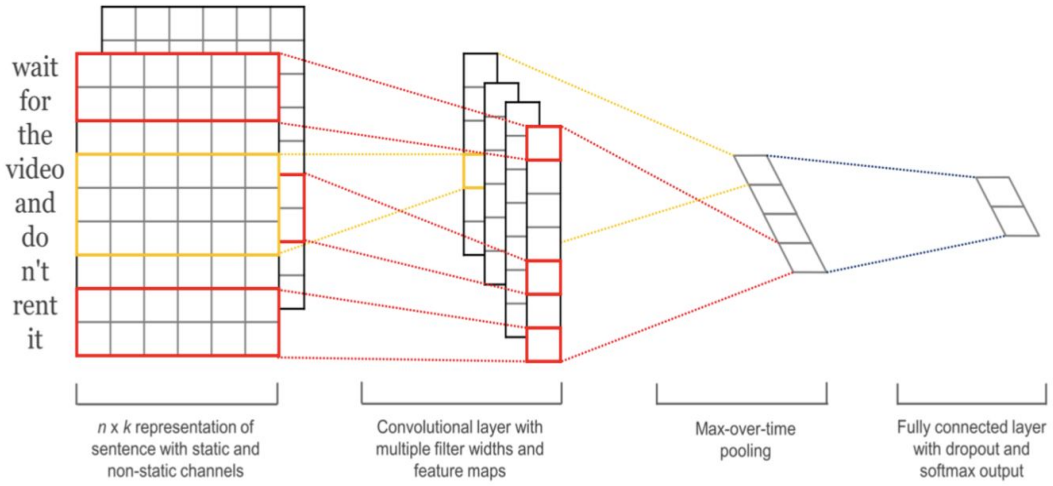

Рисунок 2.3 - Модель згорточної нейронної мережі, яка використовується в класифікації слів

2.7 Інструменти для створення програмного забезпечення

Оскільки нам потрібно розробити систему, яка буде складатися з декількох сервісів, які матимуть взаємодію в мережі. В межах даної роботи був проведений аналіз існуючих мов програмування, бібліотек та фреймворків.

Проаналізував графік популярності мов програмування, згідно з даними сайту http://pypl.github.io/ та виходячи з даних опитування stackoverflow 2019, на графіку, зображеному на рисунку 2.4 бачимо, що в останні роки Python неухильно набирає популярність і в даний час бореться за стан одного з найпопулярніших мов програмування в світі. Схвалений для додатків з веб-розробки, для сценаріїв і автоматизації процесів, Python швидко стає кращим вибором серед розробників для проектів штучного інтелекту, машинного навчання та глибокого навчання.

Одним з аспектів, який робить Python таким популярним вибором в цілому, є його велика кількість бібліотек і середовищ, які полегшують

написання коду і економлять час розробки. Пакети і бібліотеки для машинного навчання постійно вдосконалюються та мають підтримку.

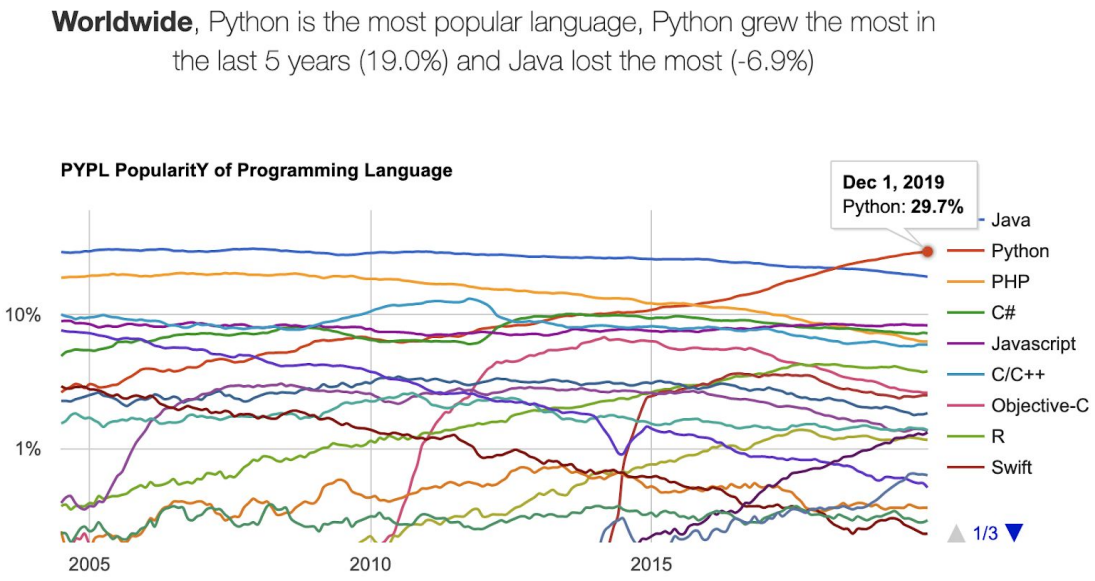

Рисунок 2.4 - Ріст популярності мов за даними pypl

NumPy, який використовується для наукових обчислень, SciPy для розширених обчислень і scikit-learn для інтелектуального аналізу даних і аналізу, є одними з найбільш популярних бібліотек, які працюють разом з такими потужними середовищами, як TensorFlow, CNTK і Apache Spark. З точки зору машинного навчання і глибокого навчання, ці бібліотеки і фреймворки по суті адаптовані для Python, в той час як деякі, такі як PyTorch, написані спеціально для Python.

Висновки до розділу 2

У даному розділі були розглянуті математичні методі представлення слів та тексту природної мови на математичній мові.

Був проведений порівняльний аналіз основних популярних сьогодні методів для перетворення слів у векторі, такі як Word2Vec, GloVe, FastText. А також були розглянуті методі машинного навчання для класифікації природних мов, та пошуку найближчого сусіда у векторних просторах.

Використання векторних представлень слів у обробці природних мов дає велику перевагу перед більш простими методами, як мішок слів і дозволяє знаходити додаткові, неочевидні взаємозв’язки між текстами.

3 РОЗРОБКА МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ТА ПРОЕКТУВАННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ ДЛЯ ОБЧИСЛЕНЬ

3.1 Вибір програмного забезпечення та інструментів розробки

На основні проаналізованої інформації, що описано у розділі 2.7, для розробки системи було обрано програмну мову Python, та широкий спектр бібліотек, для роботи з даними, такі як nltk, fasttext, pytorch, sklearn та інші.

3.2 Відбір даних

Для класифікації було обрано пости в Твітері, написані англійською мовою, оскільки значна частина контенту саме англійська, а також є багато вже нетренованих на англійську мову моделей та розмічених даних.

Як вже було зазначено у 2 розділі, для того щоб “навчити” програму класифікувати дані, потрібні великі об’єми даних. З цією метою, для класифікації неякісного контенту було зібрано та розмічено 40 тисяч повідомлень в мережі Твітер, написані різними людьми, з різною географією з яких близько 40% ідентифіковані як неякісний контент.

Для класифікації даних за категоріями, було зібрано та розмічено декілька тисяч повідомлень, але цього виявилось недостатньо, щоб натренувати класифікатор, і він видавав дуже слабкий результат, тому була знайдена вже розмічена вибірка даних з kaggle, яка містить близько 200 тис. заголовків новин з 2012 по 2018 рік, отриманих від HuffPost. Цей контент найбільш відповідає повідомленням користувачів Твітеру за об’ємом та структурою.

3.2 Опис Твітера, API, структури повідомлень

Twitter - платформа для мікроблогів і соціальна мережа, в якій користувачі розміщують і взаємодіють з повідомленнями, відомими як «твіти». Спочатку твіти були обмежені 140 символами, але в 2017 році ця межа був збільшена до 280 майже для всіх мов. Зареєстровані користувачі можуть публікувати, “лайкати” і “ретвітити” твіти, але незареєстровані користувачі можуть тільки читати їх. Користувачі мають доступ до Twitter через веб-інтерфейс, через службу коротких повідомлень (SMS) або мобільні додатки.

Крім того, Твіттер предоставляє розробникам доступ до більшості з їх доступних web API. Одними з них є “Пошук Твітів”, “Фільтрація ріал-тайм Твітів”, “Приватні повідомлення”, “Твіттер для веб-сайтів”, “Рекламне API”.

Для даної роботи необхідним є отримання всіх твітів за заданими фільтрами, наприклад хештегом, в реальному часі. Така опція дозволена в аккаунтах для розробників, який і було створено.

Сервіс надає доступ до апі, в якому повертає результат у форматі JSON. При отриманні твітів в реальному часі, ми отримуємо об’єкти твітів.

Bcs API Твітера, які повертають твіти, надають ці дані, закодовані з використанням JavaScript Object Notation (JSON). JSON заснований на парах ключ-значення з іменованими атрибутами і пов'язаними значеннями. Ці атрибути і їх стан використовуються для опису об'єктів.

Twitter обслуговує безліч об'єктів в форматі JSON, включаючи Tweets (твіти) і Users (користувачі). Всі ці об'єкти інкапсулюють основні атрибути, які описують об'єкт. У кожного твіту є автор, повідомлення, унікальний ідентифікатор, відмітка часу, коли він був опублікований, а іноді і гео-метадані, що надаються користувачем. У кожного користувача є ім'я в Twitter, ідентифікатор, кількість підписників і найчастіше опис аккаунта.

З кожним твітом також генеруються об'єкти-сутності, які представляють собою масиви загального вмісту твіта, такі як хештеги, згадки, медіа та посилання. При наявності посилань корисне навантаження JSON також може надавати метадані, такі як повністю розгорнутий URL-адресу, заголовок і опис веб-сторінки.

Таким чином, крім самого текстового вмісту, твіт може мати більше 150 атрибутів, пов'язаних з ним. Як приклад можемо розглянути твіт на рисунку 3.1.

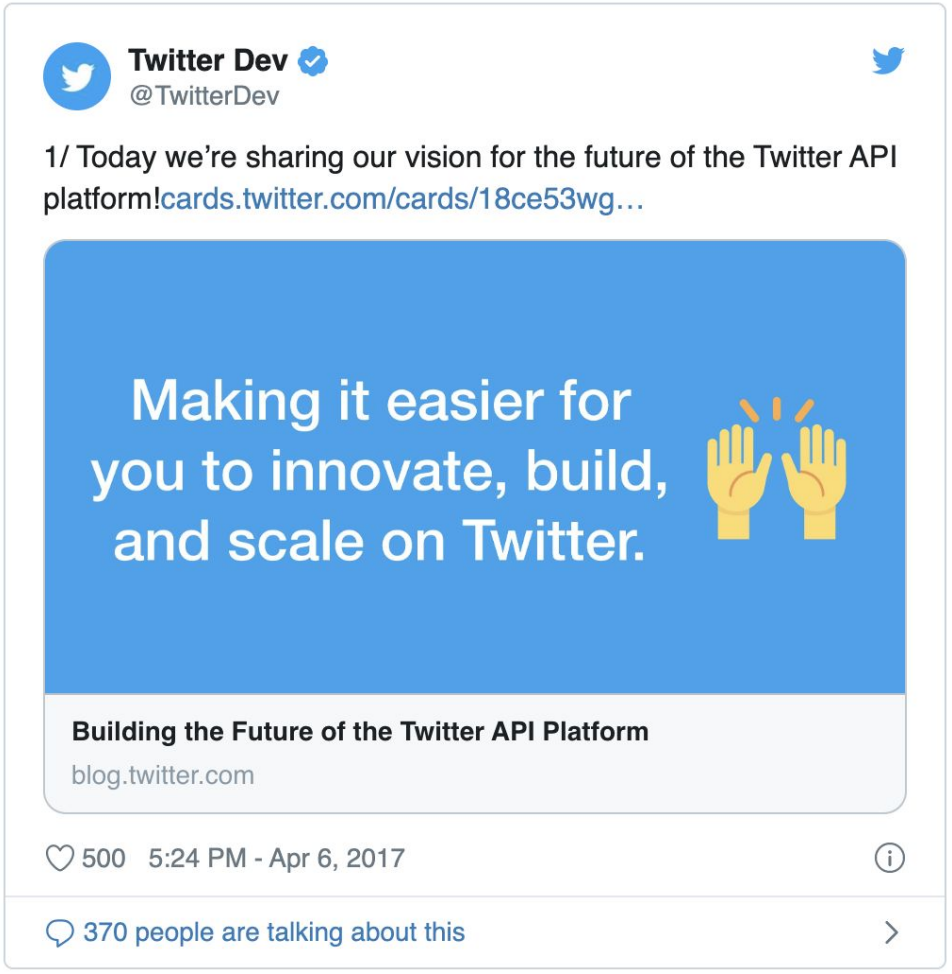

Рисунок 3.1 - Приклад повідомлення користувача в Твітері

Цей твіт ілюструється JSON об’єктом наведеним на рисунку 3.2 нижче.

```
{
  "created_at" : "Thu Apr 06 15:24:15 +0000 2017" ,
  "id_str" : "850006245121695744" ,
  "text" : "1\/ Today we\u2019re sharing our vision for the future of the Twitter API platform!\nhttps:\/\/t.co\/XweGngmxlP
  "user" : {
    "id" : 2244994945 ,
    "name" : "Twitter Dev" ,
    "screen_name" : "TwitterDev" ,
    "location" : "Internet" ,
    "url" : "https:\/\/dev.twitter.com\/" ,
    "description" : "Your official source for Twitter Platform news, updates & events. Need technical help? Visit https:\/\
  } ,
  "place" : {
  } ,
  "entities" : {
    "hashtags" : [
    ] ,
    "urls" : [
      {
        "url" : "https:\/\/t.co\/XweGngmxlP" ,
        "unwound" : {
          "url" : "https:\/\/cards.twitter.com\/cards\/18ce53wgo4h\/3xo1c" ,
          "title" : "Building the Future of the Twitter API Platform"
        }
      }
    ] ,
    "user_mentions" : [
    ]
  }
}
```


Рисунок 3.2 - JSON формат об’єкту повідомлення в Твітері

При прийомі даних повідомлень, основним об'єктом є об'єкт Tweet, який є батьківським об'єктом для декількох дочірніх об'єктів. Наприклад, всі твіти містять об'єкт “User” (користувач), який описує, хто є автором даного твіту. Якщо Tweet має гео-тег, в нього буде включено об'єкт “place” (місце). Кожен твіт включає в себе об'єкт “entities” (сутності), який інкапсулює масиви хештегов, згадок користувачів, URL-адрес, грошових тегів і медіа. Якщо в твіті є яке-небудь “прикріплене” або “нативне” медіа (фотографії, відео, анімований GIF), буде об'єкт “extended_entities” (розширені сутності).

На верхньому рівні JSON повідомлення має основні атрибути Tweet на, в тому числі, коли був опублікований Tweet, його унікальний ідентифікатор і текст повідомлення. Також на цьому кореневому рівні знаходяться інші фундаментальні (дочірні) об'єкти, такі як об'єкти користувача та сутностей.

Крім цього, Твіт може бути “Ретвітом”, що значит що він є опублікованим повідомленням з профілю іншого користувача. Ретвіт - це дія,

щоб поділитися твітів зі своїми підписниками, і ніякий інший новий контент не може бути доданий. Крім того, нове місце розташування не може бути надано з ретвітів. Хоча “оригінальний” твіт може мати геотеги, об'єкти “гео” і “місце” ретвіту завжди будуть нульовими.

Ретвіти завжди містять два об'єкти твітів. “Retweeted” (оригінальний) Tweet надається в об'єкті “retweeted_status”. Об'єкт кореневого рівня інкапсулює сам ретвіт, включаючи об'єкт користувача для облікового запису, яка виконує дію ретвіту, і час ретвіту.

Відображаючи ієрархію JSON вище, на рисунку 3.3 зображені основні об’єкти Твіту.

- Tweet - Also referred to as a ‘Status’ object, has many ‘root-level’ attributes, *parent* of other objects.
  - User - Twitter Account level metadata. Will include any available account-level enrichments, such as Profile geo.
  - Entities - Contains object arrays of #hashtags, @mentions, $symbols, URLs, and media.
  - Extended Entities - Contains up to four native photos, or one video or animated GIF.
  - Places - Parent to ‘coordinates’ object.

Рисунок 3.3 - Основні об’єкти в Твіті

3.3 Отримання даних через Twitter Streaming API

Для отримання даних в реальному часі, ми використовуємо Twitter Streaming API.

В отриманих потоках використовується streaming (потоковий) HTTP-протокол для доставки даних через відкрите потокове з'єднання API. Замість того, щоб доставляти дані в пакетному режимі через повторювані запити клієнтським додатком, як можна очікувати від REST API, відкривається одне з'єднання між додатком і API, і нові результати відправляються через це саме з'єднання при кожному збігу. Це призводить до створення механізму

доставки з низькою затримкою, який може підтримувати дуже високу пропускну здатність.

Архітектура даної взаємодії зображена на рисунку 3.4.

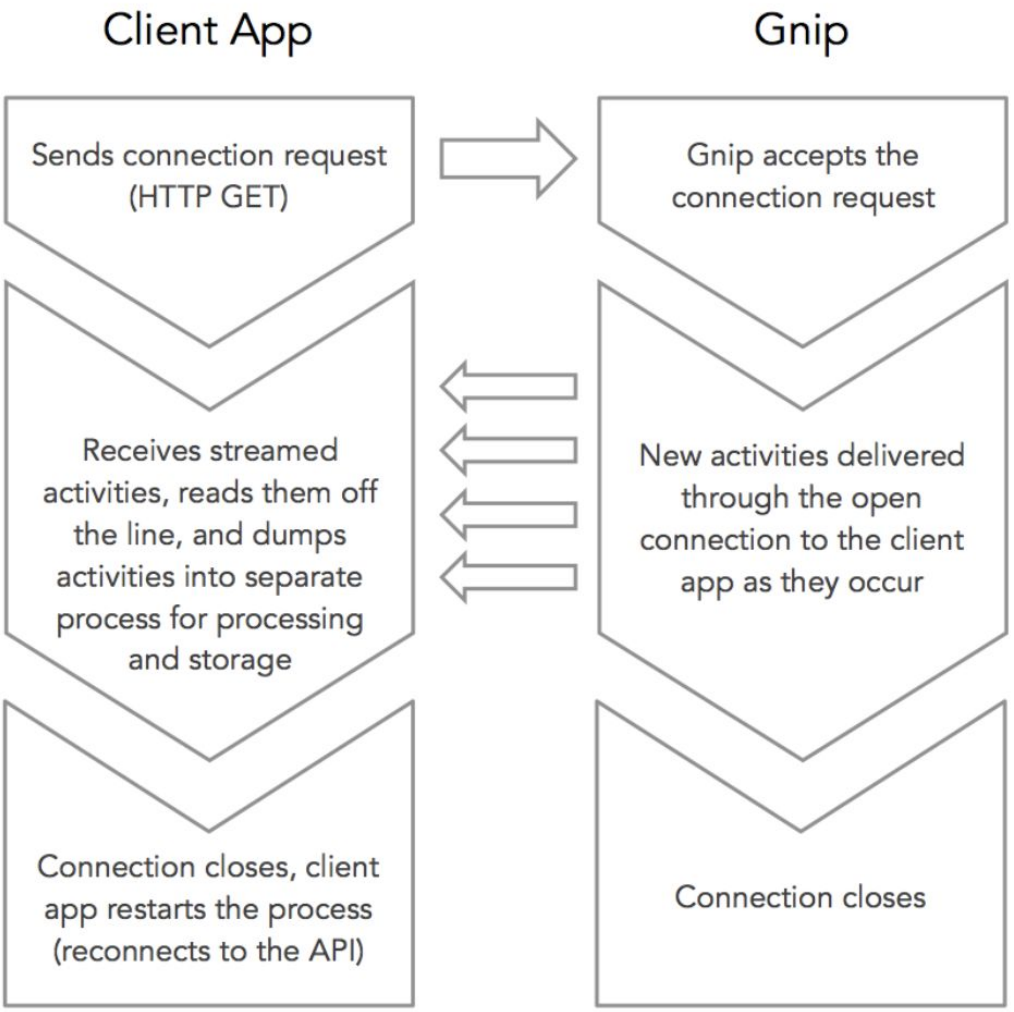

Рисунок 3.4 - Клієнт-сервер для отримання даних через streaming API

3.4 Процес обробки повідомлень

Для наглядного опису процесу обробки кожного повідомлення з Твітеру, опустимо нюанси архітектури сервісів, та розглянемо процес обробки одного повідомлення.

Першим етапом є мережевий запит на Web API Твітера з заданими фільтрами, в результаті якого ми отримуємо описані в розділі 3.2 об’єкти Твітів.

На другому етапі буде проводитись підготовка даних, а саме очищення від хештегів, згадувань, видалення стоп слів, токенізація, стемінг та лематизація слів, переведення тексту у векторне представлення. На третьому, останньому етапі, відбувається класифікація тексту. Цей процес візуалізовано на рисунку 3.5.

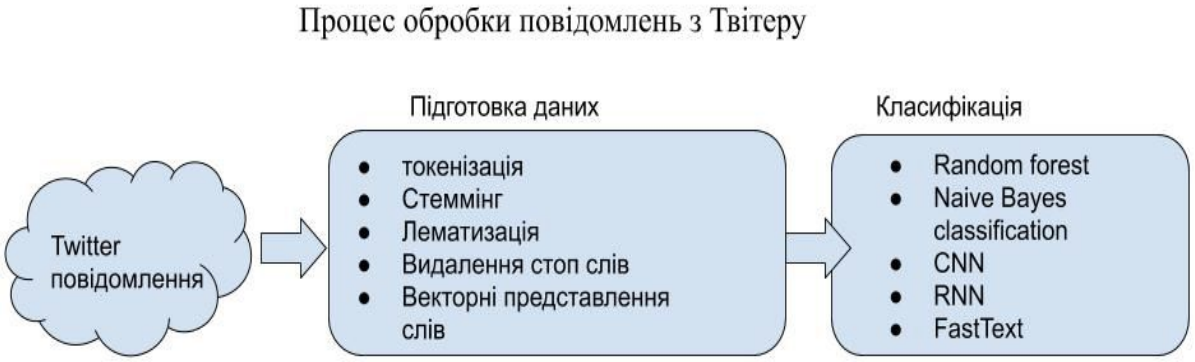

Рисунок 3.5 - Процес обробки кожного повідомлення отриманого з Твітер

На етапі підготовки даних, рядки тексту були розбиті на більш дрібні частини, або токени. Токенізація залежить від алгоритму, яким ми будемо користуватись, але в більшості полягає у розбитті тексту на більш малі “токени”, якими можуть бути речення, або слова. Подальша обробка зазвичай виконується після того, як фрагмент тексту був відповідним чином розбитий на токени.

Далі, текст нормалізується. Нормалізація включає в себе ряд пов'язаних завдань: перетворення всього тексту в один і той же регістр (верхній або нижній), видалення знаків пунктуації, перетворення чисел в їх еквівалентні слова і так далі. Нормалізацією ми ставимо всі слова в однакове становище і дозволяє процесам проходити рівномірно.

Наступним етапом є процес стемінгу, на якому ми видаляємо афіксів (суфіксів, префіксів, інфіксів, скорочень) з слова для отримання кореню слова.

Останнім етапом є лематизація, на якому ми перетворюємо слова в їх канонічні форми, засновані на лемі слова. Однак цей етап потрібен не для кожної моделі, наприклад моделі, які мають в основі нейронні мережі краще вчаться на словах без лематизації.

3.5 Мультикласова класифікація повідомлень

Як вже було сказано у розділі 3.2, для тренування моделі мультикласової класифікації, був взятий датасет з більш ніж 200 тисяч повідомлень, з 40 унікальними категоріями, список та кількість статей по категоріям можна побачити на рисунку 3.6.

На етапі підготовки даних, з тексту новин видалимо пунктуацію та спеціальні символи, переведемо текст в нижній реєстр, тому що пунктуація та розмір букв не впливає на зміст слів, з використанням бібліотеки nltk, видалимо стоп-слова, які не несуть змісту в тексті, зробимо лематизацію та токенізацію слів.

Наступним етапом, проведемо векторизацію слів, першим підходом спробуємо TF-IDF. Для цього використаємо Python бібліотеку sklearn.

Крім цього, в features вектор, можемо додати результат сентиментального аналізу та суб’єктивність тексту. Для цього використаємо бібліотеку TextBlob.

```
In [8]: print(data['category'].value_counts().plot(kind='bar', figsize=(12,7)))
Axes(0.125,0.125;0.775x0.755)
```

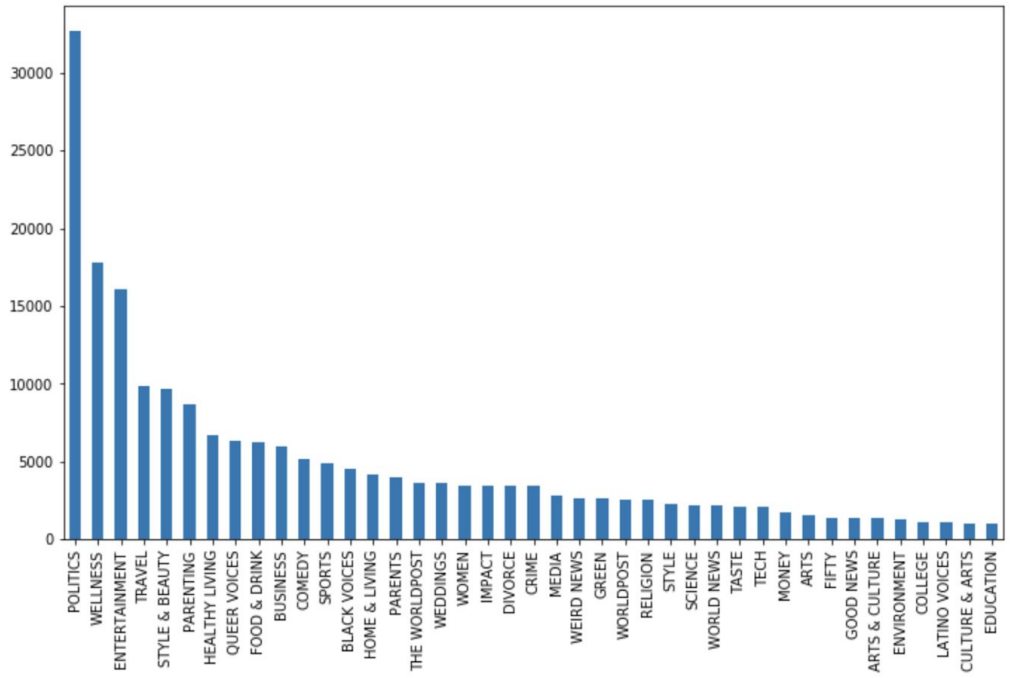

Рисунок 3.6 - Розподіл повідомлень за кожною категорією

Далі, навчимо класифікатор з використанням створених раніше feature векторів. Спробуємо спочатку моделі машинного навчання, як метод опорних векторів (SVM) та стохастичний градієнтний класифікатор.

Як результат, отримаємо таку точність моделей (рис. 3.7):

```
In [29]: classifier_svc = LinearSVC(C=1, class_weight='balanced', multi_class='ovr', random_state=3
         classifier_sgd = SGDClassifier(max_iter=300)

         sgd_scores = cross_val_score(classifier_sgd, train_vec, y_train, cv=cv, scoring='accuracy'
         svc_scores = cross_val_score(classifier_svc, train_vec, y_train, cv=cv, scoring='accuracy'

         print('sgd_score = {0:.3f}'.format(np.mean(sgd_scores)))
         print('svc_score = {0:.3f}'.format(np.mean(svc_scores)))

         sgd_score = 0.605
         svc_score = 0.582
```

Рисунок 3.7 - Точність класифікації за допомогою методу опорних векторів та стохастичного градієнтного класифікатора

Наступним етапом, використаємо модель глибокого навчання - LSTM, але модель глибокого навчання потребує векторне представлення слів, для цього використаємо модель doc2vec, передтреновану на статтях з Вікіпедії.

В результаті тренування моделі, отримали кращий результат 64% на 3 епосі. Процес тренування моделі зображений на рисунку 3.8.

```
Train on 145724 samples, validate on 14958 samples
Epoch 1/4
145724/145724 [==============================] - 317s 459us/st
Epoch 2/4
145724/145724 [==============================] - 322s 452us/st
Epoch 3/4
145724/145724 [==============================] - 320s 455us/st
Epoch 4/4
145724/145724 [==============================] - 333s 437us/st
Accuracy: 0.6417073125623442
```

Рисунок 3.8 - Точність класифікації за допомогою векторних представлень doc2vec та LSTM

Останнім методом спробуємо скористатися fasttext. Ядро FastText спирається на модель безперервної сумки слів (CBOW) для представлення слів і ієрархічний класифікатор для прискорення навчання. Для завдання класифікації використовується мультиноміальна логістична регресія, де вектор речення/документа відповідає features. Прогресс точності моделі в залежності від епохи зображений на рисунку 3.9.

В результаті отримали результат с точністю 0.77028 ~77% на десятій епосі з швидкістю навчання (learning rate) рівною 1.

Порівняння результатів роботи різних моделей зображені в таблиці 3.1.

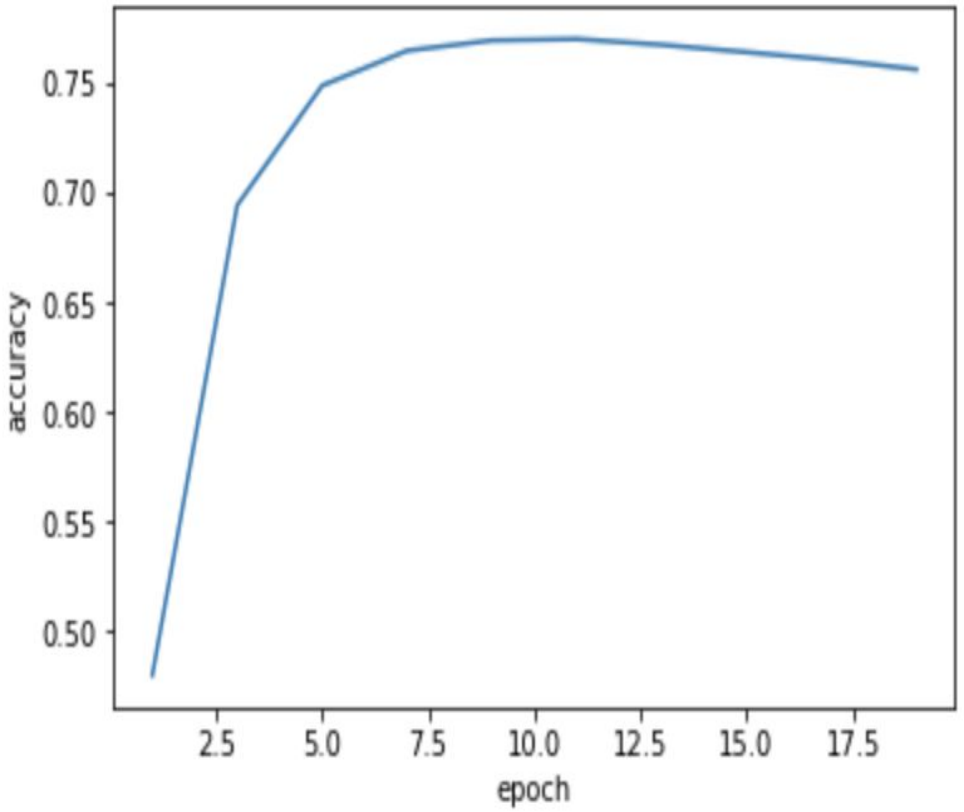

Рисунок 3.9 - Точність класифікації за допомогою FastText

Таблиця 3.1 - Порівняння роботи різних моделей

Назва моделі	Точність
SVC	58.2%
SGD	60%
LSTM	64.1%
FastText	77%

Отже, з протестованих моделей, для даної задачі оптимальною є FastText. Фінальний код для мультикласового класифікатору наведений у Додатку А.

3.6 Класифікація неякісного контенту

Неякісний, або небажаний контент являє собою схожий за змістом зі спамом контент. Таким є рекламні повідомлення, повідомлення, що не відносяться до теми якогось ланцюжка повідомлень.

Було вирішено винести цю категорію в окремий класифікатор, оскільки в класифікації більшої частини небажаних повідомлень можно досягти високих показників, через їх лексичну схожість, наповненість посиланнями та відмітками людей. Крім того, класифікувати неякісний контент допоможе інформація з профілю користувача Твітер, таку як кількість його підписників, кількість підписок, наявність фотографії, фотографії профілю, наявність вказаної геолокації, інформація про ретвіт і їх кількість.

В твітері є так звані сітки ботів, які часто мають схожі профілі з малою кількістю твітів та підписників, що допоможе класифікувати їх контент як небажаний. Однак це не є гарантією, тому ще буде проводитись аналіз тексту повідомлення за допомогою моделей машинного навчання.

Процес класифікації контенту на якісний чи неякісний зображений на рисунку 3.10.

Класифікація неякісного контенту складається з декількох етапів. На першому етапі підготовки даних текст твіту нормалізується, проводиться стемінг та лематизація.

Далі ми робимо з тексту embedding, тобто приводимо текст у векторне представлення, для подальшого тренування моделі і аналізу. Для побудови векторів було використано бібліотеки Word2Vec та fastText. Для обох бібліотек були взяти попередньо натреновані моделі на статтях з вікіпедії для англійської мови.

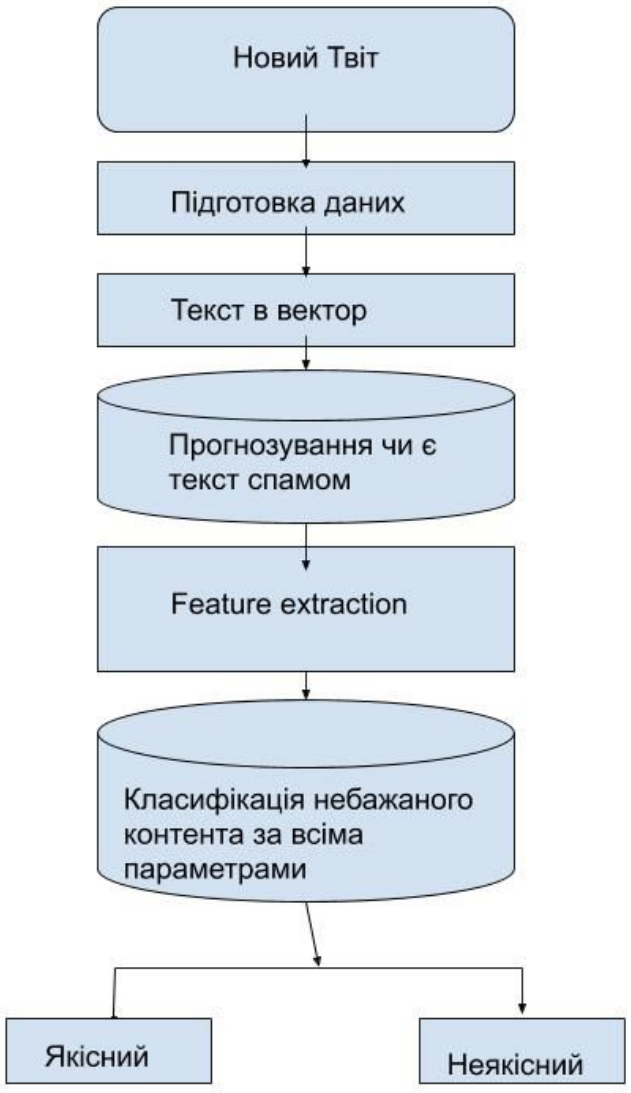

Рисунок 3.10 - Процес класифікації неякісного контенту

Проводимо подальше тренування наших моделей використовуючи датасети з неякісними повідомленнями, які було описано у розділі 3.2, щоб вони були здатні класифікувати текст і прогнозувати чи є контент якісним, або ні.

В результаті прогнозування класу тексту за допомогою тих самих алгоритмів, що були використані у розділі 3.5, з використанням Word2Vec та FastText. Кращий результат дав метод побудови опорних векторів з використанням Word2Vec моделі - 94%.

На наступному етапі ми робимо feature extraction (вилучення признаків) з об’єкту твіту, в результаті чого отримуємо таку структуру з даними, як зображено на рисунку 3.11.

	following	followers	actions	is_retweet	location	Type
6323	0.0	0.0	NaN	0.0	NaN	Spam
7170	894.0	5798.0	5072.0	0.0	United States	Quality
3086	183.0	397.0	1556.0	1.0	Snap: itsbrionna_xoxo	Quality
771	0.0	0.0	0.0	0.0	CVT	Spam
2326	0.0	0.0	NaN	1.0	Ontario, Canada	Spam

Рисунок 3.11 - Таблиця з features з твіту для класифікації якісного/неякісного контенту

Де following - є кількістю сторінок, на які користувач підписаний, followers - кількість його підписників, actions - кількість дій, is_retweet - чи є твіт ретвітом, location - надана користувачем геолокація, Type - результат роботи класифікатора з Word2Vec.

Наступним етапом використаємо RandomForest та NaiveBayesClassifier для прогнозування чи є твіт неякісним контентом. Після тренування моделі на 75% даних, протестужмо його на лишившихся 25%.

В результаті використання даних алгоритмів маємо такі результати: алгоритм випадкового лісу - 94%, Наївний баєсів класифікатор - 96%. Отже більше підходить баєсів класифікатор.

3.7 Розробка розподіленої системи з мікросервісною архітектурою

Для розробки системи було вибрано мікросервісну архітектуру, що дало такі переваги системи:

- Легко обслуговується і перевіряється
- Слабо пов'язана
- Самостійно і незалежно розгортається
- Легко розширюється
- Кожний сервіс незалежний і може бути розроблений на різних технологіях

Під кожну задачу було створено окремий сервіс, в результаті чого маємо таку архітектуру, зображену на рисунку 3.12.

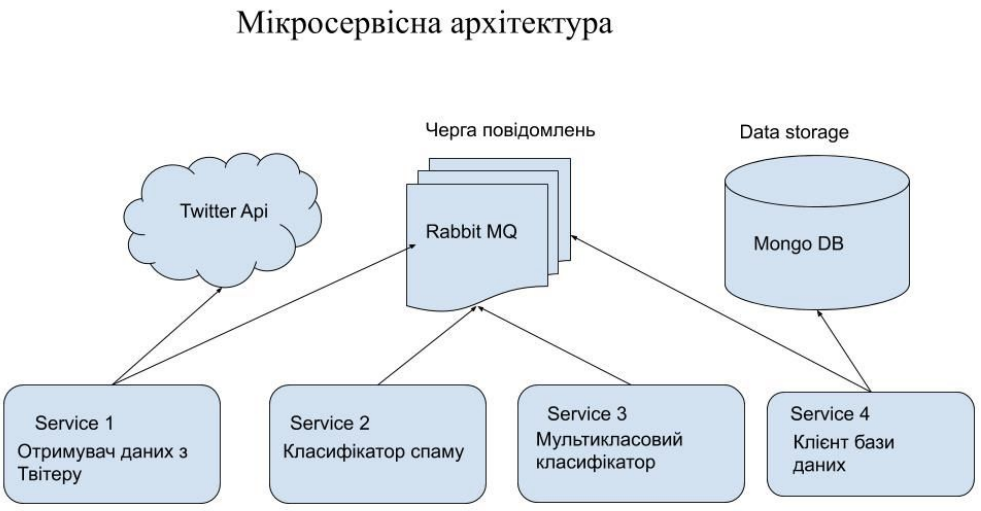

Рисунок 3.12 - Архітектура системи, опис взаємодії сервісів

Сервіси взаємодіють між собою за допомогою асинхронної черги повідомлень, таким чином кожний сервіс виконує свою роботу, не затримуючі інші сервіси. Завдяки цьому можна незалежно масштабувати кожний точку системи, яка цього потребує, не заважаючи іншим вузлам.

Входом в систему, є Service 1, який авторізується та отримує дані з Твітеру за заданими фільтрами і записує їх в чергу на обробку. Наступним

етапом обробки є класифікація неякісного контенту - Service 2, після нього йде мультикласовий класифікатор (Service 3) і останнім з черги бере дані Service 4, який зберігає оброблені дані в нереляційній базі даних - MongoDB для подальшого використання клієнтами. Ця послідовність проілюстрована на рисунку 3.13.

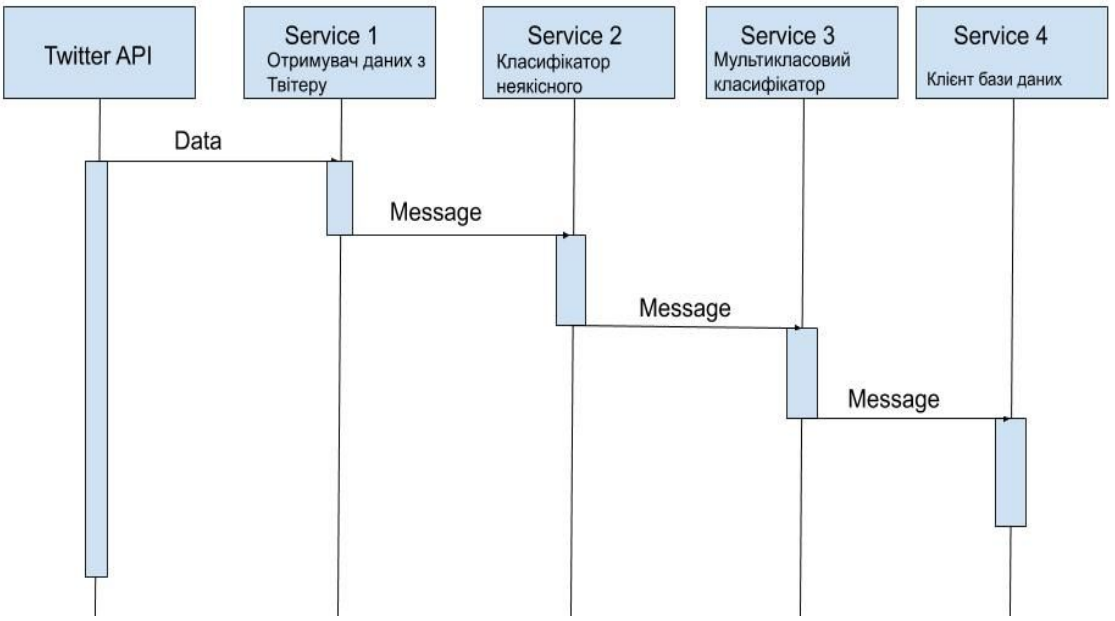

Рисунок 3.13 - UML діаграма послідовності процесу обробки кожного Твіту

Зміст docker-compose файла для всіх сервісів можна подивитись у Додатку Б, а структуру директорій і файлів сервісів у Додатку В.

3.8 Приклад моделювання та роботи системи на реальних даних

Для роботи на реальних даних було обрано фільтр хештегом “#news” та фільтр за англійською мовою, оскільки наша модель була тренована для англійської мови, а категорії були обрані для новин.

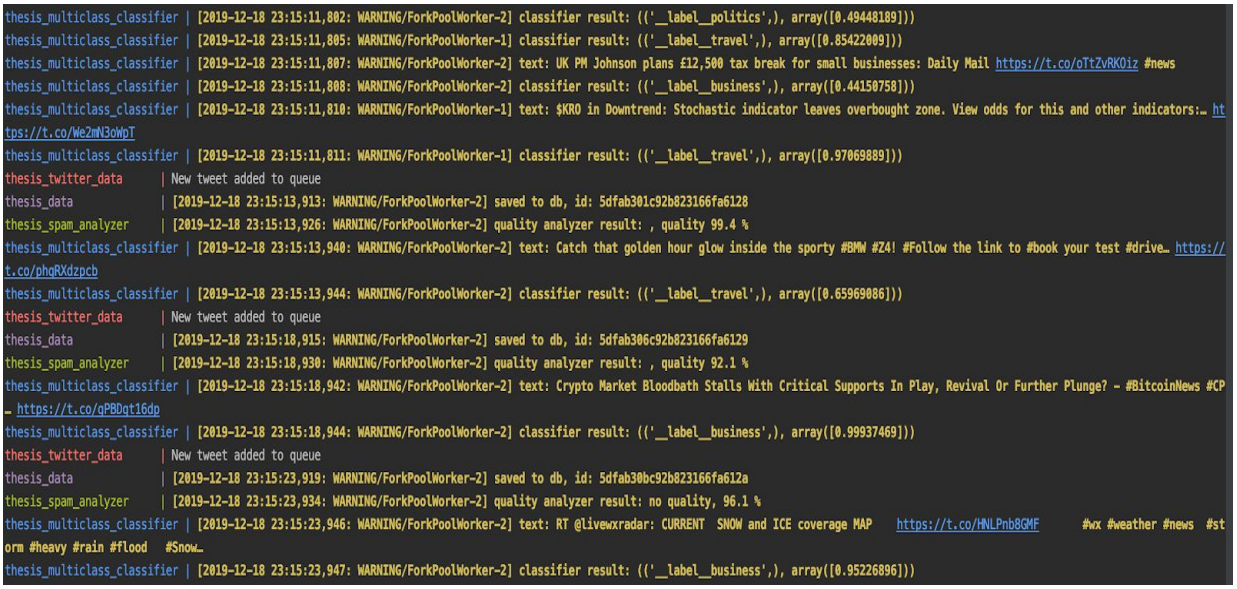

Рисунок 3.14 - Лог при роботі програми в дебаг-режимі

На рисунку 3.14 можна побачити приклад виводу програми при роботі в дебаг-режимі за декілька секунд. По порядку можна бачити логи кожного сервісу: спочатку “thesis_twitter_data” отримує дані з API Твітеру, та додає в чергу на обрбку наступними сервісами. Після нього йде сервіс “thesis_spam_analyzer”, що класифікує чи є контент якісним, або ні, виводить результат та ймовірність свого прогнозу. За ним Йде “thesis_multiclass_classifier” що класифікує вміст повідомлення за різними класами та виводить результат з ймовірністю. Останнім є сервіс “thesis_data”, що зберігає дані у постійне сховище, в розробленій системі таким виступає нереляційна база даних MongoDB.

Розглянемо декілька прикладів детальніше. Наприклад текст “#Pakistan Merkel rules out retaliation after U.S. sanctions Russian gas pipeline” (“Пакистан Меркель виключає помсту після санкцій російського газопроводу”) був оцінений як якісний з ймовірністю 97.5 %, і був класифікований тегом “politics” (політика) з ймовірністю 49%, що є правильною класифікацією.

За ним текст “Crypto Market Bloodbath Stalls With Critical Supports In Play, Revival Or Further Plunge?” (“Криптовалютні сховища крові з критичною підтримкою у грі, відродженні чи подальшому зануренні?”) був класифікований як “business” (“бізнес”) з ймовірністю 99%, що також є вірним.

Однак текст “$KRO in Downtrend: Stochastic indicator leaves overbought zone. View odds for this and other indicators” (“$ KRO у Даунтренді: Стохастичний індикатор залишає зону перекуплення. Перегляньте шанси на цей та інші показники”) був класифікований як “travel” (“подорожі”), з ймовірністю 97%, що є помилкою.

Щодо продуктивності системи, на запиті до Твітеру з найбільш широкими фільтрами, система була спроможна обробляти близько 200 повідомлень в секунду, як можна бачити на рисунку 3.15.

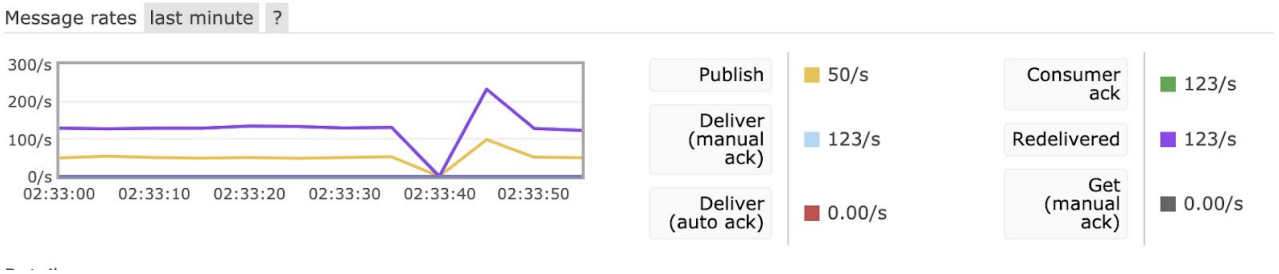

Рисунок 3.15 - Пропускна спроможність системи на реальних даних

Однак це експеримент лише для одного запиту. Оскільки система має бути здатна обробляти одночасно декілька пошукових запитів, та допустивши що маємо дуже широкий мережевий канал, та безлімітний доступ до API Твітеру. протестуємо кожний сервіс мануально на великих об’ємах.

Для мультикласового класифікатору навантаження в 500 повідомлень в секунду виявилось вже завеликим, сервіс здатний обробляти близько 400 в секунду. Графіки можна побачити на рисунку 3.16.

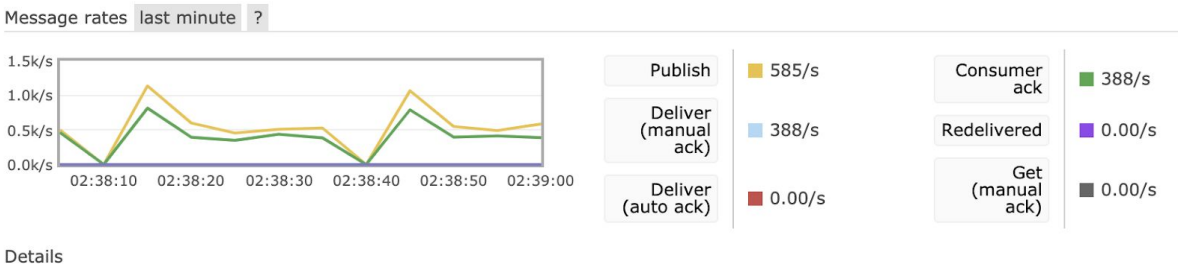

Рисунок 3.16 - Пропускна здатність сервісу класифікації повідомлень

Сервіс для фільтрації якісного/неякісного контенту дає трохи гірші результати и здатний обробляти близко 250 повідомлень в секунду.

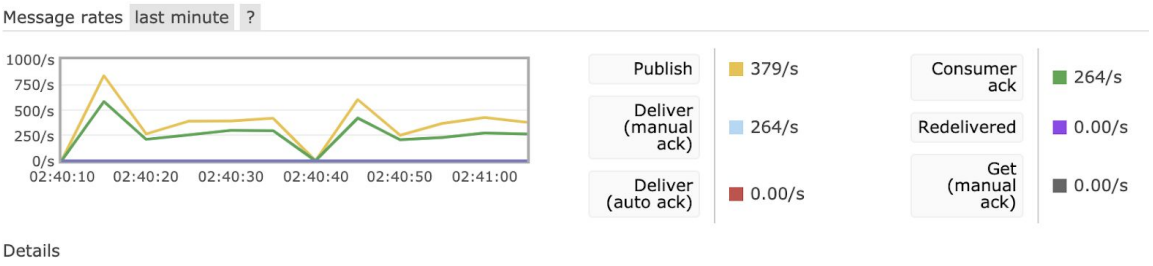

Рисунок 3.18 Ліміт пропускної здатності для сервісу класифікації якісного/неякісного контенту

Однак завдяки використанню розподіленої системи з мікросервісною архітектурою проблеми з пропускною спроможностю сервісів можна легко вирішити додаванням ще одного воркера в систему.

Висновок до розділу 3

В результаті практичної частини було проаналізовано та порівняно роботу алгоритмів машинного навчання та глибокого навчання, застосовано нейронні мережі для побудови векторного представлення повідомлень з

соціальних мереж. В результаті чого було знайдено оптимальні інструменти під задачу мультикласової класифікації текстів з Твітера.

Окрім того було розроблено розподілену систему з мікросервісною архітектурою для отримання даних з Твітера, проходження декількох стадій обробки цих даних та збереження результату у базу даних.

На виході маємо систему, яка з точністю ~96% класифікує неякісний контент, та з точністю 77% класифікує до якої категорії належить повідомлення.

ВИСНОВКИ

В процесі виконання даної роботи були детально вивчені теоретичні основи та сучасні методі аналізу та обробки природних мов за допомогою методів машинного навчання.

Для роботи з даними та навчання моделей, був зібраний та розмічений датасет за популярними хештегами з Твітеру для виявлення якісного/неякісного контенту, а також використаний датасет з вже розміченими повідомленнями за різними класами, які і було використано.

В процесі було виконано програмну реалізацію різних алгоритмів та методів класифікації природного тексту, а саме метод опорних векторів, метод стохастичного градієнта, мережі довгої короткострокової пам’яті, модель безперервної сумки слів з логістичною регресією, порівняно їх результат на реальних розмічених даних з Твітеру. Обрано оптимальний для мультикласової класифікації та окремий для класифікації якісного/неякісного контенту - це модель безперервної сумки слів з логістичною регресією в реалізації fasttext, що дало результат близько 77% на тестових даних.

Для класифікації неякісного контенту було використано метод побудови векторів слів Word2Vec та метод опорних векторів для прогнозування чи є текст спамом, а після цього Наївний баєсів класифікатор, що дало точність 96%.

Як результат, була побудована розподілена система з мікросервісною архітектурою, яка здатна в режимі реального часу отримувати повідомлення з Твітеру за заданими фільтрами та робити обробку кожного в декілька етапів, а саме класифікацію в даному випадку, але можливе розширення архітектури шляхом додавання сервісів.

Отриману систему можна використовувати як в особистих цілей для слідкування за новинним фоном якоїсь події або хештегом, та отримання

інформації в реальному часі, класифіковану за різними класами та з відфільтрованим неякісним контентом, так і в якості окремого сервісу для класифікації, результат роботи якого може бути використаний в іншій системі.

Зазначена система була випробувана і протестована на великих об’ємах реальних даних з Твітеру в режимі реального часу.

Розроблений метод має перспективи для застосування у класифікації та фільтрації контенту з соціальних мереж.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit / Steven Bird, Ewan Klein, and Edward Loper. - Режим доступу: https://www.nltk.org/book/

2. Manning Christopher D. Foundations of statistical natural language processing / Christopher D. Manning, Hinrich Schutze. - The MIT Press Cambridge, Massachusetts London, England, 2000. - 704 p.

3. Jurafsky Daniel. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition / Daniel Jurafsky, James H. Martin. - 3rd edition. - Prentice Hall, 2019. - 621 p.

4. Miller Brad N. Problem Solving with Algorithms and Data Structures using Python / Brad N. Miller, David L. Ranum. - Franklin, Beedle & Associates, 2016. - 240 p.

5. CSRankings: Computer Science Rankings. - Режим доступу: http://csrankings.org/

6. Rishabh Misra (персональний блог). - Режим доступу: https://rishabhmisra.github.io/publications/

7. Wikipedia (вільна енциклопедія). - Режим доступу: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

8. Habr. - Режим доступу: https://habr.com/

9. Micreservices io. - Режим доступу: https://microservices.io/

10. Programming Collective Intelligence: Building Smart Web 2.0 Applications - Toby Segaran

11. Machine Learning: An Algorithmic Perspective - Stephen Marsland

12. Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale (Jennifer Davis, Ryn Daniels)

13. Two Scoops of Django: Best Practices for the Django Web Framework (Daniel Roy Greenfeld and Audrey Roy Greenfeld)

14. Modern Operating Systems Andrew Tanenbaum

15. Learning Python, 5th Edition Fifth Edition (Mark Lutz)

16. Документація Твітер API - https://developer.twitter.com/

Додаток А

Лістинг коду для тренування FastText моделі

```
import string
import re
import json

import nltk
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
from nltk.corpus import stopwords

import fasttext

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')

en_stopwords = set(stopwords.words('english'))
wordnet_lemmatizer = WordNetLemmatizer()

def filter_token(token):
    return (token not in en_stopwords
            and token not in string.punctuation
            and not re.findall(r'^\W+$', token)
           )
```

```
def clean_text(text):
    # make text lower
    text = text.lower()

    #remove hashtags
    text = re.sub("(@[A-Za-z0-9]+)|(#[A-Za-z0-9]+)", " ", text)
    # tokenize words
    text = word_tokenize(text)

    return ' '.join(text)

# optionally use pandas for funcs below
def read_from_file(filename='news_category_dataset.json'):
    with open(filename, 'r') as f:
        data = f.read()
    data = data.split('\n')
    data = [json.loads(each) for each in data if each]
    return data


def preprocess_data(data):
    data = data.copy()
    for each in tqdm.tqdm_notebook(data):
        category = each['category']
        if category == "THE WORLDPOST":
            each['category'] = 'WORLDPOST'
        each['text'] =  clean_text(each['headline'] + each['short_description'])
    return data
```

```
def format_to_fasttext(data):
    return ['__label__{} {}'.format(each['category'].lower(), each['text']) for each in
data]


def write_train_test_files(data: list) -> None:
    train_set_length = int((len(data) * 0.75))
    with open('categories_train.txt', 'w') as train_f:
        train_f.write('\n'.join(data[:train_set_length]))
    with open('categories_test.txt', 'w') as test_f:
        test_f.write('\n'.join(data[train_set_length:]))



def train_model():
    model = fasttext.train_supervised(input='categories_train.txt', epoch=10, lr=1.0)
    return model

def test_model(model):
    return model.test("categories_test.txt")
```

Додаток Б

Лістинг змісту файлу docker-compose для системи

```
version: '3'

services:
  twitter_subscriber:
    build: ./twitter
    env_file:
      - twitter/.env
      - .env/rabbitmq.env
    container_name: 'thesis_twitter_data'
    depends_on:
      - rabbitmq
    tty: true
    stdin_open: true
    command: python run.py

  data:
    build: ./data
    env_file:
      - ./data/.env
      - .env/rabbitmq.env
    container_name: 'thesis_data'
    depends_on:
      - rabbitmq
      - mongodb
    tty: true
```

```
  stdin_open: true

spam_analyzer:
  build: ./spam_analyzer
  env_file:
    - ./spam_analyzer/.env
    - .env/rabbitmq.env
  container_name: 'thesis_spam_analyzer'
  depends_on:
    - rabbitmq
    - mongodb
  tty: true
  stdin_open: true

multiclass_classifier:
  build: ./multiclass_classifier
  env_file:
    - ./multiclass_classifier/.env
    - .env/rabbitmq.env
  container_name: 'thesis_multiclass_classifier'
  depends_on:
    - rabbitmq
    - mongodb
  tty: true
  stdin_open: true

mongodb:
  image: mongo:4.2.1-bionic
```

```
  container_name: 'thesis_mongo'
  env_file:
    - .env/mongodb.env

rabbitmq:
  image: rabbitmq:3.8-management-alpine
  ports:
    - 4369:4369
    - 5671:5671
    - 5672:5672
    - 15672:15672
    - 25672:25672
  env_file:
    - .env/rabbitmq.env
  container_name: 'thesis_rabbitmq'
```

Додаток В

Лістинг змісту директорій і файлів сервісів

```
├── .env
│   ├── mongodb.env
│   ├── rabbitmq.env
├── data
│   ├── Dockerfile
│   ├── data
│   │   ├── __init__.py
│   │   ├── celery_app.py
│   │   ├── celeryconfig.py
│   │   ├── log.py
│   │   └── tasks.py
│   ├── requirements.txt
│   └── run.py
├── multiclass_classifier
│   ├── Dockerfile
│   ├── model
│   │   └── model.bin
│   ├── multiclass_classifier
│   │   ├── __init__.py
│   │   ├── celery_app.py
│   │   ├── celeryconfig.py
│   │   ├── classifier.py
│   │   ├── log.py
│   │   ├── model
│   │   └── tasks.py
│   ├── requirements.txt
```

```
│   └── run.py
├── spam_analyzer
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── run.py
│   └── spam_analyzer
│       ├── __init__.py
│       ├── celery_app.py
│       ├── celeryconfig.py
│       ├── log.py
│       └── tasks.py
├── twitter
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── run.py
│   └── twitter
│       ├── __init__.py
│       ├── app.py
│       ├── celery_app.py
│       ├── celeryconfig.py
│       └── log.py
├── docker-compose.yaml
```