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:

СТАТИСТИЧНА КЛАСИФІКАЦІЯ ТЕКСТУ

Виконав студент 4-го курсу

Куценький Олексій Олександрович

Науковий керівник:

доцент

Лівінська Ганна Володимирівна

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

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

Київ – 2023

ЗМІСТ

АНОТАЦІЇ

Дипломна робота складається зі вступу, 5 розділів, висновків та списку використаних джерел (10 найменувань). Загальний обсяг роботи становить 45 сторінок, основний текст роботи викладено на 34 сторінках.

Ключові слова: обробки природної мови, попередня обробка тексту, векторизація тексту, вкладання слів, дерева рішень, випадковий ліс, кероване навчання, класифікація.

Key words: natural language processing, text preprocessing, text vectorization, word embedding, decision trees, random forest, supervised learning, classification.

ВСТУП

На сьогоднішній день інтелектуальний аналіз тексту (англ. Text Mining, далі ІАТ) набрав популярності із розвитком штучного інтелекту. Мета ІАТ полягає у отриманні інформації з колекцій текстових документів, ґрунтуючись на застосуванні ефективних, у практичному плані, методів машинного навчання та обробки природної мови (англ. Natural Language Processing, далі NLP).

У цій роботі буде розглянуто методи NLP. В його основі лежить припущення, що зміст тексту може бути визначено за найуживанішими словами. Основним завданням даного підходу є визначення кількості повторень конкретного слова та словосполучень в тексті. Основна проблема, з якою стикаються статистичні підходи, полягає в розгляді тексту як набору слів без смислового зв'язку.

Метою роботи є створення програмного забезпечення для класифікації тексту.

Методи розробки: мова Python та її бібліотеки: matplotlib, pandas, scikit-learn та інші.

РОЗДІЛ 1. ПОПЕРЕДНЯ ОБРОБКА ТЕКСТУ

У будь-якій задачі машинного навчання очищення та попередня обробка даних настільки ж важливі, як і сама побудова моделі. У випадку текстових даних, вони є однією з найбільш неструктурованих форм доступних даних, тому правильна їх обробка буде мати великий вплив на точність моделі.

Існує багато способів змінити вхідні данні [1], але не кожен з них покращить модель. Потрібно щітко розуміти, які методи використовувати залежно від ситуації.

Розглянемо методи попередньої обробки тексту:

- **Приведення до нижнього регістру**

Малий та верхній регістри трактуються машиною по різному. Тобто, слова “Книга” і “книга” будуть мати різне значення. Тому краще привести усі слова до нижнього регістру.

- **Розгорнення скорочень**

В різних мовах виникають скорочення, наприклад “I`m”, що не буде рівне словосполученню “I am”, тому при обробці варто розгорнути подібні випадки.

- **Видалення розділових знаків**

Розділові знаки та спеціальні символи не принесуть великої користі для класифікації тексту, тому прибрати їх є оптимальним рішенням.

- **Видалення комбінацій чисел і слів**

Іноді трапляється, що слова та цифри поєднуються в тексті, що створює проблеми для розуміння машин. Отже, нам потрібно видалити слова та цифри, які об’єднані, наприклад, “користувач5”. Цей тип слів важко обробити, тому краще видалити їх. Можна спробувати ввести деякі додаткові правила декомпозиції подібних сполучень, наприклад:

“користувач5” -> “користувач номер 5” (варто вирішитись залишити число як є чи перевести у письмову форму).

- **Видалення стоп-слів**

Стоп-слова — це слова, які не додають великого значення реченню і повторюються частіше всього. Їх можна видалити, оскільки насправді вони не допомагають розрізняти два речення. Також до стоп-слів можна додати і виключення, які теж видаляться. Наприклад посилання, HTML теги та інші конкретні слова.

- **Перефразування тексту**

Можна провести зміну деякого тексту або шаблону на певний рядок, що спрощує ідентифікацію, наприклад, замінити повну назву пошти на словосполучення “електронна адреса”. Хорошою практикою є заміна непідходящих слів або словосполучень, а не їх видалення, оскільки вхідні дані обмежені і їх потрібно використовувати по максимуму.

- **Стемінг і лематизація**

Стемінг (англ. *stemming*) — це процес скорочення слова до основи шляхом відкидання допоміжних частин, таких як закінчення чи суфікс. Техніка скорочення до основи не настільки ефективна, і в більшості випадків створює небажані слова. Лематизація, на відміну від стемінгу, зводить слова до слова, наявного в мові. Лематизація є кращою за стемінг, оскільки лематизація виконує морфологічний аналіз слів.

РОЗДІЛ 2. ПРОСТІ МЕТОДИ ВЕКТОРИЗАЦІЇ ТЕКСТУ

2.1 Торба слів

Найпростіша з усіх існуючих технік. Включає в себе три операції:

- **Токенізація**

Речення представляється у вигляді списку його слів, і це робиться для всіх вхідних речень.

- **Створення словника**

З усіх отриманих токенізованих слів для створення словника вибираються лише унікальні слова, а потім сортуються в алфавітному порядку.

- **Створення вектора**

Для вхідних даних створюється розріджена матриця. Її рядки відповідатимуть конкретному реченню, а стовпці - окремо взятому слову зі словника. При цьому числові елементи в кожному рядку будуть вказувати на те, скільки разів те чи інше слово зустрічається в даному реченні.

Переваги:

1. Простий та інтуїтивно зрозумілий
2. Мовний агностик, тобто його можна застосувати до будь-якої мови з деякою попередньою обробкою

Недоліки:

1. Може призвести до високовимірного та розрідженого векторного представлення ознак, що може бути дорогим з обчислювальної точки зору та вимагати багато пам’яті.

2. Вразливий до шумів у даних, таких як орфографічні та граматичні помилки, які можуть негативно вплинути на точність моделі.
3. Ігнорує порядок слів у тексті, що призводить до втрати контекстної інформації.
4. Не здатний вловити семантичне значення слів.

```python
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

corpus = ['Today is a great day',
          'Great weather outside today',
          'It is hot today']

cv = CountVectorizer()
X = cv.fit_transform(corpus)

pd.DataFrame(X.toarray(), columns=cv.get_feature_names_out())
```

	day	great	hot	is	it	outside	today	weather
0	1	1	0	1	0	0	1	0
1	0	1	0	0	0	1	1	1
2	0	0	1	1	1	0	1	0

Рисунок 2.1 - Приклад використання торби слів на мові Python

2.2 TF-IDF

TF-IDF або частота термінів – інверсна частота документа (тексту) – це числова статистика, яка має на меті відобразити, наскільки важливе слово є для документа. У “Торбі слів” векторизація була пов’язана лише з частотою слів з словника у певному документі. В результаті прийменники та сполучники, які не мають великого значення для значення, набувають такого ж значення, як прикметники. TF-IDF допомагає нам подолати цю проблему. Слова, які повторюються занадто часто, не переважають менш частих, але важливих слів.

Формули:

$$TF = \frac{\text{кількість разів, коли слово зустрічається в тексті}}{\text{загальна кількість слів в тексті}}$$

$$IDF = \log \frac{\text{кількість документів, у яких зустрічається слово}}{\text{кількість документів}}$$

Кінцевий результат, що заноситься у матрицю рахується за формулою:

$$TF - IDF$$

Логарифм береться для послаблення впливу IDF на остаточне значення.

Переваги:

1. Враховує важливість слова: TF-IDF враховує важливість слова в документі, зважуючи його на основі його частоти в документі та рідкості в корпусі.
2. Зменшує шум: слова, які часто з’являються в усіх документах, мають нижчий бал і, отже, з меншою ймовірністю впливатимуть на прогнози моделі. Це допомагає зменшити шум і підвищити точність моделі.

Недоліки:

1. Не вловлює порядок слів
2. Розрідженість
3. Вичерпаний словниковий запас

```python
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = ['Today is a great day',
          'Great weather outside today',
          'It is hot today']

tfidf = TfidfVectorizer()
X = tfidf.fit_transform(corpus)

pd.DataFrame(X.toarray(), columns=tfidf.get_feature_names_out())
```

	day	great	hot	is	it	outside	today	weather
0	0.631745	0.480458	0.000000	0.480458	0.000000	0.000000	0.373119	0.000000
1	0.000000	0.444514	0.000000	0.000000	0.000000	0.584483	0.345205	0.584483
2	0.000000	0.000000	0.584483	0.444514	0.584483	0.000000	0.345205	0.000000

Рисунок 2.2 - Приклад використання TF-IDF на мові Python

2.3 N-грами

У NLP n-грами - це безперервна послідовність n елементів, як правило, слів або символів, які зустрічаються у фрагменті тексту. Модель n-грам — це статистична модель мови, яка передбачає ймовірність появи наступного слова в послідовності на основі n-грам попередніх слів. Наприклад, у реченні «Сьогодні на вулиці чудова погода» кілька прикладів n-грам:

Уніграми (n=1): сьогодні, на, вулиці, чудова, погода

Біграми (n=2): сьогодні на, на вулиці, вулиці чудова, чудова погода

Триграми (n=3): сьогодні на вулиці, на вулиці чудова, …

Вибір значення n залежить від конкретної задачі та розміру текстового корпусу. N-грами широко використовуються в НЛП для таких завдань, як класифікація текстів, мовне моделювання та машинний переклад.

До торби слів та TF-IDF можна додати генерацію n-грам. Їх використання підвищить точність класифікації.

РОЗДІЛ 3. ВКЛАДАННЯ СЛІВ

3.1 Основні поняття

Вкладання слів (англ. *word embedding*) — це загальна назва низки методик мовного моделювання та навчання ознак в обробці природної мови. Гарною аналогією було б те, як представлені кольори у RGB (англ. Red, Green, Blue) моделі.

Недоліки методів описаних у 2 розділі:

- Розмір векторів залежить від розміру нашого словникового запасу (який може бути величезним). Це витрачає пам’ять і експоненціально збільшує складність алгоритму, що призводить до "прокляття розмірності" (англ. curse of dimensionality).
- Передавальне навчання до моделі з використанням іншого словника того ж розміру, додавання/вилучення слів зі словника, буде майже неможливим, оскільки це вимагатиме повторного навчання всієї моделі.
- Немає співвідношень між словами, які мають схоже значення чи використання.

Вкладання слів покриває всі ці недоліки. Результат вкладання слів – це векторне представлення слів, що зберігають семантичний зв'язок між ними. Наприклад, для слова “кішка” одним з найближчих буде слово “собака”. Однак векторне зображення слова “погода” буде досить сильно відрізнятися від “кішки”. Ця схожість обумовлена частотою зустрічі цих двох слів (тобто [кішка, собака] або [кішка, погода]) в одному контексті.

3.2 Word2vec

Одну з найпопулярніших методик представлення тесту запропонував у 2013 році чеський аспірант Томаш Миколов і назвав її word2vec [2]. Запропонована модель дуже проста – необхідно передбачати ймовірність слова по його оточенню (контексту). Тобто необхідно підібрати такі вектори слів, щоб ймовірність, що привласнюється слову в даному контексті, була близька до ймовірності зустріти це слово в цьому оточенні в реальному тексті написаному людиною.

Робота моделі здійснюється наступним чином: word2vec приймає великий корпус документів у якості вхідних даних і зіставляє кожному слову вектор, видаючи координати слів на виході. Спочатку модель генерує словник корпусу, а потім обчислює векторне подання слів, «навчаючись» на вхідних текстах. Векторне подання ґрунтується на контекстній близькості: слова, що зустрічаються в тексті поруч з однаковими словами (а отже, мають схожий зміст), будуть мати близькі вектори [3][4].

У word2vec реалізовані два основних алгоритми навчання: “**Безперервна торба слів**” (англ. *Continuous Bag of Words*) і “**Пропуск-грама**” (англ. *Skip-gram*).

У моделі “Безперервна торба слів” розподілене представлення контексту (або навколишніх слів) об’єднуються, щоб передбачити слово в середині. У той час як у моделі “Пропуск-грама” розподілене представлення вхідного слова використовується для передбачення контексту.

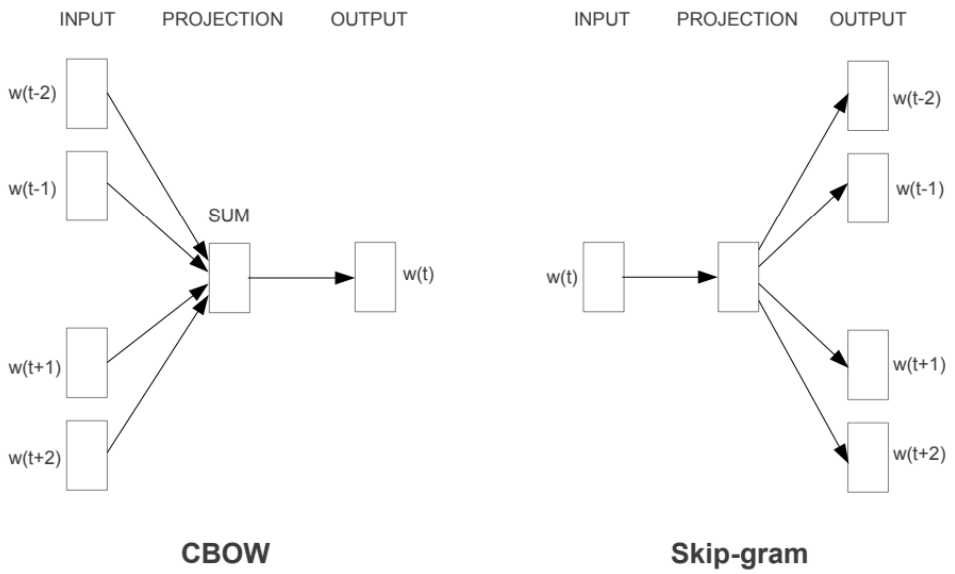


Рисунок 3.1

3.2.1 Безперервна торба слів

Цільова функція моделі “Безперервної торби слів” полягає в тому, щоб передбачити середнє слово як ціль, коли дається N/2 попередніх слів та N/2 наступних. Наприклад, “Сьогодні на вулиці чудова погода”. У цьому випадку “вулиці” є середнім або цільовим словом, яке потрібно передбачити, а наші контекстні слова мають бути “сьогодні”, “на”, “чудова”, “погода”.

Під час навчання вхідний шар передає унітарно закодовані (англ. *one-hot encoding*) контекстних слів у прихований шар(и) (рівень проекції). У проекційному шарі ці вектори контекстних слів просто усереднюються як стиснене представлення. Вихідний рівень приймає це представлення та створює векторне представлення для цільового слова. Модель позначається як “Безперервна торба слів”, оскільки на відміну від стандартної моделі, вона використовує безперервне розподілене представлення контексту.

Кількість прихованих шарів у проекційному шарі та кількість нейронів у кожному шарі є параметрами, які можна налаштувати під час навчання для покращення ефективності моделі.

Векторні представлення “навчаються” шляхом мінімізації функції втрат, яка вимірює розбіжність між прогнозованими та фактичними цільовими словами. Вагові коефіцієнти мережі оновлюються за допомогою зворотного поширення та стохастичного градієнтного спуску.

Після того, як модель навчена, векторні представлення можна використовувати для представлення слів у безперервному векторному просторі, який фіксує їхні семантичні зв’язки (див. діаграма 1).

3.2.2 Пропуск-грама

Ця модель має ціль, протилежну моделі “Безперервна торба слів”. Враховуючи поточне слово “вулиці” як вхід, вона передбачає найближчі контекстні слова в певному діапазоні як попередні так і наступні: “сьогодні”, “на”, “чудова”, “погода”. Збільшення діапазону покращує якість результуючих векторів слів, але також збільшує складність обчислень.

Під час навчання вхідний рівень подає унітарно закодований вектор для цільового слова на прихований(і) шар(и) (проекційний рівень). Рівень проекції потім перетворює вхідний вектор у стиснене представлення (просте середнє значення), яке фіксує контекст цільового слова. Рівень виводу приймає це стиснуте представлення та виробляє унітарно закодовані вектори для контекстних слів.

Кількість прихованих шарів і кількість нейронів у кожному шарі є параметрами, які можна налаштувати під час навчання для покращення ефективності моделі.

Ваги мережі “навчаються” шляхом мінімізації функції втрат, яка вимірює розбіжність між прогнозованими та фактичними контекстними словами. Вагові коефіцієнти оновлюються за допомогою зворотного поширення та стохастичного градієнтного спуску.

Після того, як модель навчена, векторні представлення можна використовувати для представлення слів у безперервному векторному просторі, який фіксує їхні семантичні зв’язки (див. діаграма 1).

Переваги:

1. Створені вектори мають обмежені розміри.
2. Розрідженість буде зменшено.
3. Семантичне значення між векторами зберігається.

Недоліки:

1. Для ефективного навчання моделі Word2Vec потрібні великі обсяги даних, які в деяких випадках можуть бути недоступними.
2. Моделі Word2Vec можуть не працювати так добре на предметно-спеціальних або спеціалізованих словниках, оскільки вони зазвичай навчаються на великих і загальних корпусах.
3. Відсутність інтерпретованості.
4. Word2Vec обмежений вибраним розміром контексту, а це означає, що він може не мати змоги охопити зв’язки між словами, які розділені великою кількістю слів.

“Безперервна торба слів”: тренується в кілька разів швидше, ніж скіп-грам, трохи краща точність для частих слів.

“Пропуск-грама”: добре працює з невеликою кількістю навчальних даних, добре представляє навіть рідкісні слова чи фрази.

3.2.3 Негативна проба

Ідею word2vec можна інтерпретувати як максимізацію скалярного добутку між векторами слів, які з'являються поряд одне з одним в тексті, та у мінімізації добутку векторів слів, які не з'являються в одному контексті. При роботі з великим корпусом тренування мережі на всіх контекстах з корпусу займатиме багато часу.

Негативна проба (англ. *negative sampling*) є одним із способів вирішення цієї проблеми. Замість використанні всіх можливих контекстів, випадковим чином вибирається кілька негативних (тих що не мають у собі ключового слова). У результаті, якщо слово "гарна" з'являється в контексті слова "погода", то їх вектори будуть більш схожими ніж вектори кількох інших випадково вибраних слів (“тварина”, “країна”, т. д.), замість всіх інших слів з корпусу. Це робить тренування word2vec набагато швидшим при збереженні точності моделі.

3.3 Глобальні вектори

Метод векторного представлення слів “Глобальні вектори” (англ. *Global Vectors* або скорочено GloVe) в NLP був розроблений у Стенфордському університеті Пеннінгтоном та іншими [5]. Він отримав назву "глобальні вектори" через те, що модель безпосередньо використовує глобальну статистику корпусу. Він показує високу продуктивність в задачах світової аналогії та розпізнавання іменованих сутностей.

Ця техніка зменшує обчислювальні витрати на тренування моделі завдяки простішій квадратичній функції втрат або помилки, що в свою чергу призводить до отримання відмінних векторних представлень слів. Вона поєднує методи локального контексту, такі як модель пропуск-грама

Міколова, та методи глобальної матричної факторизації для створення низько розмірних представлень слів.

Метод латентного семантичного аналізу (LSA) є одним з методів глобальної матричної факторизації, який не демонструє високої ефективності в задачах світової аналогії, але використовує статистичну інформацію для побудови менш оптимальної векторної просторової структури.

Метод пропуск-грами у свою чергу показує кращі результати в задачі аналогії. Однак, він не ефективно використовує статистику корпусу через відсутність тренування на глобальних кількостях повторення слів.

Отже, на відміну від Word2Vec, який створює векторні представлення слів з використанням локального контексту, GloVe зосереджується на глобальному контексті для створення векторних представлень слів, що дає йому перевагу перед Word2Vec. У GloVe семантичні відношення між словами отримуються за допомогою матриці співвідношень.

3.3.1 Матриця співвідношень GloVe

Розглянемо два речення:

“Сьогодні гарна погода.”

“Завтра має бути гарна погода.”

Матриця для довжини контексту 1:

	сьогодні	гарна	погода	завтра	має	бути
сьогодні	0	1	0	0	0	0
гарна	1	0	2	0	0	1
погода	0	2	0	0	0	0
завтра	0	0	0	0	1	0
має	0	0	0	1	0	1

бути	0	1	0	0	1	0

Таблиця 3.1

Кожне значення в цій матриці представляє кількість співвідношень з відповідним словом в рядку/стовпці. Ця матриця співвідношень створена з використанням глобального підрахунку співвідношень слів (кількість разів, коли слова з'являлися послідовно; для розміру контексту = 1). Якщо в текстовому корпусі є 1 тис. унікальних слів (після попередньої обробки), то матриця співвідношень матиме розмірність 1 тис. на 1 тис. Основна ідея за GloVe полягає в тому, що співвідношення слів є найважливішою статистичною інформацією, доступною для моделі для "тренування" векторних представлень слів.

Для прикладу розглянемо ймовірності співвідношень для цільових слів "лід" (ice) та "пара" (steam) з різними пробними словами зі словника. Ось кілька фактичних ймовірностей з корпусу з 6 мільярдів слів:

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

де P(i|j) - ймовірність того, що і буде у контексті j. Вона вираховується як

$$P(i|j) = \frac{x_{ij}}{x_i}$$

де x_{ij} - - елемент матриці співвідношень, а x_i - - сума елементів матриці і-го рядка.

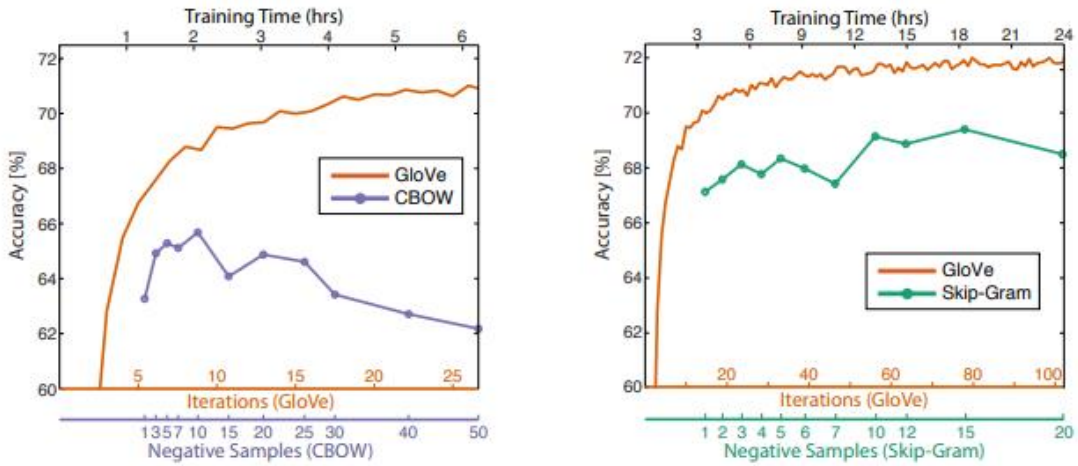
Візьмемо k = твердий, тобто слова, пов'язані з льодом, але не пов'язані з парою. Очікуване співвідношення Pik / Pjk буде великим (для і = лід, j = пар). Аналогічно, для слів k, які пов'язані з парою, але не з льодом, якщо ж k = газ - співвідношення буде невеликим. Для слів, таких

як вода або мода, які пов'язані або не пов'язані ні з льодом, ні з парою відповідно, співвідношення повинно бути приблизно одиницею.

Співвідношення ймовірностей дозволяє краще відрізняти відповідні слова (твердий і газ) від непов'язаних слів (мода і вода), ніж просто ймовірність. Воно також здатне краще розрізняти два відповідні слова. Тому в GloVe вихідною точкою для вивчення векторів слів є співвідношення ймовірностей співвідношень, а не самі ймовірності.

3.3.2 Порівняння з Word2Vec

На графіках (рисунок 3.1) продемонстрована загальна точність на задачі словесних аналогій як функція часу навчання, яка визначається кількістю ітерацій для GloVe і кількістю негативних проб для “Безперервної торби слів” (a) та “Пропуск-грами” (b). У всіх випадках тренуються 300-виміркові вектори на одному й тому ж корпусі з 6 млрд токенів з однаковим словником на 400 000 слів і симетричним контекстом розміром 10.



(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

Рисунок 3.1 - GloVe vs Word2Vec

РОЗДІЛ 4. ДЕРЕВА РІШЕНЬ

4.1 Кероване навчання

Кероване або контрольоване навчання [6], також відоме як кероване машинне навчання, є підкатегорією машинного навчання та штучного інтелекту. Воно визначається використанням попередньо категоризованих наборів даних для навчання алгоритмів, які класифікують дані або точно прогнозують результати. Коли вхідні дані надходять у модель, вона коригує свої ваги, доки модель не буде налаштована належним чином, що відбувається як частина процесу перехресної перевірки (англ. cross-validation).

Кероване навчання використовує навчальний набір даних, щоб навчити моделі отримувати бажаний результат. Цей навчальний набір даних включає вхідні та правильні вихідні дані, які дозволяють моделі навчатися з часом. Алгоритм вимірює свою точність за допомогою функції втрат, коригуючи, доки помилка не буде достатньо мінімізована.

Кероване навчання можна розділити на два типи проблем під час інтелектуального аналізу даних — класифікація та регресія:

Класифікація використовує алгоритм для точного віднесення тестових даних до певних категорій. Вона розпізнає конкретні сутності в наборі даних і намагається зробити деякі висновки про те, як ці сутності слід позначати або визначати. Поширеними алгоритмами класифікації є лінійні класифікатори, опорні векторні машини (SVM), дерева рішень, k-найближчий сусід і випадковий ліс (дерева рішень).

Регресія використовується для розуміння зв’язку між залежними та незалежними змінними. Лінійна регресія, логістична регресія та поліноміальна регресія є популярними алгоритмами регресії.

4.2 Дерева рішень у машинному навчанні

Дерево рішень — це непараметричний контрольований алгоритм навчання, який використовується як для завдань класифікації, так і для регресії. Воно має ієрархічну структуру дерева, яка складається з кореневого вузла, гілок, внутрішніх вузлів і листових вузлів (рисунок 4.1).

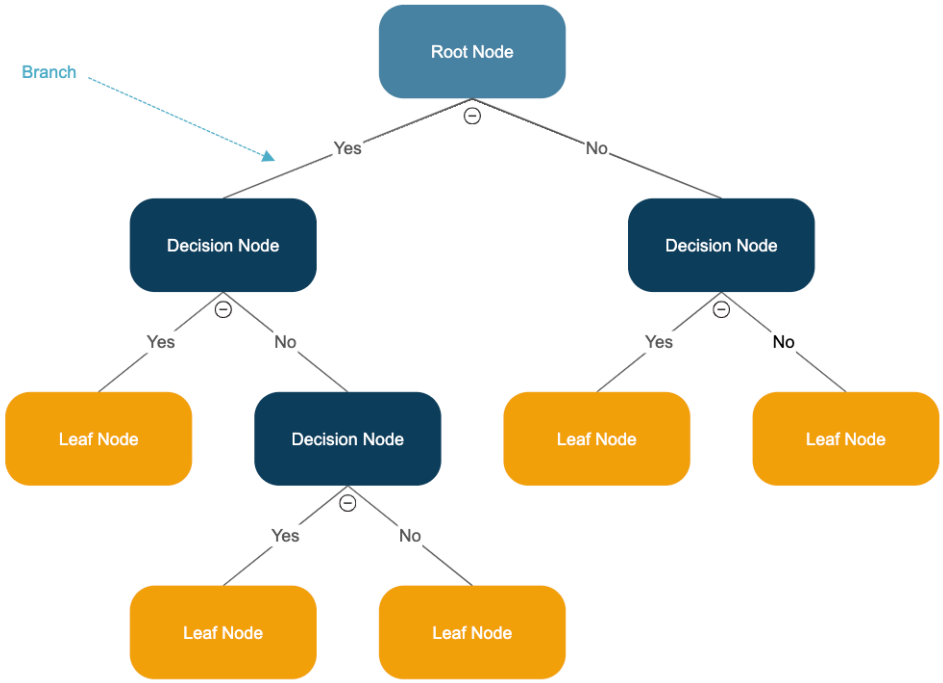


Рисунок 4.1

Дерево рішень починається з кореневого вузла, який не має вхідних гілок. Вихідні гілки від кореневого вузла потім надходять у внутрішні вузли, також відомі як вузли прийняття рішень. На основі доступних функцій обидва типи вузлів проводять оцінки для формування однорідних підмножин, які позначаються листовими вузлами або кінцевими вузлами. Листові вузли представляють усі можливі результати в наборі даних

Навчання дерева рішень використовує стратегію «розділяй і володарюй», проводячи «жадібний пошук» (англ. *greedy algorithm*), щоб визначити оптимальні точки поділу всередині дерева. Потім цей процес

поділу повторюється рекурсивним способом зверху вниз, доки всі або більшість записів не буде класифіковано за певними категоріями. Чи всі точки даних класифікуються як однорідні набори, значною мірою залежить від складності дерева рішень. Меншим деревам легше отримати чисті вузли листя, тобто точки даних в одному класі.

Однак при збільшенні розміру дерева стає все важче підтримувати цю чистоту, і це зазвичай призводить до того, що в дане піддерево потрапляє занадто мало даних. Такий процес називають фрагментацією даних, і це часто може призвести до перенавчання (англ. *overfitting*). Як наслідок, у контексті дерев рішень надають перевагу малим деревам. Інакше кажучи, дерева рішень повинні додавати складності лише за необхідності, оскільки найпростіше пояснення часто є найкращим.

Щоб зменшити складність і запобігти перенавчанню, зазвичай використовується обрізка; це процес, який видаляє гілки, які розбиваються на функції з низьким значенням. Відповідність моделі потім можна оцінити за допомогою процесу перехресної перевірки. Ще один спосіб, за допомогою якого дерева рішень можуть підтримувати свою точність, — це формування ансамблю за допомогою алгоритму випадкового лісу; цей класифікатор передбачає більш точні результати, особливо коли окремі дерева не корельовані одне з одним.

4.3 Випадковий ліс

4.3.1 Ансамбль моделей

Методи ансамблювання складаються з набору класифікаторів, наприклад, дерев прийняття рішень, і їх прогнози агрегуються для визначення найпопулярнішого результату.

Найвідоміші методи ансамблювання - це беггінг (англ. *bagging*), також відомий як бутстреп-агрегація (англ. *bootstrap aggregating*) та бустінг (англ. *boosting*). У 1996 році Лео Брейман вперше представив метод беггінгу [7]; в цьому методі вибирається випадковий вибірок даних із навчального набору з повторенням - це означає, що окремі точки даних можуть бути вибрані більше одного разу. Після створення кількох вибірок даних ці моделі навчаються незалежно, і в залежності від типу завдання - регресії або класифікації - середнє або більшість цих прогнозів дають більш точну оцінку. Цей підхід часто використовується для зменшення варіації у шумних наборах даних.

4.3.2 Алгоритм випадкового лісу

Алгоритм випадкового лісу є розширенням методу беггінгу, оскільки він використовує як беггінг, так і випадковість ознак для створення некорельованого лісу дерев прийняття рішень. Випадковість ознак, також відома як беггінг ознак або "Метод випадкового підпростору" [8], генерує випадкову підмножину ознак, що забезпечує низьку кореляцію між деревами прийняття рішень. Це ключова відмінність між деревами прийняття рішень та випадковими лісами. Подолання усіх можливих розбиттів ознак розглядають дерева прийняття рішень, тоді як випадкові ліси вибирають лише підмножину цих ознак.

Алгоритми випадкового лісу мають три основні параметри, які необхідно встановити перед навчанням. До них належать розмір вузла, кількість дерев і кількість відібраних ознак. Далі, класифікатор випадкового лісу може бути використаний для вирішення задач регресії або класифікації.

Алгоритм випадкового лісу складається з набору дерев рішень, і кожне дерево в ансамблі складається із вибірки даних (рисунок 4.2),

отриманої з навчального набору із заміною, яка називається бутстреп вибіркою. З цієї навчальної вибірки частина, наприклад третина, відкладена для тестування. Інший фактор випадковості вводиться через баггінг ознак, додаючи більше різноманітності до набору даних і зменшуючи кореляцію між деревами рішень. Залежно від типу проблеми визначення прогнозу буде різним. Для завдання регресії буде усереднено окремі дерева рішень, а для завдання класифікації – більшість голосів, тобто. найпоширеніша категоріальна змінна — дасть прогнозований клас. Наприкінці, тестовий зразок використовується для перехресної перевірки, завершуючи процес передбачення.

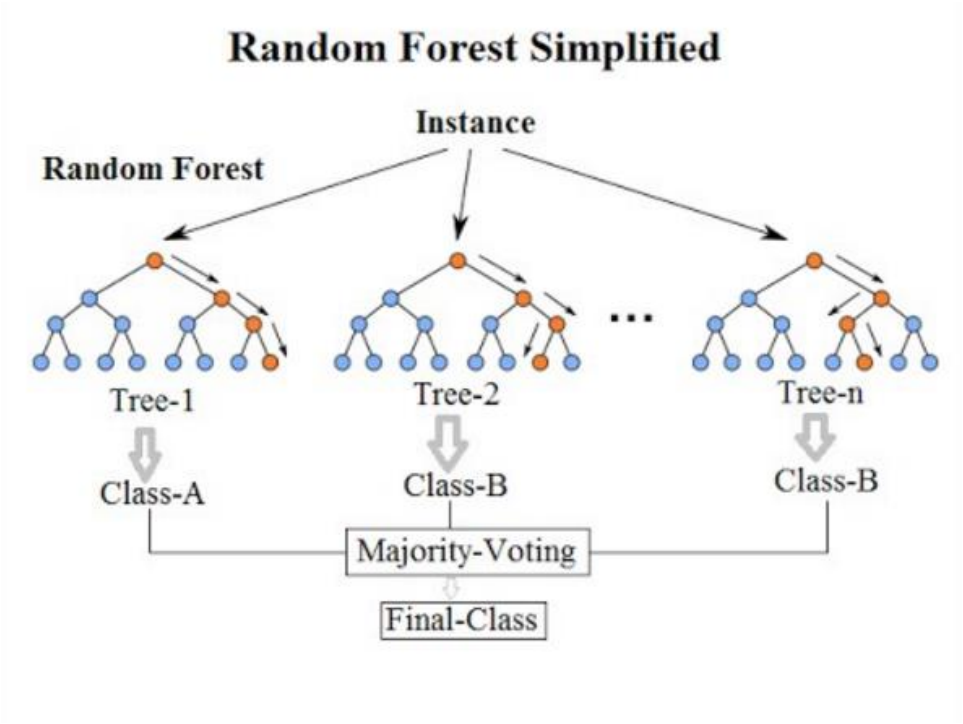


Рисунок 4.2 - Класифікація за допомогою випадкового лісу

Переваги:

1. Зменшений ризик перенавчання: Дерева прийняття рішень мають ризик перенавчання, оскільки вони мають тенденцію дуже щільно підлаштовуватися під всі зразки в тренувальних даних. Однак, коли

випадковий ліс містить значну кількість дерев, класифікатор не перенавчатиме модель, оскільки усереднення некорельованих дерев знижує загальну дисперсію та помилку прогнозу.

2. Надає гнучкість: Оскільки випадковий ліс може справлятися як з завданнями регресії, так і класифікації з високою точністю, він є популярним методом серед дослідників даних. Беггінг ознак також робить класифікатор випадкового лісу ефективним інструментом для оцінки відсутніх значень, оскільки він зберігає точність при відсутності частини даних.
3. Легко визначити важливість ознак: Випадковий ліс дозволяє легко оцінювати важливість змінних або їх внесок у модель. Існують кілька способів оцінити важливість ознак. Важливість за критерієм Джині (англ. *Gini importance*) та середнє зменшення нечистоти (англ. *mean decrease in impurity* або MDI) зазвичай використовуються для вимірювання того, наскільки зменшується точність моделі, коли виключається певна змінна. Однак, існує ще один показник важливості - важливість перестановки, також відома як середнє зменшення точності (англ. *permutation importance* або MDA). MDA визначає середнє зменшення точності шляхом випадкової перестановки значень ознак у вибірках поза мішенню.

Недоліки:

1. Процес займає багато часу: оскільки алгоритми випадкового лісу можуть обробляти великі набори даних, вони можуть надавати точніші прогнози, але можуть повільно обробляти дані, оскільки вони обчислюють дані для кожного окремого дерева рішень.
2. Вимагає більше ресурсів: оскільки випадкові ліси обробляють більші набори даних, їм знадобиться більше ресурсів для зберігання цих даних.

3. Більш складний: передбачення окремого дерева рішень легше інтерпретувати порівняно з лісом із них.

4.3.2 Застосування випадкового лісу

Алгоритм випадкового лісу застосовувався в багатьох галузях, що дозволило їм приймати кращі бізнес-рішення. Деякі випадки використання включають:

- Фінанси: цьому алгоритму надають перевагу порівняно з іншими, оскільки він скорочує час, витрачений на завдання керування даними та попередньої обробки. Його можна використовувати для оцінки клієнтів із високим кредитним ризиком, виявлення шахрайства та проблем із ціною опціонів.
- Охорона здоров'я: алгоритм випадкового лісу має застосування в обчислювальній біології, що дозволяє лікарям вирішувати такі проблеми, як класифікація експресії генів, виявлення біомаркерів і анотація послідовностей. У результаті лікарі можуть оцінити реакцію на певні ліки.
- Електронна комерція: її можна використовувати для механізмів рекомендацій для перехресних продажів.

РОЗДІЛ 5. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КЛАСИФІКАЦІЇ ЛЮДСЬКИХ ЗАХВОРЮВАНЬ

5.1 Опис датасету для навчання

Для навчання моделі було використано невеликий англомовний датасет типу симптом-хвороба [9]. Його обробка виконувалась за допомогою бібліотеки Pandas.

Не беручи до уваги індекс, датасет має дві колонки: “label” та “text”.

- label: містить назви хвороб
- text: містить описи симптомів природною мовою.

Набір даних включає 24 хвороби, кожна з яких має 50 описів симптомів, загалом 1200 точок даних (рисунок 5.1).

```
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  1200 non-null   int64
 1   label       1200 non-null   object
 2   text        1200 non-null   object
dtypes: int64(1), object(2)
memory usage: 28.2+ KB
```

Рисунок 5.1 - Загальна інформація про вибраний датасет

1. Колонка “label”.

Містить назви захворювань (рисунок 5.2).

0	Psoriasis	1075	drug reaction
1	Psoriasis	903	urinary tract infection
2	Psoriasis	1102	peptic ulcer disease
3	Psoriasis	438	Pneumonia
4	Psoriasis	1138	peptic ulcer disease
	...	0	Psoriasis
1195	diabetes	637	Bronchial Asthma
1196	diabetes	865	Malaria
1197	diabetes	911	urinary tract infection
1198	diabetes	440	Pneumonia
1199	diabetes	371	Common Cold

Рисунок 5.2 - Прикладна вибірка даних у колонці “label”

Перелік хвороб включених в набір даних:

Псоріаз, варикозна вена, тиф, вітрянка, імпетиго, денге, грибкова інфекція, простуда, пневмонія, геморой, артрит, акне, бронхіальна астма, артеріальна гіпертензія, мігрень, спондильоз, жовтяниця, малярія, сечовипускання інфекція, алергія, гастроезофагеальна рефлюксна хвороба, небажана реакція на ліки, пептична виразка, діабет.

```
['Psoriasis' 'Varicose Veins' 'Typhoid' 'Chicken pox' 'Impetigo' 'Dengue'
 'Fungal infection' 'Common Cold' 'Pneumonia' 'Dimorphic Hemorrhoids'
 'Arthritis' 'Acne' 'Bronchial Asthma' 'Hypertension' 'Migraine'
 'Cervical spondylosis' 'Jaundice' 'Malaria' 'urinary tract infection'
 'allergy' 'gastroesophageal reflux disease' 'drug reaction'
 'peptic ulcer disease' 'diabetes']
```

Рисунок 5.3 - Наведені у датасеті унікальні хвороби

2. Колонка “text”.

Містить описи симптомів, стану здоров’я, природною мовою, які б відчувала людина якби мала відповідне у колонці “label” захворювання (рисунок 5.2). Усі значення унікальні, за винятком семи дублікатів.

```
0       I have been experiencing a skin rash on my arms, legs, and torso for the past few week...
1       My skin has been peeling, especially on my knees, elbows, and scalp. This peeling is o...
2       I have been experiencing joint pain in my fingers, wrists, and knees. The pain is ofte...
3       There is a silver like dusting on my skin, especially on my lower back and scalp. This...
4       My nails have small dents or pits in them, and they often feel inflammatory and tender...
                                              ...
1195    I'm shaking and trembling all over. I've lost my sense of taste and smell, and I'm exh...
1196    Particularly in the crevices of my skin, I have skin rashes and irritations. My skin b...
1197    I regularly experience these intense urges and the want to urinate. I frequently feel ...
1198    I have trouble breathing, especially outside. I start to feel hot and start to sweat. ...
1199    I constantly sneeze and have a dry cough. My infections don't seem to be healing, and ...
```

```
1033    Even when I don't have anything acidic in my stomach, I constantly have a sour taste i...
386     I've been coughing a lot and finding it difficult to breathe. My throat hurts and I fe...
1149    I unintentionally lose weight and find it challenging to gain it back. To relieve the ...
622     I've been feeling really ill lately. I've had this persistent cough and difficulty bre...
175     I have seen rashes on my skin, and it's difficult for me not to scratch. Also, I've ha...
1058    I am experiencing changes in my menstrual cycle and unexpected vaginal discharge. I of...
729     Along with distorted eyesight, excessive appetite, a painful neck, melancholy, irritab...
659     My symptoms include a headache, chest pain, dizziness, lack of balance, and trouble co...
1143    I've lost a lot of stuff because of my bloody stools, including iron and bloos. As a r...
1038    It feels like food or acid is backing up in my throat. My chest discomfort only become...
```

Рисунок 5.4 - Прикладна вибірка даних у колонці “text”

5.2 Векторизація тексту та навчання моделі

Перед початком тренування потрібно виділити дані для тестування. У роботі, на тестування було відділено випадковим вибором 20 відсотків всього датасету. Тестування буде розглянуто у розділі 5.3.

Для підібраного датасету добре підійде TF-IDF векторизатор TfidfVectorizer з бібліотеки scikit-learn. Порівнюючи його з складнішими алгоритмами векторизації типу Word2Vec, можна навести декілька причин щодо його використання:

- Простота: TF-IDF - це простий алгоритм порівняно з складними техніками, такими як Word2Vec. Він простий у використанні і не потребує великої кількості настроювань параметрів або обширних навчальних даних.
- Ефективність для класифікації тексту: TF-IDF — це добре налагоджений алгоритм для завдань класифікації тексту. Він широко використовувався та довів свою ефективність у різних програмах

обробки природної мови, включаючи класифікацію документів та аналіз настроїв.

- Інтерпретованість: TF-IDF генерує зрозумілі ознаки. Кожна ознака представляє важливість слова у документі та в усьому корпусі. Це може надати уявлення про те, які симптоми є більш важливими для передбачення певних хвороб.
- Невеликий розмір набору даних: якщо ваш набір даних відносно малий, TF-IDF все одно може забезпечити хороші результати. Моделі, подібні до Word2Vec, зазвичай вимагають великих обсягів даних, щоб дізнатися точні представлення слів, що може бути неможливим за обмежених даних.

Також застосування n-грам майже завжди підвищить точність моделі, тому був застосований найрозповсюдженіший варіант - триграму для векторизації тексту.

Таким чином векторизований корпус датасету можна побачити на рисунку 5.5

```
'rts' ' ot' 'otr' 'epi' ' wa' 'on ' 'ur ' 'nsi' 'mid' 's-f' 'tee' 'poo'
'ped' 'roa' 'hac' 'fe.' 'ul-' 'am ' ' li' 'odo' 'ind' ' ra' ' wo' 'ons'
'lp!' 'oom' 'wou' 'loc' ' re' 'npl' 'avy' 'avo' ' sc' 'mat' 'hop' 'irl'
'ge,' 'sgu' 'bum' 'pic' 'izi' 'ode' 'nfa' 'ry,' 'tio' "y'v" 'tod' 'mpi'
'tas' 'oub' 'lie' ' sn' 'nki' 'bbe' 'uin' 'lod' 'e.i' 'abi' 'et,' 'nus'
'agi' 'age' 'thl' 'ze,' 'il ' 'xio' 'uff' 'ue.' 'vem' 'bat' 'nf ' 'eos'
'urb' 'cco' 'mso' 'ais' 'aso' ' ly' '-co' 'udy' 'tat' ' so' 'ora' 'cke'
"re'" 'wro' 'ypi' 'mou' 'ey,' 't, ' 'kne' 'id.' 'liz' 'puk' 'nla' 'fal'
'ep ' ' vo' 'nio' "sh'" 'tt.' 'ze.' 'd? ' 'wan' 'ody' 't-c' 'ty ' 'st.'
'ny ' ' ch' 'dro' 'eru' ' ro' 'mia' 'oze' 'acq' 'cru' 'son' 'sta' 'ras'
'nce' 'fai' 'bra' 'won' 'lut' 'hey' 'ake' 'be.' 'reh' 'bef' 'ckn' 'nse'
'kac' 'od ' 'ler' 'ls.' 'ife' 'try' 'so.' 'abu' "ey'" 'puf' 'siv' 'mpl'
"dn'" 'ibl' 'eap' 'bet' 'mee' 'olt' 'tra' 'act' 'owe' 'par' 'cup' 'mb.'
'rte' 'pel' 'nex' 'g, ' 'wed' 'fec' 'bar' 'nda' 'lym' ' hi' 'dec' 'rod'
```

Рисунок 5.5 - Прикладна вибірка триграм корпусу

Далі, використовуючи векторизований текст, була натренований класифікатор за алгоритмом випадкового лісу RandomForestClassifier з бібліотеки scikit-learn.

5.3 Метрики моделі

Після тренування важливо протестувати отриману модель. Збережені 20 відсотків були використані, отримати детальний звіт точності натренованої моделі. На рисунку 5.6 наведені результати метрик для кожного класу. На рисунку 5.7 - загальні результати моделі, зокрема точність, яка дорівнює 96 відсоткам.

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	8
Arthritis	1.00	1.00	1.00	10
Bronchial Asthma	1.00	1.00	1.00	13
Cervical spondylosis	0.91	1.00	0.95	10
Chicken pox	0.80	0.86	0.83	14
Common Cold	1.00	1.00	1.00	7
Dengue	0.67	0.50	0.57	8
Dimorphic Hemorrhoids	1.00	1.00	1.00	10
Fungal infection	1.00	1.00	1.00	6
Hypertension	1.00	1.00	1.00	18
Impetigo	1.00	0.94	0.97	16
Jaundice	1.00	1.00	1.00	8
Malaria	1.00	1.00	1.00	9
Migraine	1.00	1.00	1.00	9
Pneumonia	1.00	1.00	1.00	11
Psoriasis	0.89	1.00	0.94	8
Typhoid	1.00	0.80	0.89	10
Varicose Veins	0.82	1.00	0.90	9
allergy	1.00	1.00	1.00	13
diabetes	1.00	1.00	1.00	7
drug reaction	1.00	1.00	1.00	9
gastroesophageal reflux disease	1.00	1.00	1.00	12
peptic ulcer disease	1.00	1.00	1.00	5
urinary tract infection	1.00	1.00	1.00	10

Рисунок 5.6

accuracy			0.96	240
macro avg	0.96	0.96	0.96	240
weighted avg	0.96	0.96	0.96	240

Рисунок 5.7

Наочно видно продуктивність алгоритму у матриці невідповідностей, наведеній на рисунку 5.8.

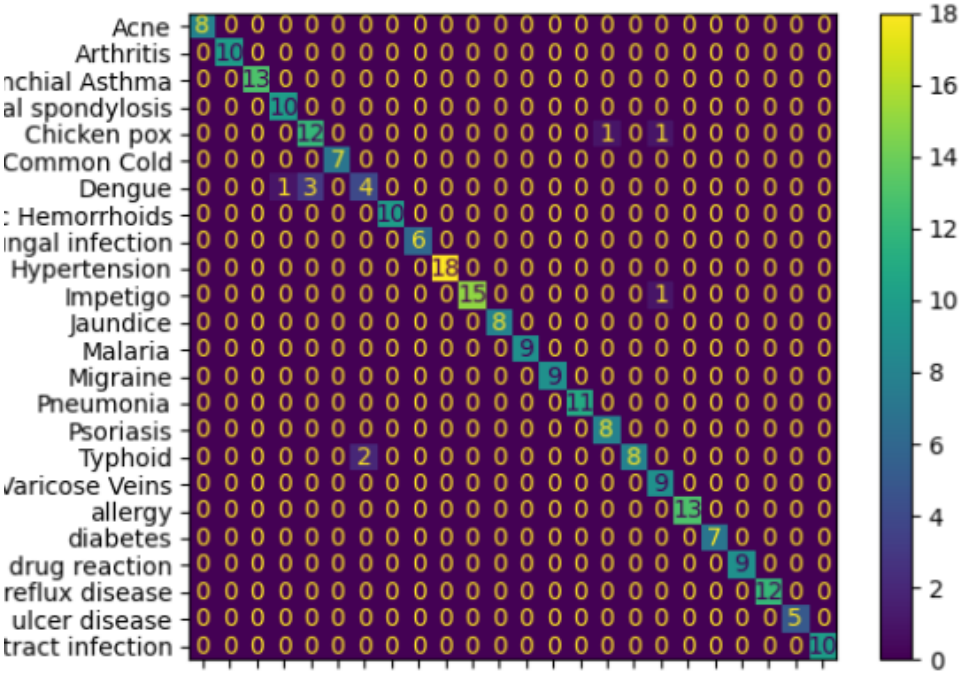


Рисунок 5.8

Також була порахована метрика перехресної перевірки, вона наведена на рисунку 5.9.

```
Cross value score: 0.9766666666666666 +- 0.011055415967851333
```

Рисунок 5.9

5.4 Імплементація простого телеграм бота з використанням навчених моделей

Для наочної демонстрації роботи моделі, прости використання та можливості подальшого розширення функціоналу був імплементований простий телеграм бот використовуючи базову бібліотеку python-telegram-bot [10].

Бот має мінімальний функціонал:

- Команда “start”. Надіслана відповіть вітає користувача.
- Команда “help”. Надіслана відповіть описує роботу бота та перелічує захворювання, що підтримуються для класифікації.
- Використання натренованої моделі для відповіді на вхідне повідомлення. Повідомлення користувача проходять векторизацію, потім вона використовується як вхідні дані для моделі, що передбачає хворобу. У відповідь бот надсилає просте повідомлення
  “You might have <disease1>. Confidence score: <score1>
  Next most likely prediction is <disease2> (score2)
  Please consult with a qualified healthcare professional”,
  де <disease> замінюється передбаченою моделлю хворобою, а <score> - відповідна оцінка впевненості.

5.5 Демонстрація результатів

Код програми можна побачити у додатку №1.

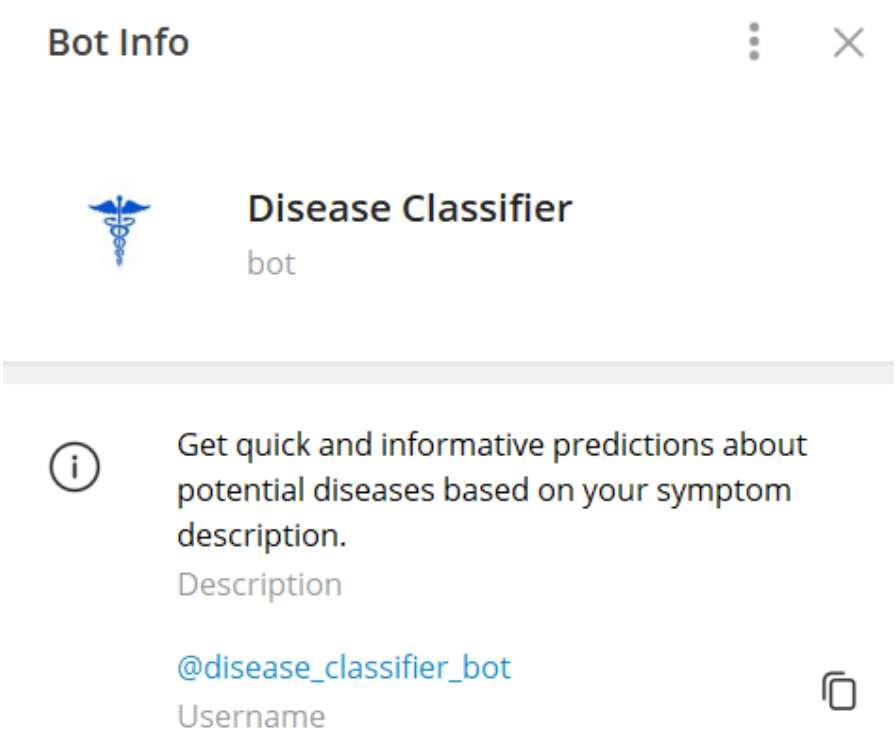


Рисунок 5.10 - Біографія бота

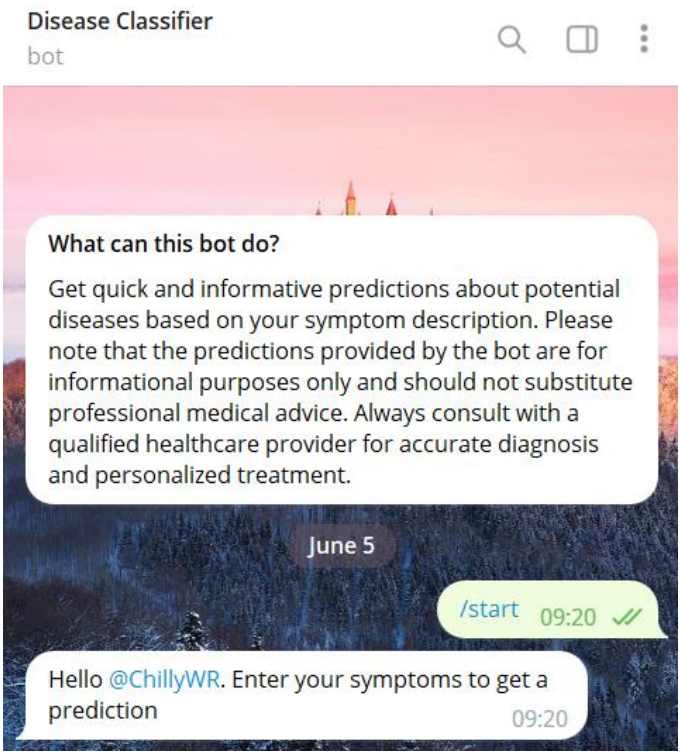


Рисунок 5.11 - Результат виконання команди “start”

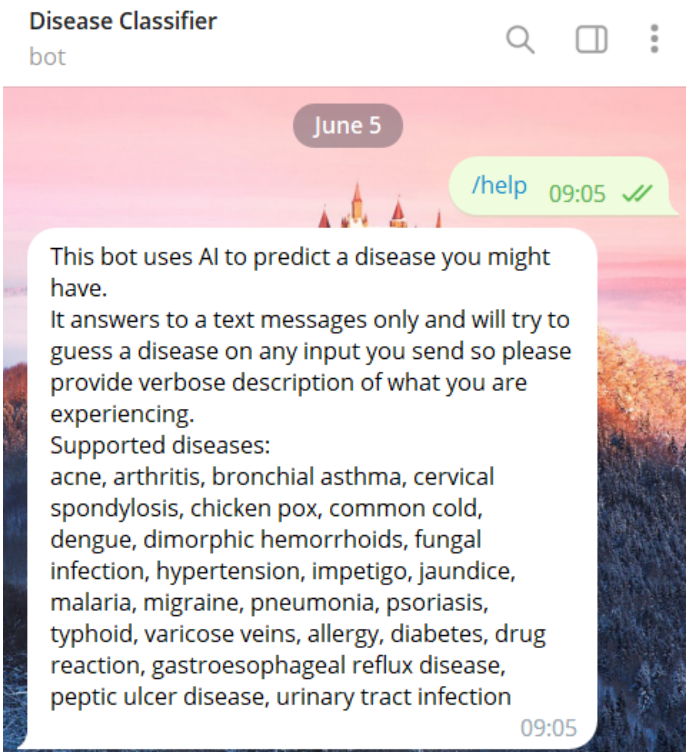


Рисунок 5.12 - Результат виконання команди “help”

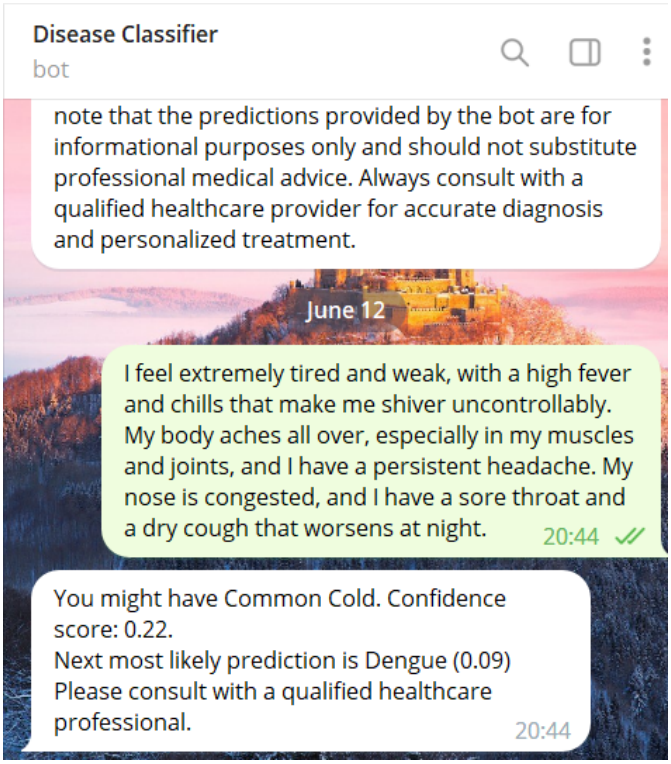


Рисунок 5.13 - Результат класифікації

ВИСНОВКИ

У роботі будо розглянуто низку методів та алгоритмів обробки природньої мови. Базові методи попередньої обробки тексту та його векторизації, від простих - Торба слів, TF-IDF, до сучасних комплексних - Word2Vec, GloVe, що здатні передавати семантичне значення слів.

Результатом кваліфікаційної роботи стало програмне забезпечення, що реалізує роботу моделі нейронної мережі для класифікації тексту за допомогою алгоритму випадкового лісу. Для прикладу модель була натренована на невеликому датасеті “симптоми-хвороба” з досягнутою точністю близько 95 відсотків на тестових даних.

Задля демонстрації роботи моделі як інтерфейс був розроблений чат-бот у месенджері Telegram, що використовує побудовану модель як генератор відповіді на повідомлення користувача.

У перспективі таку модель можна використати як асистента сімейного лікаря, що після обробки симптомів пацієнта зможе згенерувати список можливих захворювань та дати відповідні направлення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Preprocessing / Scikit-learn — Методи попередньої обробки тексту
2. Efficient Estimation of Word Representations in Vector Space / Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean
3. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method / Yoav Goldberg, Omer Levy
4. Word2vec / Google Code Archive — інформація про проект
5. GloVe: Global Vectors for Word Representation / Jeffrey Pennington, Richard Socher, Christopher D. Manning
6. Supervised learning / IBM
7. Bagging Predictors / Leo Breiman
8. Random Forest / Leo Breiman Statistics Department University of California Berkeley, CA 94720 - Метод випадкового підрозділу
9. Symptom2Disease / Niyar R Barman – Kaggle датасет
10. Python-telegram-bot - python library, a wrapper for Telegram API

ДОДАТКИ

Додаток №1. Код програми

source.py

```
from bot import start
from const import DATASET_FILE_PATH
from input_parser import prepare_train_data
from nlp import prepare_model


def main():
    print("Preparing dataset")
    X, y = prepare_train_data(DATASET_FILE_PATH)

    print("Preparing model")
    model = prepare_model(X, y)

    start(model)


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

const.py

```
DATASET_FILE_PATH = "./dataset/Symptom2Disease.csv"
```

input_parser.py:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


def load_dataset(file_path: str) -> pd.DataFrame:
```

```
    return pd.read_csv(file_path)


def prepare_train_data(file_path: str) -> (np.ndarray, np.ndarray):
    df = load_dataset(file_path)
    df.info()

    X = df['text']
    y = df['label']

    # pd.set_option('display.max_colwidth', 90)
    # print(X)
    # print(X.sample(11))
    # print(X.describe())
    # print(y)
    # print(y.sample(11))
    # print(y.unique())
    # print(y.describe())

    return X, y
```

nlp.py:

```
from dataclasses import dataclass

import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split,
cross_val_score


class Model:
    @dataclass
    class __Data:
```

```
        X: np.ndarray
        y: np.ndarray

    __train_data: __Data
    __test_data: __Data

    def __init__(self):
        self._vectorizer = TfidfVectorizer(analyzer='char',
ngram_range=(3, 3))
        self._classifier = RandomForestClassifier()

    def fit(self, X_input, y_input):
        X = self._vectorizer.fit_transform(X_input)

        X_train, X_test, y_train, y_test = train_test_split(X,
y_input, test_size=0.2, shuffle=True)
        self.__train_data = Model.__Data(X_train, y_train)
        self.__test_data = Model.__Data(X_test, y_test)

        self._classifier.fit(X_train, y_train)

    def predict(self, _input: list[str]) -> np.ndarray:
        _output =
self._classifier.predict(self._vectorizer.transform(_input))
        return _output

    def predict_proba(self, _input: list[str]) -> np.ndarray:
        _output_proba =
self._classifier.predict_proba(self._vectorizer.transform(_input))
        return _output_proba

    def get_ordered_predictions(self, _input: list[str]):
        confidence_scores = self.predict_proba(_input)
        sort_indices = np.argsort(-confidence_scores, axis=1)
        return np.dstack((
            np.array(self._classifier.classes_)[sort_indices],  #
sort classes for each entry in sort_indices
```

```
            np.take_along_axis(confidence_scores, sort_indices,
axis=1)
        ))

    def measure_efficiency(self):
        y_pred = self._classifier.predict(self.__test_data.X)
        print(f"Classification
Report:\n{classification_report(self.__test_data.y, y_pred)}")

        ConfusionMatrixDisplay.from_predictions(self.__test_data.y,
y_pred, display_labels=self._classifier.classes_)
        plt.show()

    def measure_cross_value_score(self, X_input, y_input):
        cvs = cross_val_score(self._classifier,
self._vectorizer.transform(X_input), y_input, cv=10)
        print(f"Cross value score: {cvs.mean()} +- {cvs.std()}")

    def get_classes(self) -> list[str]:
        return self._classifier.classes_

    def get_vectorizer_features(self):
        return self._vectorizer.get_feature_names_out()


def prepare_model(X, y):
    model = Model()
    model.fit(X, y)

    model.measure_efficiency()
    model.measure_cross_value_score(X, y)

    return model
```

bot.py

```
import logging
import os
from abc import ABC, abstractmethod

from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler,
MessageHandler, ContextTypes, filters


class ClassificationModel(ABC):
    """ClassificationModel is a model interface bot needs to function"""

    @abstractmethod
    def get_ordered_predictions(self, _input: list[str]):
        pass

    @abstractmethod
    def get_classes(self) -> list[str]:
        pass


_Model: ClassificationModel


async def start_command(update: Update, context: ContextTypes.DEFAULT_TYPE)
-> None:
    logging.info("Triggered command '/start'")
    await update.message.reply_text(f'Hello {update.effective_user.name}.
Enter your symptoms to get a prediction.')


async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE)
-> None:
    logging.info("Triggered command '/help'")
    await update.message.reply_text(
        f"This bot uses AI to predict a disease you might have.\n"
        f"It answers to a text messages only and will try to guess a
disease on any input you send "
        f"so please provide verbose description of what you are
experiencing.\n"
        f"Supported diseases:\n"
```

```
        f"{', '.join(_Model.get_classes()).lower()}")


async def classify_disease(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    text = update.message.text
    logging.info(f"Received message: {text}")
    predictions = _Model.get_ordered_predictions([text])[0]
    logging.info(f"Predictions: {predictions}")
    await update.message.reply_text(
        f"You might have {predictions[0][0]}. Confidence score:
{predictions[0][1]}.\n"
        f"Next most likely prediction is {predictions[1][0]}
({predictions[1][1]})\n"
        f"Please consult with a qualified healthcare professional.")


def start(model):
    bot_token = os.environ.get('TELEGRAM_BOT_TOKEN')
    if bot_token == '':
        raise Exception('TELEGRAM_BOT_TOKEN env var must be set')

    global _Model
    _Model = model
    logging.basicConfig(level=logging.INFO)

    application = ApplicationBuilder().token(bot_token).build()

    application.add_handler(CommandHandler("start", start_command))
    application.add_handler(CommandHandler("help", help_command))

    application.add_handler(MessageHandler(filters=filters.TEXT &
~filters.COMMAND, callback=classify_disease))

    application.run_polling()
```