МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 59.00.00.000 ПЗ

Група ІПм-23-3

Синишен Владислав

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Синишен Владислав Петрович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи фреймворків автоматизації рішення задач Data Science

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Синишен В.П.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Мельник Віталій Дмитрович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Синишену Владиславу Петровичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи фреймворків автоматизації рішення задач Data Science ”

керівник проекту (роботи) Мельник Віталій Дмитрович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування програмних технологій Data Science

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз предметної області автоматизації рішень в Data Science
2. Підходи генерування персоналізованого контенту засобами розмовних інтерфейсів
3. Представлення власного рішення
4. Оцінка запропонованого фреймворку автоматизації рішення задач Data Science

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Бажані рівні автоматизації на кожному етапі проекту DS/ML (рис. 1.1)
2. Огляд методології дослідження розмовних інтерфейсів (рис. 1.2)
3. Методологія дослідження (рис. 1.3)
4. Приклад запуску Auto-WEKA на наборі даних iris (рис. 1.4)
5. Три обрані типи особистості чат-боту, використані для тестів користувачів (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Аналіз предметної області автоматизації рішень в Data Science	15.10.2024	виконано
4	Підходи генерування персоналізованого контенту засобами розмовних інтерфейсів	08.11.2024	виконано
5	Представлення власного рішення	20.11.2024	виконано
6	Оцінка запропонованого фреймворку автоматизації рішення задач Data Science	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 80 с., 35 рис., 4 табл., 52 джерел.

Тема: Моделі та методи фреймворків автоматизації рішення задач Data Science

Об'єкт дослідження: процеси автоматизації вирішення задач Data Science.

Мета роботи: розробка та оцінка фреймворку автоматизації рішень у Data Science із використанням технологій AutoML і персоналізованих розмовних інтерфейсів.

Предмет дослідження: методи та інструменти розробки персоналізованих фреймворків автоматизації задач Data Science із використанням розмовних інтерфейсів.

Результати дослідження

Розроблений фреймворк є ефективним інструментом для автоматизації задач Data Science. Його універсальність, простота у використанні та адаптивність до різних типів задач відкривають нові можливості для впровадження інновацій у процеси аналізу даних.

Висновок

В магістерській роботі запропоновано механізм персоналізації контенту чат-бота до специфічних потреб користувачів у контексті задач Data Science.

DATA SCIENCE, АВТОМАТИЗАЦІЯ, ПЕРСОНАЛІЗАЦІЯ, ЧАТ-БОТ, РОЗМОВНІ ІНТЕРФЕЙСИ, ОЧИЩЕННЯ ДАНИХ, ВИЯВЛЕННЯ АНОМАЛІЙ, ГЕНЕРАЦІЯ КОДУ, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Master Thesis: 80 pp., 35 fig., 4 tab., 52 sources.

Thesis Subject: Models and methods of Data Science task automation frameworks

Object of research: Data Science task automation processes.

Purpose of work: development and evaluation of a Data Science decision automation framework using AutoML technologies and personalized conversational interfaces.

Subject of research: methods and tools for developing personalized Data Science task automation frameworks using conversational interfaces.

Research results

The developed framework is an effective tool for automating Data Science tasks. Its versatility, ease of use, and adaptability to different types of tasks open up new opportunities for introducing innovations into data analysis processes.

Conclusion

The master's thesis proposes a mechanism for personalizing chatbot content to specific user needs in the context of Data Science tasks.

DATA SCIENCE, AUTOMATION, PERSONALIZATION, CHAT-BOT, CONVERSAL INTERFACES, DATA CLEANSING, ANOMALIES DETECTION, CODE GENERATION, SOFTWARE ARCHITECTURE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AutoML – Automated machine learning

DS - Data Science

EDA - Exploratory Data Analysis

IA – Artificial Intelligence

NLP – Natural Language Processing

ETL - Extract, Transform, Load

NLP - Natural Language Processing

CV - Computer Vision

RL - Reinforcement Learning

SGD - Stochastic Gradient Descent

ML – Machine Learning

GAN - Generative Adversarial Network

KNN - K-Nearest Neighbors

PCA - Principal Component Analysis

TPOT - Tree-based Pipeline Optimization Tool

ВСТУП

Актуальність теми.

У сучасному світі обсяг даних зростає експоненційно, що створює значний попит на швидкі та ефективні методи їхньої обробки й аналізу. Data Science як дисципліна відіграє ключову роль у вирішенні цих завдань, проте зростання складності даних і необхідність їхньої попередньої обробки роблять традиційні підходи недостатньо ефективними. На цьому тлі автоматизація рутинних процесів аналізу даних стає важливим інструментом для підвищення продуктивності спеціалістів і компаній.

Водночас, автоматизація має враховувати індивідуальні потреби користувачів і забезпечувати зручну взаємодію з інструментами, оскільки рівень кваліфікації аналітиків може значно варіюватися. Більшість існуючих рішень орієнтовані на технічно підкованих користувачів, що обмежує їхню доступність для ширшої аудиторії. Технології автоматизованого машинного навчання (AutoML) уже зробили значний внесок у спрощення процесу побудови моделей, проте питання інтеграції AutoML із персоналізованими розмовними інтерфейсами залишається мало дослідженим.

Чат-боти, як розмовні агенти, мають потенціал для спрощення доступу до аналітичних інструментів і забезпечення інтуїтивного способу роботи із задачами Data Science. Їхня здатність спілкуватися природною мовою та адаптуватися до користувача може стати важливим елементом створення нових рішень. Персоналізація взаємодії через такі інтерфейси дозволяє не лише підвищити зручність користування, а й зробити процеси аналізу даних більш доступними для широкого кола спеціалістів.

Запропоноване дослідження є актуальним, оскільки воно фокусується на інтеграції сучасних технологій автоматизації та персоналізації, що дозволяє створити нові інструменти для ефективного аналізу даних. Це сприяє зниженню бар’єрів для входу в сферу Data Science, оптимізує

аналітичні процеси та відкриває нові можливості для розвитку бізнесу, науки й освіти.

Мета дослідження - розробка та оцінка фреймворку автоматизації рішень у Data Science із використанням технологій AutoML і персоналізованих розмовних інтерфейсів.

Об'єкт дослідження - процеси автоматизації вирішення задач Data Science.

Предмет дослідження - методи та інструменти розробки персоналізованих фреймворків автоматизації задач Data Science із використанням розмовних інтерфейсів.

Задачі дослідження:

- Провести аналіз сучасного стану автоматизації процесів у Data Science.

- Визначити проблемні аспекти існуючих рішень та окреслити напрямки їхнього вдосконалення.

- Розробити концепцію та архітектуру фреймворку автоматизації задач Data Science із персоналізацією.

- Реалізувати прототип фреймворку з використанням чат-бота та технологій AutoML.

- Провести тестування розробленого фреймворку на практичних кейсах, таких як виявлення аномалій та очищення даних.

Методи дослідження

1. Аналіз літератури: вивчення сучасних підходів до автоматизації задач Data Science та створення розмовних інтерфейсів.

2. Моделювання: розробка архітектури та функціональних модулів фреймворку.

3. Емпіричні методи: тестування фреймворку на реальних даних.

4. Порівняльний аналіз: оцінка ефективності розробленого рішення порівняно з існуючими підходами.

Наукова новизна отриманих результатів

Розроблено концептуальну модель інтеграції AutoML із розмовними інтерфейсами для автоматизації задач Data Science і реалізовано прототип фреймворку, що забезпечує ефективне рішення задач, таких як очищення даних та виявлення аномалій.

Практичне значення результатів

Розроблений фреймворк може бути використаний для автоматизації рутинних задач у Data Science, підвищуючи ефективність роботи як окремих спеціалістів, так і організацій. Його застосування дозволяє скоротити час на виконання типових задач, забезпечити високу якість даних і покращити комунікацію між системою та користувачем.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 80 сторінок, і містить 35 рисунків, 4 таблиці, список використаних джерел із 52 найменувань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ АВТОМАТИЗАЦІЇ РІШЕНЬ В DATA SCIENCE

1.1. Опис стану автоматизації процесів Data Science

Наука про дані (DS) та машинне навчання (ML) набирають популярності та сили в різних областях, таких як інформатика в охороні здоров'я [2], Інтернет речей [3] та програмна інженерія [4]. Застосування технологій ML і DS є пріоритетом для багатьох. Але, згідно з [5], це така складна та тривала діяльність, що бракує професіоналів, щоб заповнити попит на роботу. Нещодавно, щоб розширити використання та покращити результати, спільнота почала досліджувати різні рівні автоматизації виконання проектів DS/ML. Однак розробка автоматизації для підтримки користувачів протягом процесів DS/ML не є тривіальним завданням, оскільки це вимагає створення систем, здатних надавати відповіді для науковців даних з різним рівнем знань та експертизи.

В [5] провели опитування серед 217 фахівців з DS і ML, щоб визначити бажані рівні автоматизації протягом життєвого циклу проекту DS/ML, декілька з яких зацікавлені в проекті DS/ML: зацікавлені сторони, програмісти та доменні експерти. Однак автори вважають "науковців даних" лише тих, хто кодує та будує моделі. Вони класифікували автоматизацію на п'ять рівнів:

- без автоматизації (L0),
- керована людиною (L1),
- запропонована системою (L2),
- керована системою (L3),
- повна автоматизація (L4).

Рисунок 1.1 зображує інтерес усіх учасників щодо рівнів автоматизації на етапах побудови моделі та інженерії ознак. Однак більшість часу науковці даних віддають перевагу пропозиціям системи перед повною

автоматизацією. Навпаки, доменні експерти віддають перевагу повній автоматизації на більшості етапів, що включають кодування. Це представляє різні точки зору щодо автоматизації в проектах DS/ML і показує, наскільки практики хочуть автоматизації цих проектів.

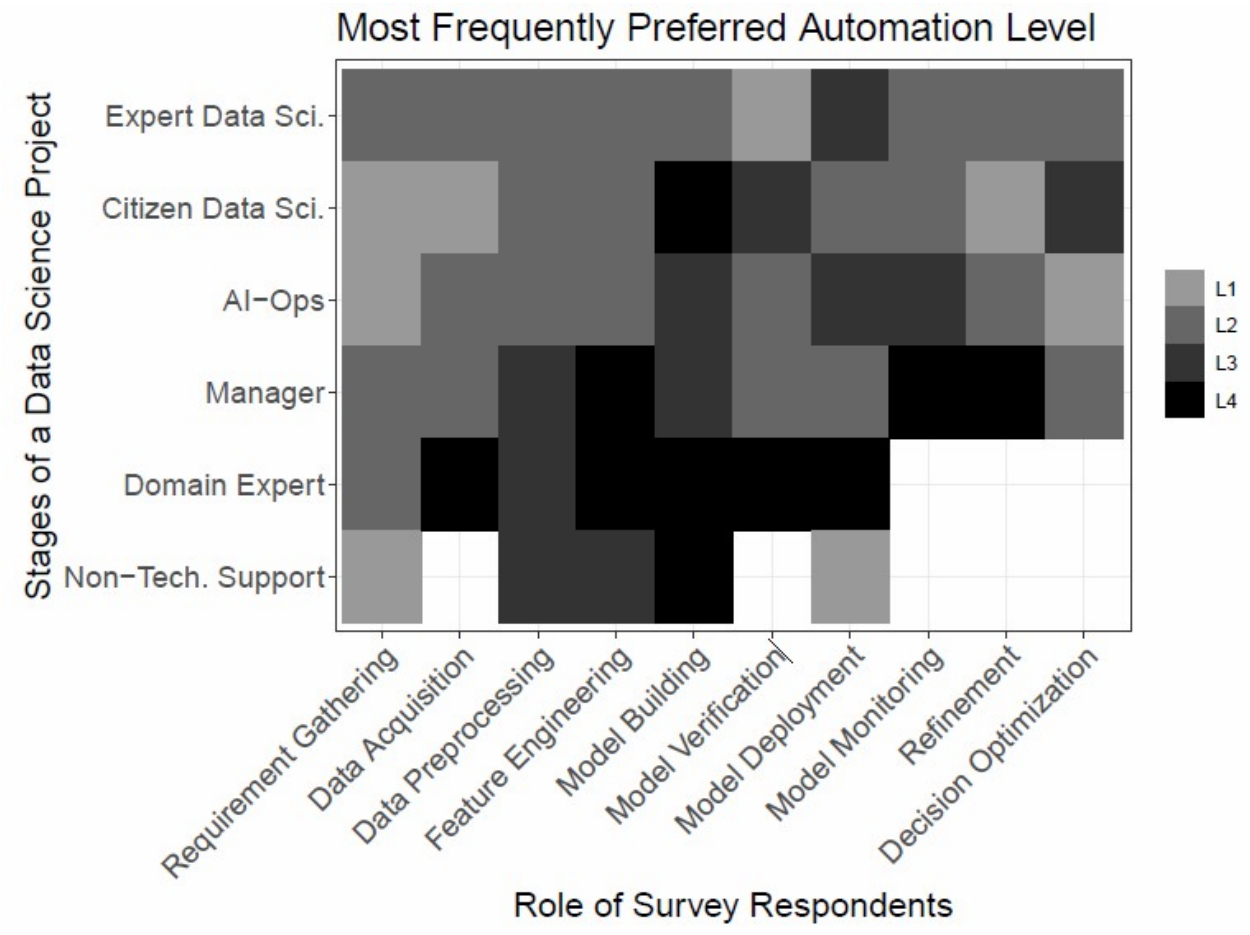


Рис. 1.1. Бажані рівні автоматизації на кожному етапі проекту DS/ML

Щоб автоматизувати різні етапи процесу DS/ML, нам потрібно визначити різні рівні автоматизації, які ми надамо науковцям даних. Є кілька спроб описати та формалізувати профіль науковця пов’язують використання терміна "наука про дані" з описом різних профілів. Крім того, вони представляють використання терміна для приховування очікувань від ролі. Тому автори назвали науковця даних загальним поняттям, що охоплює традиційні області, такі як математика та комп’ютерна наука, серед інших.

Такі області мають чіткі очікування, але всі разом створюють плутанину. Щоб краще визначити очікування науковців даних, автори провели онлайн-опитування, щоб зібрати дані про необхідні або бажані навички. Вони узагальнили 22 навички, розділені на п’ять категорій: Бізнес, ML, Математика, Програмування та Статистика. Автори згрупували результати в чотири профілі: Бізнес-аналітики даних, Креативники даних, Розробники даних та Дослідники даних.

Бізнес-аналітики даних зосереджуються на організації та бізнес-цінності через аналіз даних. Вони бачать себе лідерами або радниками. Креативники даних зосереджуються на широкому наборі інструментів для вирішення проблеми. Вони мають академічний та професійний досвід і бачать себе як митці або хакери. Більшість учасників зробили внесок у проекти з відкритим кодом. Розробники даних зосереджуються на технічних проблемах, пов’язаних з даними, таких як: де їх зберігати та як їх витягувати. Вони є безумовно групою з найбільшими навичками кодування, з дипломами з комп’ютерних наук або інженерії. Група Дослідників даних об’єднує людей з академічними публікаціями, і більшість з них мають ступінь PhD. Вони зосереджуються на використанні даних для розуміння складних процесів.

В [9] представляють концептуальну модель профілю науковця даних, поділяючи характеристики науковців даних на дві групи: базу знань та набір навичок. База знань містить формальні вимоги, такі як безпека, етика даних, інформаційні системи та управління даними, серед інших. З іншого боку, набір навичок зосереджується на завданнях, які повинен виконувати науковець даних, таких як візуалізація, збір даних, статистичні методи, щоб назвати декілька.

Автори визначають науковців даних як здатних витягувати закономірності з даних незалежно від складнощів, спілкуватися про свої результати, генерувати артефакти даних та покращувати процеси прийняття рішень. Ми можемо спостерігати, що об’єднання чотирьох груп, описаних в [10], доповнює концептуальну модель.

В [11] автор оглядає необхідні навички науковців даних, враховуючи три категорії: новачок, учень та експерт. Новачки-науковці даних можуть покращити результати чітко визначеної проблеми науки даних. Учні повинні мати справу зі збором і обробкою даних і можуть відповідати на гіпотетичні запитання про дані. У свою чергу, група експертів повинна мати справу з погано описаними ситуаціями та пропонувати нові напрямки дослідження. Від них очікується, що вони постійно вдосконалюватимуть свої моделі за допомогою відгуків користувачів і використовуватимуть знання з попереднього досвіду. Профілі, тут описані, мають міцний теоретичний фундамент, доповнений знаннями з попереднього досвіду. Ці категорії доповнюють попередні роботи, роблячи висновок, що науковці даних повинні мати як теоретичні знання, так і практичний досвід.

Ми передбачаємо використання персональних помічників для автоматизації частин проектів DS/ML. Автор [12] наводить приклад використання розмовних інтерфейсів для створення персональних помічників у галузі науки про дані. Однією з головних переваг, на яку вони вказують, є продуктивність, і їхні експерименти показали, що користувачі, які використовують помічників з науки про дані, виконують певні завдання з науки про дані в 2,6 рази швидше, ніж за допомогою сценаріїв Python. Досвідчені та початківці користувачі виділяють різні переваги: досвідчені користувачі помічають, що використання помічників може економити час як обгортки функцій Python, а початківці користувачі використовують структурний супровід, наданий персональним помічником. Однак, щоб досягти всіх бажаних рівнів автоматизації, ми повинні розширити розмовні інтерфейси здатністю пропонувати подальші кроки (L2 system suggestions). Пропозиції чат-бота можуть відрізнятися залежно від ситуації.

Наприклад, початківці користувачі можуть потребувати більше керівництва, тоді як досвідчені користувачі можуть бажати створювати персоналізовані конвеєри виконання. У будь-якому випадку, розмовний

інтерфейс повинен адаптувати свої пропозиції відповідно до потреб користувача.

В [12] основною метою дослідження є вивчення того, про що запитують розробники чат-ботів. Для досягнення цієї мети автори вдаються до аналізу дискусій розробників на Stack Overflow, оскільки він забезпечує багатий набір даних і використовувався в подібних дослідженнях в інших областях, таких як паралельне програмування [6], криптографічні API та машинне навчання.

Хоча Stack Overflow надає структуровані дані з питаннями, відповідями та їх відповідними метаданими (наприклад, прийняті відповіді), він не містить жодної детальної інформації про тему, пов’язану з чат-ботами. Тому спочатку нам потрібно ідентифікувати пости зі Stack Overflow, які пов’язані з чат-ботами, згрупувати їх за їхньою домінуючою темою, а потім провести наш аналіз.

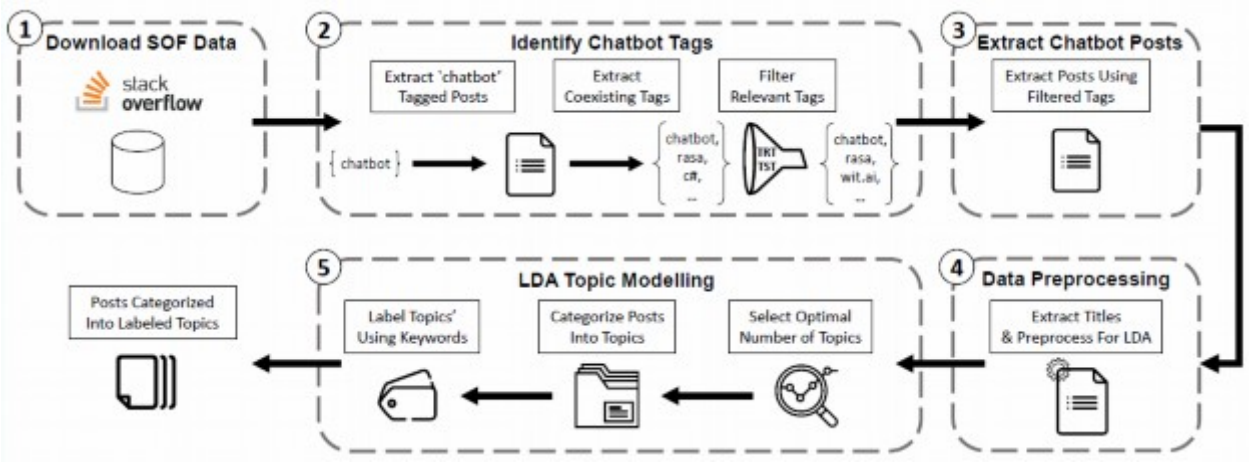


Рис. 1.2. Огляд методології дослідження розмовних інтерфейсів

Як показано на рисунку 1.2, автори виконують вибір постів, пов’язаних з чат-ботами, за п’ятикроковою методологією.

Недавні дослідження розмовних інтерфейсів підняли питання персоналізованої взаємодії [13]. У більшості випадків автори згадують персоналізований контент як засіб підвищення загального задоволення

користувачів. Однак це спосіб надання конкретного контенту для різних груп користувачів. В [14] виділяють два обмеження: розуміння контексту та генерація відповідної відповіді. Їхня робота є більш конкретною щодо цих обмежень і стверджує, що боти повинні відповідати з емоціями або/і персоналізованим контентом. Вони показують, як розробники мають реальні проблеми з розумінням природної мови, зокрема, здатністю відповідати. На думку авторів, це має реальний вплив на задоволення користувачів. Тому здатність розуміти є вирішальною, але правильна відповідь також є критично важливою.

В [14] наводять ще один приклад, висвітлюючи проблеми лікаря та пацієнта, випадки використання та обмеження використання чат-ботів у медичних середовищах. Однак такі обмеження можна узагальнити як здатність працювати з різними групами користувачів. У своєму конкретному випадку автори посилаються на конфіденційну інформацію та конфіденційність лікар-пацієнт. У більш загальному сенсі автори посилаються на раніше представлену роботу про необхідність надання персоналізованого контенту для різних груп користувачів.

1.2. Визначення проблемної області дослідження

Зі зростанням попиту на інтелектуальні програмні рішення сфера обробки даних отримала величезне визнання. Процеси Data Science є складними та трудомісткими, і їм не вистачає кваліфікованих спеціалістів, які б задовольнили потреби роботи. Складність і відсутність кваліфікованих фахівців викликають потребу в певному ступені автоматизації. У цій роботі ми розглядаємо актуальну потребу в автоматизації процедур DS/ML. Розмовні інтерфейси з’являються як рішення для автоматизації різних процесів і взаємодії з технологіями за допомогою природної мови. Ми віримо, що персоналізований вміст у бічних розмовних інтерфейсах полегшить автоматизацію, яку запитують фахівці з даних на деяких етапах

DS/ML. Зокрема, наша мета полягає в тому, щоб запропонувати структуру для автоматизації процесів DS/ML через розмовні інтерфейси з особою, ¬яка виконує дії.

Для досягнення нашої мети ми вирішуємо такі питання:

1. Чи можемо ми використовувати персоналізовану взаємодію в розмовних інтерфейсах, щоб досягти рівня автоматизації системних пропозицій (L2) всередині основних функцій інфраструктури?

2. Чи зможуть розробники даних розширити запропоновану структуру для конкретних сценаріїв?

1.3. Представлення методології дослідження

На рисунку 1.3 ми представляємо наш план відповіді на запитання дослідження.

По-перше, ми пропонуємо огляд літератури про персоналізовані відповіді на вміст і чат-ботів. Після огляду літератури ми створимо структуру макросу. Ця початкова архітектура може бути скоригована в ході реалізації. Архітектура повинна включати основні функції фреймворку та гарячі точки. Ми також плануємо інтегрувати наш розмовний агент з інтерфейсами користувача. Нарешті, ми створимо нашу структуру для деяких сценаріїв DS і проведемо дослідження, щоб перевірити, чи можуть інші фахівці з обробки даних розширити запропоноване рішення.

Наскільки нам відомо, ніхто не пропонував використовувати розмовні інтерфейси з персоналізованою взаємодією для досягнення рівня автоматизації L2. Ми використовуємо розмовні інтерфейси, щоб взаємодіяти з фахівцем з обробки даних, і застосовуємо стратегію, щоб навчатися з історії користувача та персоналізувати взаємодію. Зокрема, ми зосереджуємося на пропозиціях, які користувач отримує від чат-бота. Ми вдосконалюємо попередні пов’язані роботи, такі як [15, 16]. Автори представляють функцію генерації коду та ієрархію команд, щоб допомогти дослідникам даних. Ми

розширюємо їхній підхід і включаємо конвеєри та інші нові функції, такі як підтримка деяких завдань automl, відстеження історії діалогів користувача та персоналізовані пропозиції.

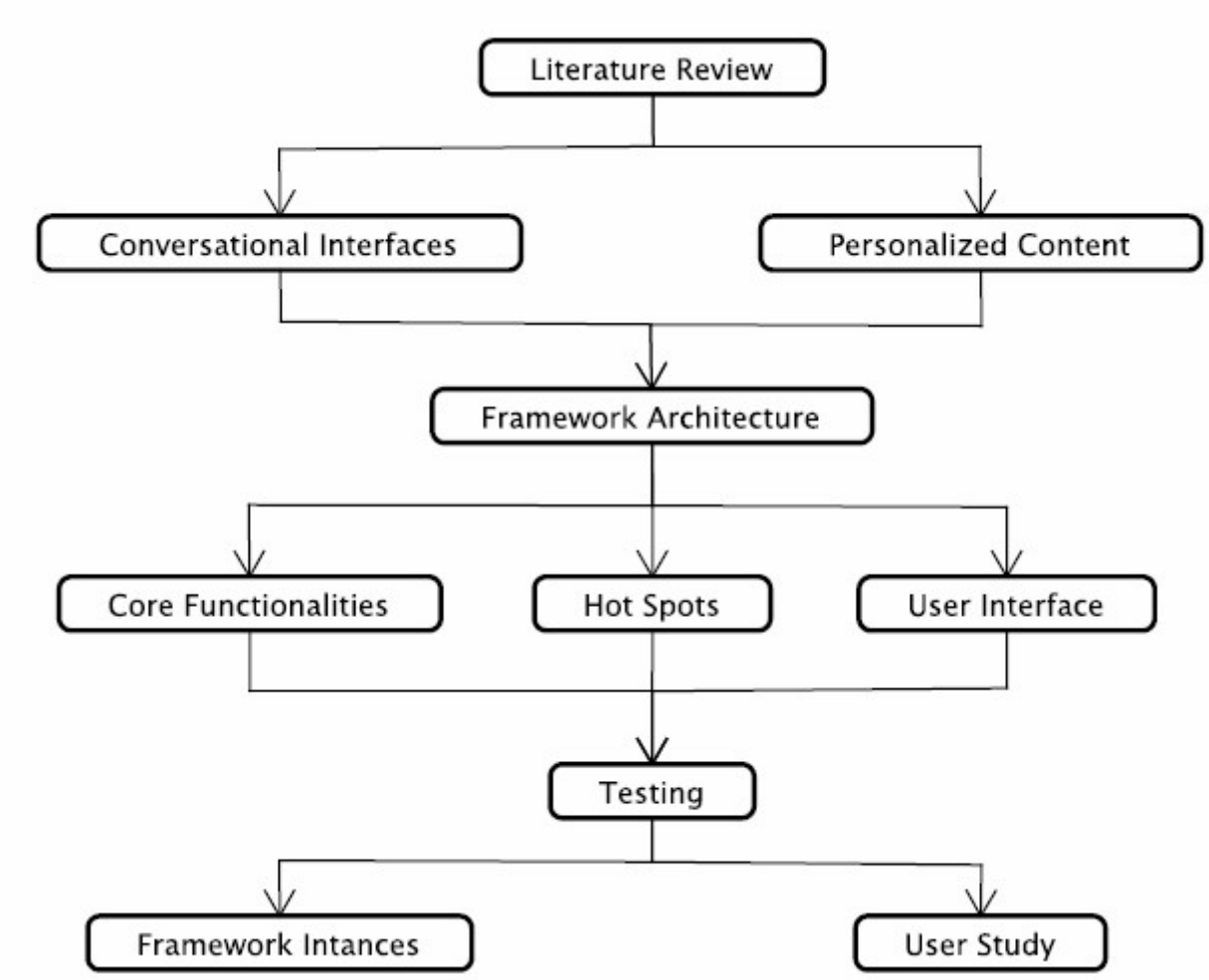


Рис. 1.3. Методологія дослідження

1.4. Загальні концепції побудови чат-ботів як програмного агента

Цей розділ представляє основні концепції, використані в цій роботі. Тут ми надамо загальне визначення систем чат-ботів.

Систему чат-ботів можна визначити як програмного агента, який використовує природну мову для надання доступу до даних і послуг. Чат-боти широко використовуються в різних сценаріях, таких як охорона здоров’я, електронна комерція та підтримка клієнтів.

Згідно з [19], чат-боти можна охарактеризувати за двома вимірами: контролем діалогу та тривалістю відносин.

Контроль розмови визначає, хто контролює діалог, і може бути одним із таких варіантів:

- Діалог, керований чат-ботом: У цій категорії ми знаходимо ботів із попередньо визначеною взаємодією. У цьому випадку діалог певною мірою вимушений чат-ботом.

- Діалог, керований користувачем: Цей тип бота дозволяє більш гнучкі розмови і адаптується до варіацій вводу користувача. У цьому випадку розмова контролюється користувачем до певного рівня.

Іншою характеристикою чат-ботів є тривалість відносин з користувачем.

Короткострокові відносини: Цей тип ботів зосереджується на наданні взаємодії без стійких відносин.

Довгострокові відносини: У цій категорії боти схильні витягувати інформацію з попередніх розмов і використовувати її у подальшій взаємодії з користувачами.

У нашому випадку ми пропонуємо проект із керованим чат-ботом і довгостроковими відносинами. Взаємодії обмежені певною областю та синтаксичними обмеженнями. Тому бот буде керувати взаємодією. З іншого боку, відносини між ботом і користувачем уточнюються разом із кількома розмовами.

1.5. Поняття автоматизованого машинного навчання (AutoML)

AutoML можна визначити як автоматизацію або часткову автоматизацію конвейєра машинного навчання. Більшість конвейєрів машинного навчання містять етапи підготовки даних, інженерії ознак, генерації моделей та оцінки моделей. Однак у [20] автор розширює конвейєр більш складним баченням автоматизації та пропонує інші пов’язані завдання,

такі як автоматичне генерування звітів, підбір проблем до алгоритмів, передавальне навчання тощо. Ці доповнення забезпечують більш широке розуміння поля діяльності та можливостей, які з'явилися разом із викликом AutoML.

Іноді AutoML визначається як максимізація продуктивності інструменту навчання з меншою людською допомогою та обмеженим обчислювальним бюджетом. Тому це стала гарячою темою в галузі, і в Інтернеті доступно кілька інструментів, таких як AutoWeka, TPOT, Auto-Pytorch тощо.

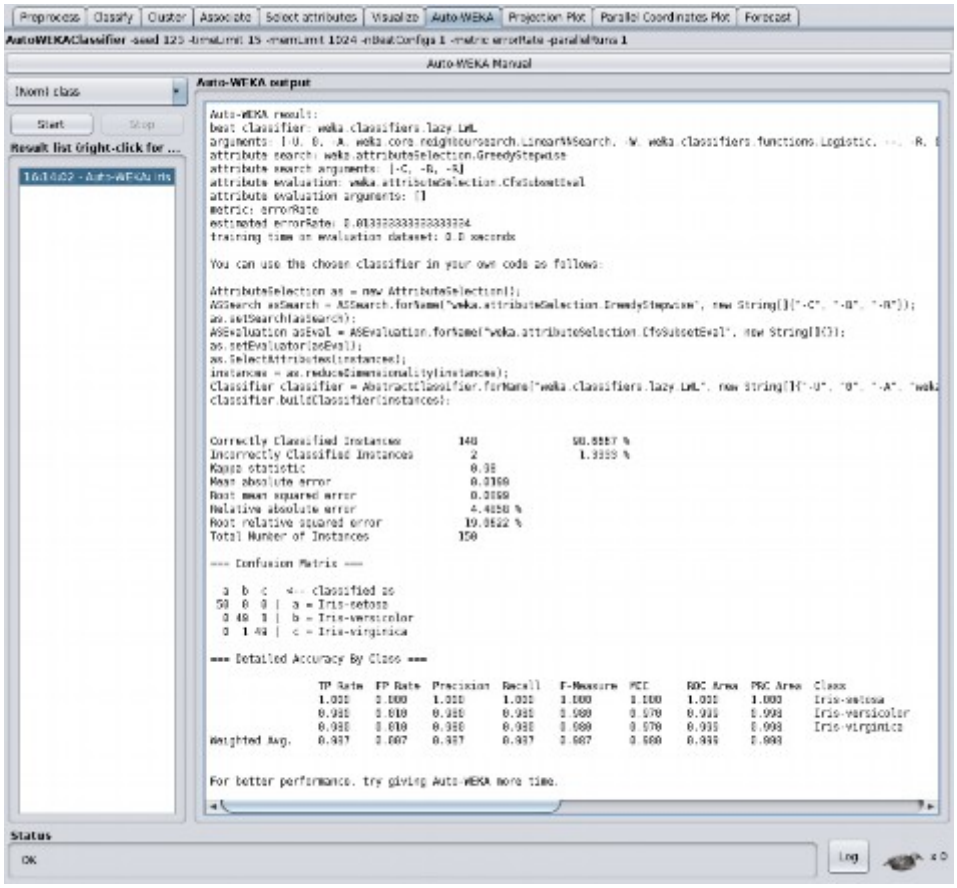


Рис. 1.4. Приклад запуску Auto-WEKA на наборі даних iris

Найкращий класифікатор разом з його налаштуваннями параметрів друкується спочатку, потім його продуктивність. Під час роботи Auto-WEKA він реєструє в рядку стану, скільки конфігурацій він оцінив до цього моменту.

Основний конвейєр, описаний у [21], містить чотири основні кроки.

Крок підготовки даних зосереджений на обробці даних. Типовими завданнями на цьому кроці є нормалізація, очищення, заповнення пропущених значень тощо. Цей крок готує дані для подальшої обробки. На кроці інженерії ознак метою є перетворення даних на кінцевий використовуваний набір даних шляхом застосування алгоритмів зниження розмірності, алгоритмів важливості ознак, методів аргументації даних тощо. Зверніть увагу, що на першому кроці кількість ознак залишається такою ж. Однак на цьому кроці кількість ознак можна зменшити за допомогою алгоритмів зниження розмірності або збільшити шляхом генерації нових ознак. Деякі стратегії для генерації нових ознак - це перетворення даних на складніші уявлення. Наприклад, алгебраїчні лінійні застосування можуть перетворити простір Rn на Rm. Крок генерації моделей містить відображення даних на алгоритм і налаштування гіперпараметрів.

На сьогоднішній день існує велика різноманітність алгоритмів, і кожен з них може мати безліч параметрів. Більшість традиційних алгоритмів, таких як дерева рішень і SVM, мають обмежену кількість гіперпараметрів. Однак сучасні нейронні мережі можуть мати кілька шарів з різною архітектурою. Автоматичне налаштування архітектури нейронних мереж може мати справу з мільйонами гіперпараметрів, як у випадку VGG-16 з більш ніж 130 мільйонами параметрів. Нарешті, крок оцінки моделі визначив якість згенерованих моделей. Однак навчання кількох моделей може бути тривалим завданням, і існує кілька стратегій для покращення оцінки моделей.

Наприклад, раннє зупинення - це стратегія, яка використовується для уникнення перенавчання в традиційних методах, а ресурсно-обізнана - це стратегія для моніторингу споживання ресурсів (пам’ять, процесор і GPU) та компенсації іншими метриками, такими як точність, точність, відкликання та f1-метрика. У цьому випадку, навіть якщо модель може втратити продуктивність, вона може значно заощадити ресурси.

Висновки до розділу

У цьому розділі проведено детальний аналіз стану автоматизації процесів у сфері Data Science, окреслено проблемну область дослідження, визначено методологічні засади та розглянуто ключові концепції, що стосуються автоматизації рішень.

Аналіз сучасного стану показав, що процеси Data Science все більше потребують автоматизації через постійно зростаючі обсяги даних, складність моделей та необхідність швидкого отримання результатів. Хоча існують численні інструменти та фреймворки, їх використання часто вимагає значних знань і навичок, що обмежує доступність Data Science для ширшої аудиторії.

Проблемна область дослідження була визначена як необхідність розробки універсальних рішень, здатних автоматизувати ключові етапи процесу Data Science, включаючи підготовку даних, вибір моделей, налаштування гіперпараметрів і оцінку якості. Особливу увагу приділено інтеграції таких рішень у програмних агентів, зокрема чат-ботів, для забезпечення зручності взаємодії з користувачем.

Розглянута методологія дослідження пропонує комплексний підхід до створення автоматизованих рішень, що поєднує теоретичний аналіз, розробку прототипів і емпіричну оцінку ефективності. Цей підхід гарантує адаптивність і практичну спрямованість дослідження.Загальні концепції побудови чат-ботів показали їх потенціал як інтерфейсу для взаємодії з системами автоматизації. Чат-боти можуть ефективно виступати як програмні агенти, що допомагають користувачам отримувати необхідні результати без потреби в глибоких технічних знаннях.

Автоматизоване машинне навчання (AutoML) було виділено як одна з ключових концепцій, що дозволяє спрощувати побудову моделей і підвищувати продуктивність Data Science. Впровадження AutoML у системи автоматизації відкриває нові можливості для оптимізації роботи, дозволяючи фахівцям зосередитися на вирішенні складніших задач.

РОЗДІЛ 2. ПІДХОДИ ГЕНЕРУВАННЯ ПЕРСОНАЛІЗОВАНОГО КОНТЕНТУ ЗАСОБАМИ РОЗМОВНИХ ІНТЕРФЕЙСІВ. ПРЕДСТАВЛЕННЯ ВЛАСНОГО РІШЕННЯ

2.1. Огляд літератури створення концептуальної моделі соціальних характеристик для чат-ботів

Цей розділ представляє роботи, пов’язані з персоналізованим контентом, згенерованим розмовними інтерфейсами.

Автори в [31] пропонують огляд літератури для виведення концептуальної моделі соціальних характеристик для чат-ботів. Автори стверджують, що чат-боти повинні бути збагачені соціальними характеристиками, щоб уникнути незадоволення користувачів. Література, оглянута в цьому огляді, сильно підтримує чат-ботів із соціальними характеристиками. Автори виявили три основні групи соціальних характеристик у чат-ботах (розмовний інтелект, соціальний інтелект і персоніфікація).

Персоналізація розташована всередині групи соціального інтелекту і зосереджена на наданні персоналізованої взаємодії. На думку авторів, основним недоліком персоналізації є конфіденційність даних: система повинна мати особисту інформацію, щоб адаптуватися і надавати персоналізовану взаємодію. Однак особиста інформація користувача є чутливою і вимагає безпеки, що стає проблемою конфіденційності. Персоналізована взаємодія поділяється на три основні переваги. По-перше, збагачення міжособистісних стосунків: користувач повинен контролювати кількість особистої інформації, до якої мають доступ боти. По-друге, надання унікальних послуг. Наприклад, система допомоги в турі може використовувати геолокаційні служби, щоб надати зручно розташовані місця. Нарешті, зменшення збоїв у взаємодії. У цьому випадку інтерфейс

користувача повинен адаптуватися до різних типів користувачів. Наприклад, система може використовувати більший шрифт для літніх людей.

З іншого боку, в [22] автори зосереджуються на переконливих і персоналізованих розмовах чат-ботів. По-перше, автори підкреслюють важливість створення персоналізованої взаємодії між чат-ботами та користувачами. Загалом, для проектування та тестування розмов було обрано три базові типи особистості (комбінація вимірів MBTI) (рис. 2.1). Метою було створити контрастні розмови. Перший базовий тип особистості (pt1) мав баланс між кожним виміром з метою показати базову лінію для порівняння з двома іншими типами особистості. Другий базовий тип особистості (pt2) мав особистість ENF (екстравертний, інтуїтивний і чутливий), що передбачало його перевагу до багатослів'я, низького рівня деталізації, неформальних речень та використання смайликів. Третій базовий тип особистості (pt3) мав особистість IST (інтровертний, сенсорний і мислячий) і, таким чином, передбачалося, що він віддає перевагу низькому рівню багатослів'я, високому рівню деталізації, формальним реченням і відсутності використання смайликів.

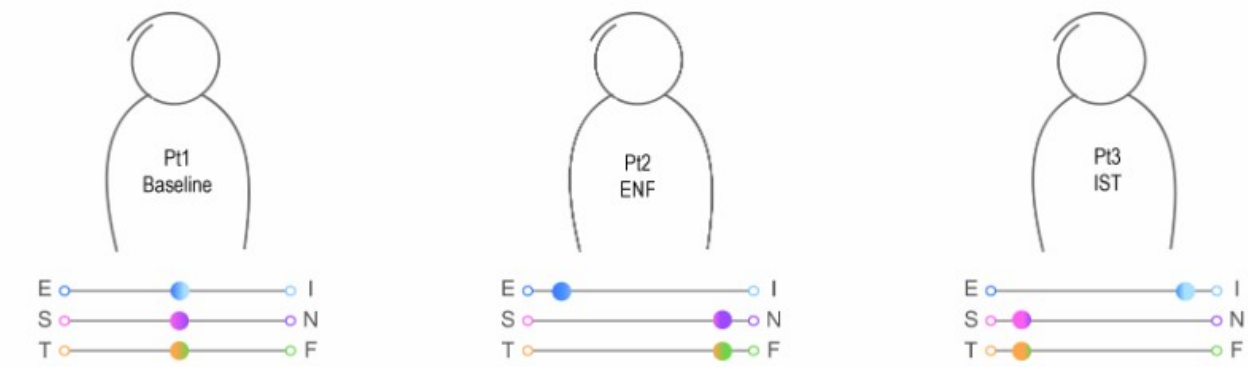


Рис. 2.1. Три обрані типи особистості чат-боту, використані для тестів користувачів на основі індикатора типу Майєрс-Бріггс

Боти, як правило, реалізовані як "однаковий розмір для всіх". Однак автор представляє персоналізований контент як складне завдання. Автор

зосереджує дослідження на сферах водіння та безпеки. Але важливість викладених проблем і рекомендацій застосовується до багатьох областей. Автор представляє три виміри персоналізації: частини системи, цільову аудиторію та автоматизацію. Перший стосується частин системи, які можна або потрібно персоналізувати. Другий стосується цільової аудиторії або групи цілей, на яку спрямована персоналізація. Нарешті, останній вимір стосується автоматизації персоналізації або того, чи вона вимагає ручних дій. Автори представляють остаточну версію рекомендацій. Рекомендації, як стверджує автор, повинні допомогти в проектуванні переконливих і персоналізованих чат-ботів. На думку авторів, персоналізація є важливою частиною дизайну чат-бота і забезпечує індивідуальний досвід користувача.

В [23] автори зосереджуються на представленні потреб і проблем, що виникають у галузі чат-ботів. Автори представляють мотивацію до майбутнього використання чат-ботів з точки зору користувача. По-перше, автори стверджують, що чат-боти є не тільки зміною в інтерфейсі з технологією, але й зміною в динаміці та способах використання технології. Це означає, що боти є не тільки новими способами доступу до технології такою, якою вона є, але й технологія повинна рухатися вперед і адаптуватися до нових тенденцій, що виникають разом із використанням чат-ботів. Для досягнення цієї мети автор показує необхідність покращення моделей користувачів і контексту. На думку авторів, користувачі повинні відчувати себе під контролем технології. Це означає можливість швидко і ефективно виконувати завдання. Нарешті, автор наводить важливий урок, винесений з їхнього дослідження: чат-боти не є єдиним рішенням для всіх користувачів, оскільки мотивація та цілі людей різноманітні. Замість "одного рішення для всіх" автор пропонує кілька варіантів використання в контексті. Це тісно пов’язано з відповіддю на персоналізований контент, оскільки це показує необхідність правильно поводитися в різних сценаріях.

В дослідженні [24] автори представляють порівняння того, як користувачі взаємодіють з емоційно експресивним ботом проти нейтрального

бота. Автори розробляють і реалізують чат-бота, чутливого до емоцій, який відповідає емоційно відповідними розмовами. Автори дійшли висновку, що учасники виявили більше позитивних емоцій при спілкуванні з емоційним ботом. Автори додали смайлики, щоб пом'якшити розмови. Автор провів дослідження з 39 учасниками та визначив кілька можливих рекомендацій для майбутніх робіт. Одним із обмежень роботи є те, що автори написали сценарій усіх текстових взаємодій, і зрештою відповідь бота стала передбачуваною. Таким чином, автори пропонують використовувати методи машинного навчання для генерації автоматичних відповідей, посилених почуттями та емоціями. Це приклад персоналізованих відповідей на контент. У цьому випадку автори використовують емоційні компоненти для визначення персоналізованої відповіді для кожної емоції, включеної в дослідження. Ми можемо відобразити емоцію на групу користувачів. Таким чином, система генерує персоналізовані відповіді для різних груп користувачів.

У роботі [25] представлена рамка для проектування орієнтованих на користувача чат-ботів з особистістю та впливу особистості на враження користувача під час взаємодії з чат-ботами. Мотивацією роботи було дослідити, наскільки особистість впливає на сприйняття користувачем чат-ботів. Автор показує, що два чат-боти, що пропонують однакові послуги та ефективність, можуть оцінюватися по-різному на основі прагматичних якостей. Пропонована рамка забезпечує стабільний шлях для створення прототипів чат-ботів з особистістю відповідно до теорії особистості. Автори провели експеримент, щоб оцінити вплив особистості на враження користувача, і дійшли висновку, що вона значно покращує враження користувача. Як майбутню роботу автори пропонують розширити рамку та включити аналізатор тону, який може визначити настрій вводу користувача та динамічно змінювати відповідь. Аналізатор тону може скористатися перевагами рамки для генерації відповіді відповідно до особистості. Ця

майбутня робота є прикладом персоналізованих відповідей на контент, у якому бот генеруватиме різні відповіді для різних настроїв.

Якщо чат-бот має сприйматися як продовження або розширення бренду, він повинен дотримуватися тону мови бренду. Тон мови визначається як особистість бренду, що передається через його письмову комунікацію, а також візуальну комунікацію. Автори визначили рамки для визначення тону мови на основі 4 вимірів: смішний проти серйозного, формальний проти неформального, шанобливий проти нешанобливого, ентузіастичний проти фактичний.

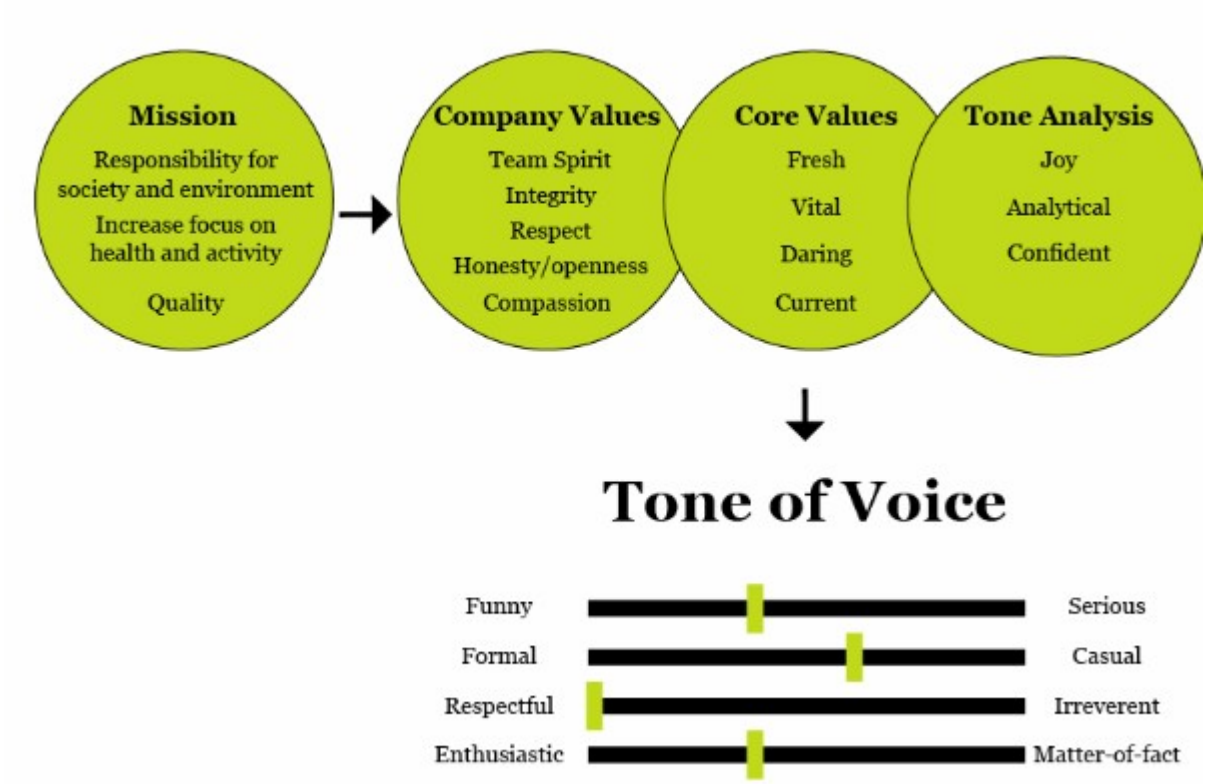


Рис. 1.2. Представлення тону мови, що був визначений шляхом розуміння місії, цінностей та проведення аналізу тону письмового тексту

В [26] автори були мотивовані відсутністю соціальної компетентності агентів розмови та тим, які характеристики можуть сприяти автентичності чат-ботів у майбутніх реалізаціях. Автори провели серію інтерв'ю з експертами з різних областей, щоб дійти консенсусного визначення

автентичності в агентах розмови. Основним внеском роботи є теоретичне визначення автентичності шляхом злиття концепцій, знайдених у штучному інтелекті та соціальній етиці. Автор робить висновок, що існує кілька характеристик, які сприяють автентичності, таких як прозора мета, навчання на досвіді, демонстрація сильної розмовної поведінки тощо. Однак автори погодилися, що головним моментом є навчання на досвіді. Навчаючись на досвіді, автор має на увазі відстеження розмов, поведінки та когерентність. Автори стверджують, що надання персоналізованої інформації може зміцнити стосунки з користувачем і підтримувати залученість користувача.

2.2. Огляд основних функцій та архітектури пропонованого рішення

Запропоноване рішення розвивалося через ітерації планування та впровадження. На рисунку 2.3 показано, як ми додаємо функції до нашого рішення з часом. По-перше, ми створили модуль, здатний ідентифікувати наміри користувача: під час розмови система відображає введений текст із заздалегідь визначеним набором намірів.

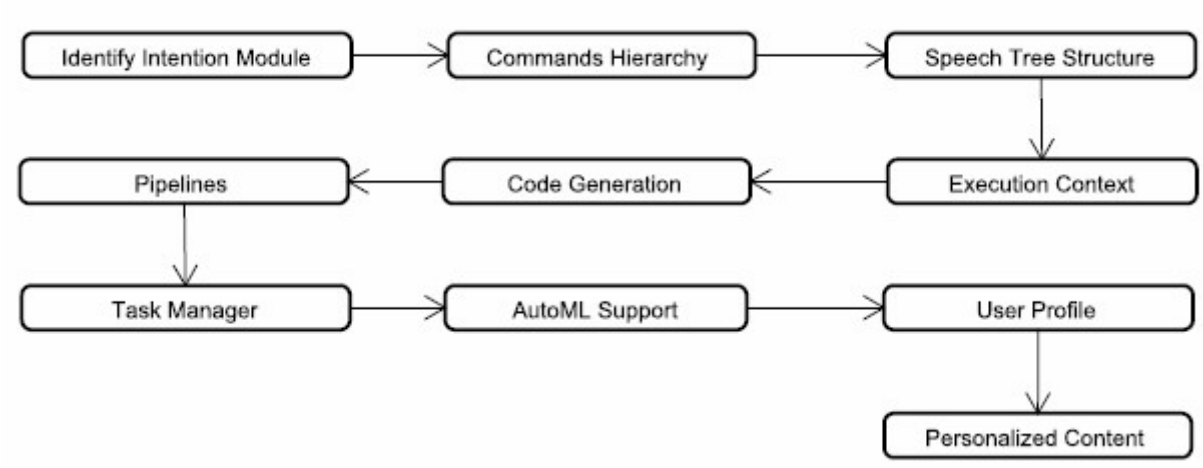


Рис. 2.3. Пропонований функціонал

Модуль ієрархії команд збирає базові класи та основні функції кожної команди. Деревоподібна структура мовлення зберігає розмову з

користувачем і перетворює розмову на деревоподібну структуру даних. Контекстна функція зберігає всі змінні під час виконання команд. Генератор коду експортує всі команди в придатний для використання код Python, а конвеєри об’єднують кілька команд в одну. Диспетчер завдань відокремлює потік виконання команд від інтерфейсу користувача. Пропоноване рішення підтримує функції autoML для покращення використання конвеєра. Система створює профіль користувача за введеними користувачами. Нарешті, модуль персоналізованого контенту використовує інформацію, зібрану в профілі користувача, для надання пропозицій.

Пропонована система складається з чотирьох модулів (Command Manager, Speech Manager, ChatBot і User Profile). На рисунку 2.4 показано, як функції організовані в модулях.

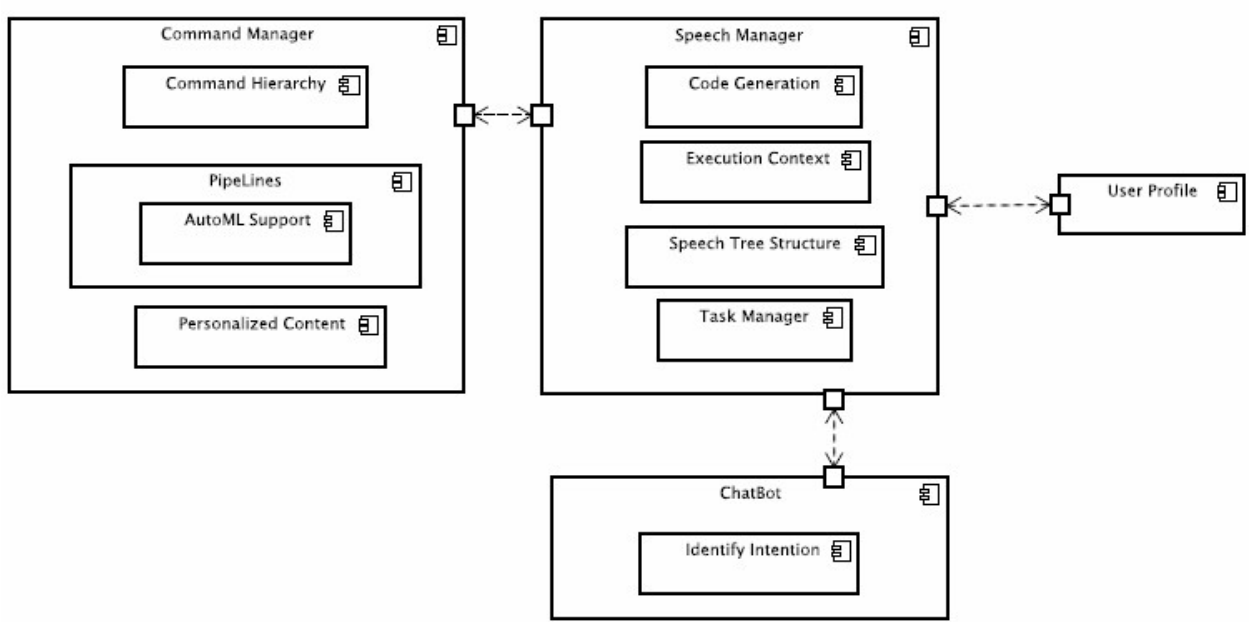


Рис. 2.4. Архітектура запропонованого рішення

Модуль «Command Manager» містить ієрархію команд, конвеєри з підтримкою AutoML і функції персоналізованого вмісту. Модуль ChatBot містить функцію визначення наміру, але модуль профілю користувача є окремою функцією. Нарешті, Speech Manager включає функції генерації коду, контексту виконання, структуру дерева мовлення та функції диспетчера

завдань. Модуль Speech Manager з’єднує всі інші модулі та працює як командний центр і точка входу в рішення, як показано на рисунку 2.5.

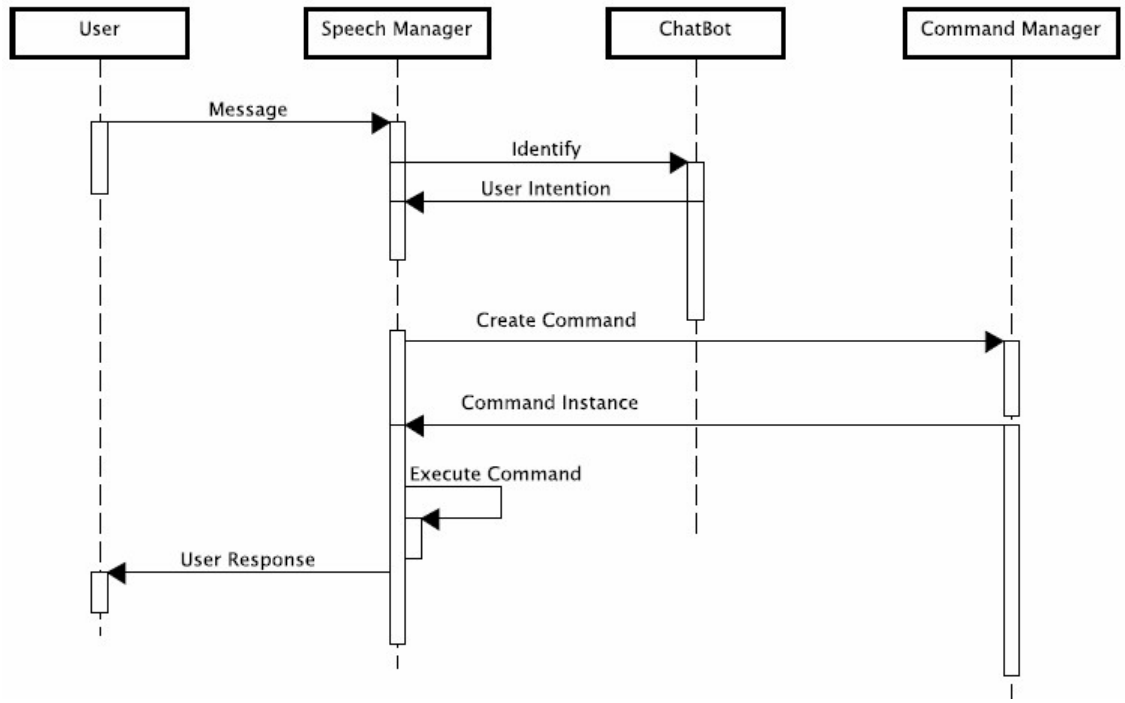


Рис. 2.5. Послідовність подій, ініційованих після нового повідомлення користувача

Speech Manager отримує повідомлення користувача. Повідомлення надсилається чат-боту, щоб визначити намір користувача. За бажанням користувача Speech Manager викликає Command Manager для створення екземпляра вказаної команди та організації його в структурі дерева. Нарешті, Speech Manager виконує команду, щоб повернути правильну відповідь користувачеві. Виконання команд асинхронно обробляється в диспетчері мовлення за допомогою функції диспетчера завдань, тому кожна послідовність подій є незалежною.

2.3. Деталізація реалізації модулів чат-боту

У цьому розділі ми збираємося представити труднощі впровадження, які ми пройшли під час розробки кожної функції.

2.3.1. Ідентифікація наміру бота

Модуль ChatBot містить функцію ідентифікації наміру. Він відповідає за розпізнавання того, чого бажає користувач з кожним введенням. Ми визначаємо проблему ідентифікації наміру користувача як проблему класифікації з навчанням з учителем.

При запуску системи модуль чат-бота створює набір даних фраз і відображає кожну фразу на один намір. Наприклад, речення "Давайте завантажимо набір даних" буде відображено на намір load_dataset. У нас є скінченна кількість намірів, по одному для кожної команди, доступної в чат-боті. Отже, зі зростанням кількості команд зростає і складність цього завдання.

Рисунок 2.6 показує, як був побудований набір даних намірів. x представляє навчальні дані, а y містить мітки.

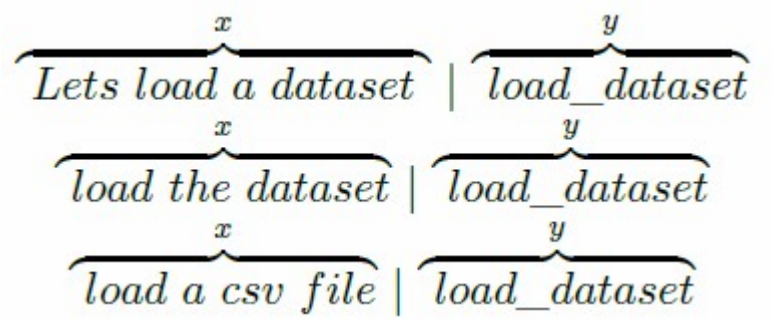


Рис. 2.6. Приклад набору фраз

Ми попередньо обробляємо наш набір даних фраз, щоб побудувати класифікатор і виявити намір користувача, як показано на рисунку 2.6. Спочатку ми застосовуємо стратегію токенізації, щоб розділити речення на вектори. Потім ми видаляємо стоп-слова та застосовуємо орфографічну корекцію. Однак у деяких сценаріях, таких як системні шляхи або посилання на набори даних, ми повинні уникати орфографічної корекції.

Далі ми створюємо глобальний вектор слів і перетворюємо кожне речення, встановлюючи одиниці індексів його слів.

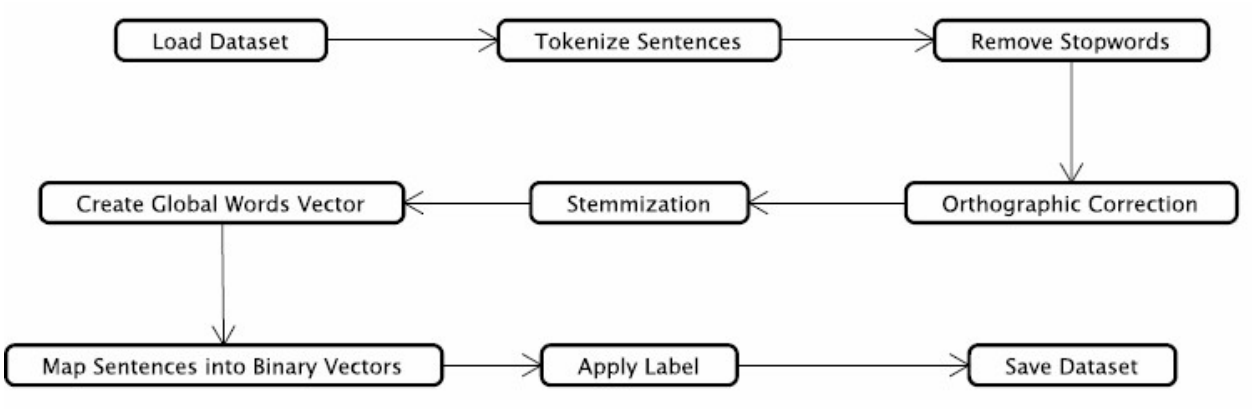
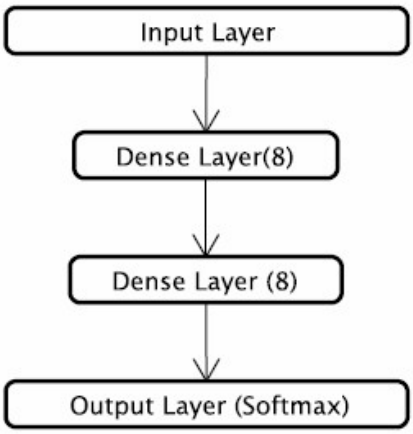


Рис. 2.7. Послідовність етапів попередньої обробки набору даних

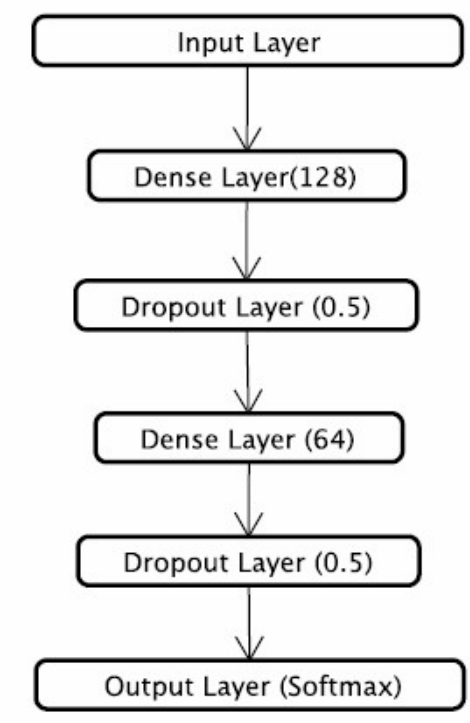
Отже, ми створюємо вектор нулів довжиною глобального вектора слів і змінюємо на одиницю позицію, що відповідає словам речення. Таким чином, ми представляємо речення як двійковий вектор. Потім ми зіставляємо кожен вектор з його класифікацією у і зберігаємо набір даних для подальшого вивчення. Для навчання класифікаторів ми використовуємо моделі нейронних мереж, описані на рисунках 2.8 а та 2.8 б.

На рисунку 2.8 а показана перша спроба класифікувати наміри користувача. Ця модель була придатна для використання на перших етапах проекту, але оскільки кількість команд зростала, нам довелося перейти на нашу другу модель, представлену на рисунку 2.8 б. Модуль ChatBot містить навчену модель для класифікації кожного введення користувача. Однак зауважте, що останній шар обох моделей є шаром softmax. Таким чином, класифікатор поверне ймовірність того, що введені користувачем дані належать кожному можливому наміру. Це означає, що класифікатори не повертатимуть намір безпосередньо. Однак бот використовує порогове значення для порівняння ймовірностей softmax і повертає дійсний намір лише тоді, коли будь-яке значення ймовірності перевищує попередньо визначений поріг. Якщо ймовірність не перевищує порогове значення, бот поверне диспетчеру мовлення текст із символом підстановки та стан no_understand . Підводячи підсумок, бот попередньо оброблятиме кожен

запис користувача та перетворюватиме його на дійсне двійкове представлення. Після перетворення він прогнозує ймовірності кожного можливого наміру та, нарешті, порівнює з пороговим значенням, щоб повернути найбільш ймовірний намір. Якщо дві ймовірності перевищують порогове значення та дуже близькі одна до одної, бот поверне більшу ймовірність.



а) Початкова модель класифікації



б) Модель класифікації з випадаючими шарами

Рис. 2.8. Моделі, що використовуються для класифікації

2.3.2. Ієрархія команд

У цьому підрозділі ми покажемо, як ми перетворюємо наміри користувача на команди, пояснюємо, як ми групуємо їх за категоріями, і деталізуємо ієрархію класів за ними.

Існує створений механізм для захоплення тригерів і доповнення базових команд. Наприклад, коли система виявляє новий вхід для використання команди, система додає цей новий вхід до колекції тригерів вибраної команди. Тому DSL і механізм, посилений тригерами, забезпечують базове перетворення з функцій на команди. Отже, ми створюємо початкове відображення функцій на команди та доповнюємо тригери новими входами. Після додавання нової фрази тригера ми повинні перенавчити нашу модель ідентифікації наміру. Рисунок 2.9 показує деякі наміри користувачів, розділені за категоріями.

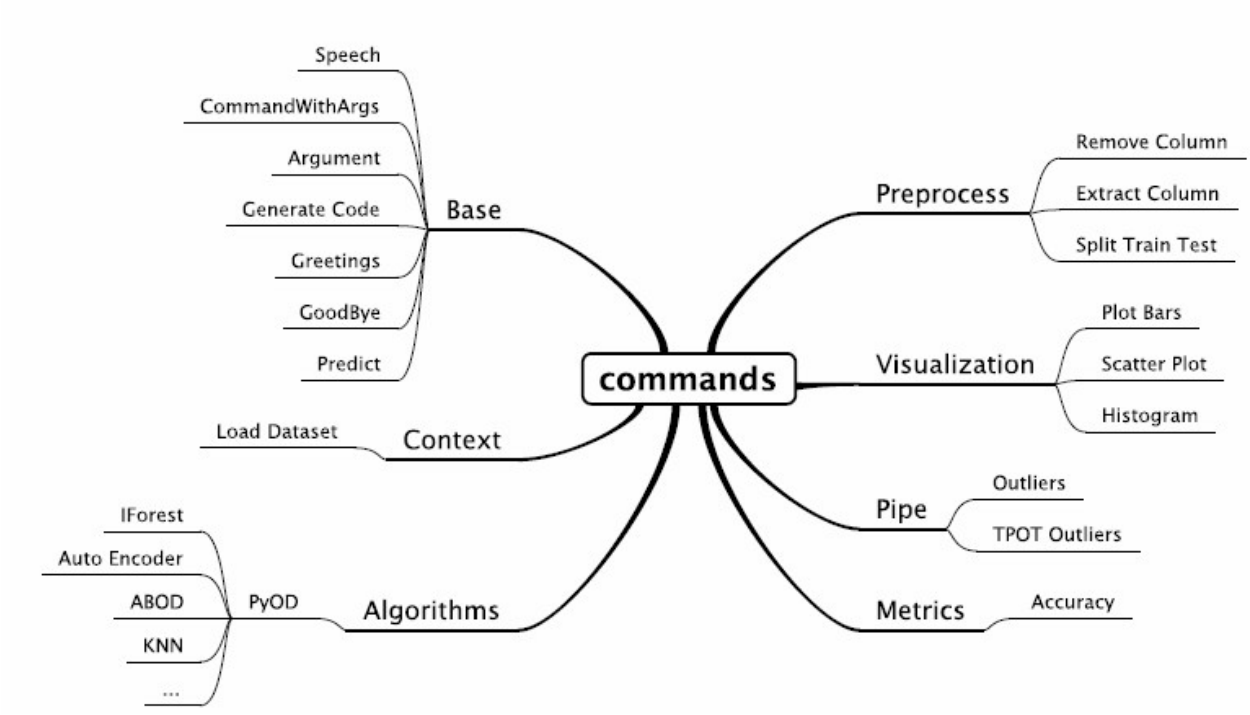


Рис. 2.9. Представлення розподілу команд за групами

Ми створили сім категорій для групування команд: Base, Context, Algorithms, Preprocess, Visualization, Pipes та Metrics. У категорії Base ми

розмістили базові класи ієрархії та команди, пов’язані з базовою розмовою (привітання та прощання). Категорія Context збирає команди, пов’язані з середовищем виконання. Наприклад, LoadDatasetCommand отримує доступ до файлової системи для завантаження файлу CSV. Категорія Algorithms містить реалізації різних алгоритмів, а категорія Metrics має команди для оцінки результатів алгоритму. Категорія Pipe містить наші реалізовані конвеєри, а категорія Visualization представляє команди для генерації діаграм. Нарешті, категорія Preprocess містить інструменти для роботи з наборами даних після етапу завантаження.

Базова категорія містить надкласи ієрархії команд: Command, CommandWithArgs та Argument. Клас Command представляє всі команди, а клас CommandWithArgs представляє всі команди з аргументами. Наприклад, LoadDatasetCommand вимагає двох аргументів: шляху до набору даних і назви. Зверніть увагу, що є кілька команд, які не вимагають жодних параметрів, наприклад GreetingsCommand. Нарешті, клас Argument представляє аргументи команд. Рисунок 2.10 детально описує методи та властивості цих класів.

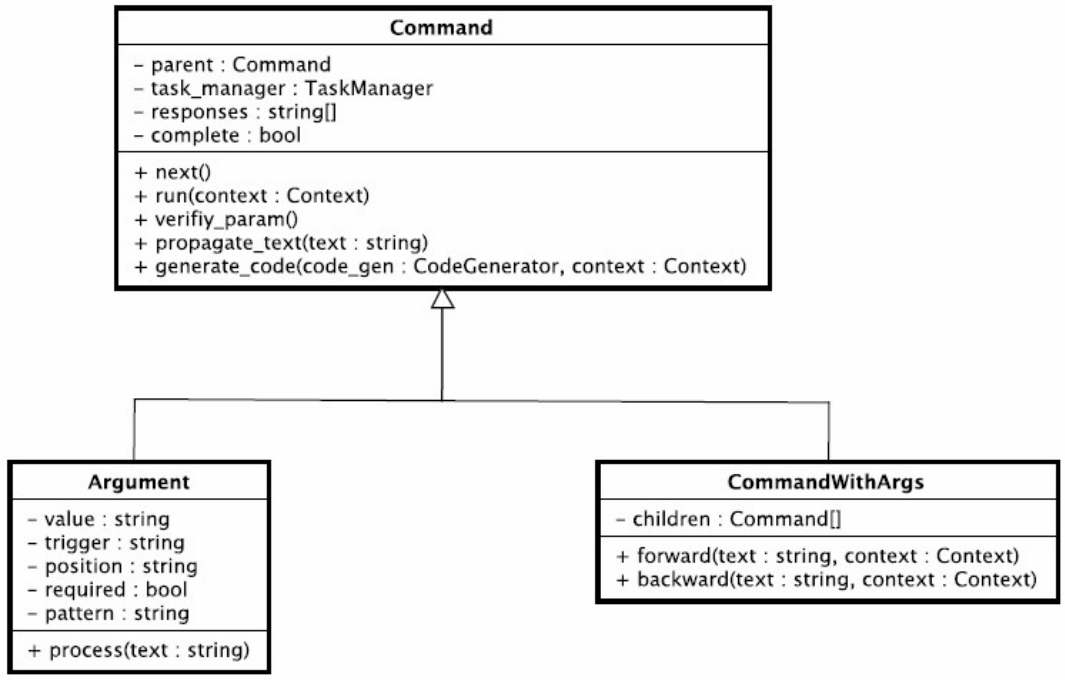


Рис. 2.10. Базові класи ієрархії команд

Зокрема, клас Argument використовує регулярні вирази для захоплення значення аргументів. Рисунок 2.11 показує приклад.

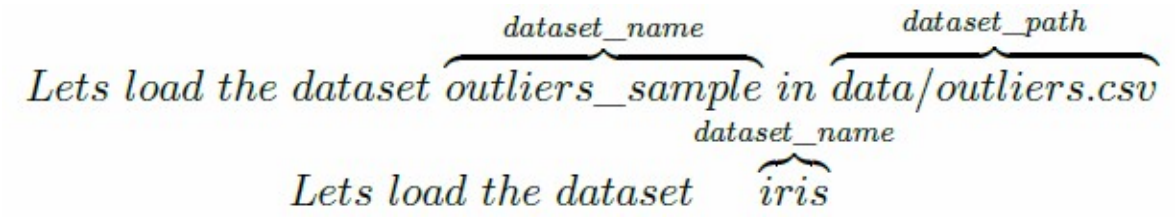


Рис. 2.11. Приклад аргументів у введених користувачами

Повний ввід користувача "Let’s load the dataset outliers_sample in data/outliers.csv" ідентифікується з наміром load_dataset. Система перетворює намір load_dataset на LoadDatasetCommand. LoadDatasetCommand має два аргументи (dataset_name і dataset_path). Кожен аргумент є екземпляром класу Argument. У цьому випадку dataset_name використовує регулярний вираз dataset ([a-zA-Z0-9_]+) для вилучення імені набору даних, а dataset_path використовує at ([a-zA-Z0-9_]+) для вибору шляху до файлу. Аргументи можуть мати кілька прикріплених регулярних виразів і будуть оцінювати всі, поки один не знайде відповідності. Наприклад, аргумент dataset_path містить три регулярні вирази: in ([a-zA-Z0-9_.]+), in the file ([a-zA-Z0-9_.]+) та dataset_path = ([a-zA-Z0-9_.]+). Аргументи можуть бути необов’язковими, наприклад аргумент dataset_path команди LoadDatasetCommand. У реченні «Давайте завантажимо набір даних iris» dataset_path відсутній. Однак набір даних iris є добре відомим набором даних. Команди LoadDatasetCommand містять колекцію добре відомих наборів даних. У цих випадках команда не потребує шлях для завантаження набору даних.

2.3.3. Реалізація деревовидної структури даних мовлення

Цей підрозділ описує, як команди створюють структуру даних дерева та як ми ітеруємося по розмові. Усі команди успадковуються від базових

класів Command або CommandWithArgs і містять список дочірніх команд. Крім того, кожна команда має властивість parent, яка посилається на її батьківську команду. Ці дві властивості (children і parent) створюють структуру дерева під час розмови.

Зокрема, ми використовуємо абстрактне синтаксичне дерево (AST) для захоплення значення розмови, оскільки воно може формально представляти критичну структуру введення, одночасно уникаючи синтаксичних ресурсів. Реалізації AST виграють від мов об'єктно-орієнтованого програмування (ООП) завдяки успадкуванню та поліморфізму. У цьому випадку всі вузли AST походять від базових класів, які мають спільні атрибути. Корисним шаблоном для доповнення AST і ООП є шаблон Builder. Шаблон builder дозволяє динамічно розширювати AST новими вузлами.

Структура дерева розмови містить три рівні. Перший рівень - це SpeechManager, другий рівень включає дітей SpeechManager, які є екземплярами Command або CommandWithArgs, а третій рівень містить екземпляри Argument. Рисунок 2.12 показує приклад дерева мовлення наступної розмови.

Користувач: Привіт

Бот: Привіт

Користувач: Давайте завантажимо набір даних outliers_sample_ data в data/outliers.csv

Бот: Готово

Користувач: ...

Бот: ...

Користувач: Бувай

Бот: Бувай

Спочатку в дереві є лише вузол SpeechManager. Цей вузол є кореневим і не має батьківських вузлів. Після першого введення користувача "Привіт" SpeechManager ідентифікує намір привітання і створює дочірній вузол GreetingsCommand. GreetingsCommand не має аргументів, тому у нього немає

власних дочірніх елементів. Коли користувач просить завантажити набір даних, SpeechManager ідентифікує намір load_dataset і додає дочірній елемент LoadDatasetCommand. LoadDatasetCommand має два аргументи, витягнуті з введення користувача, і створює два дочірні вузли Argument (по одному для кожного аргументу). Нарешті, коли користувач каже "Бувай", SpeechManager розпізнає намір прощання і створює вузол GoodbyeCommand.

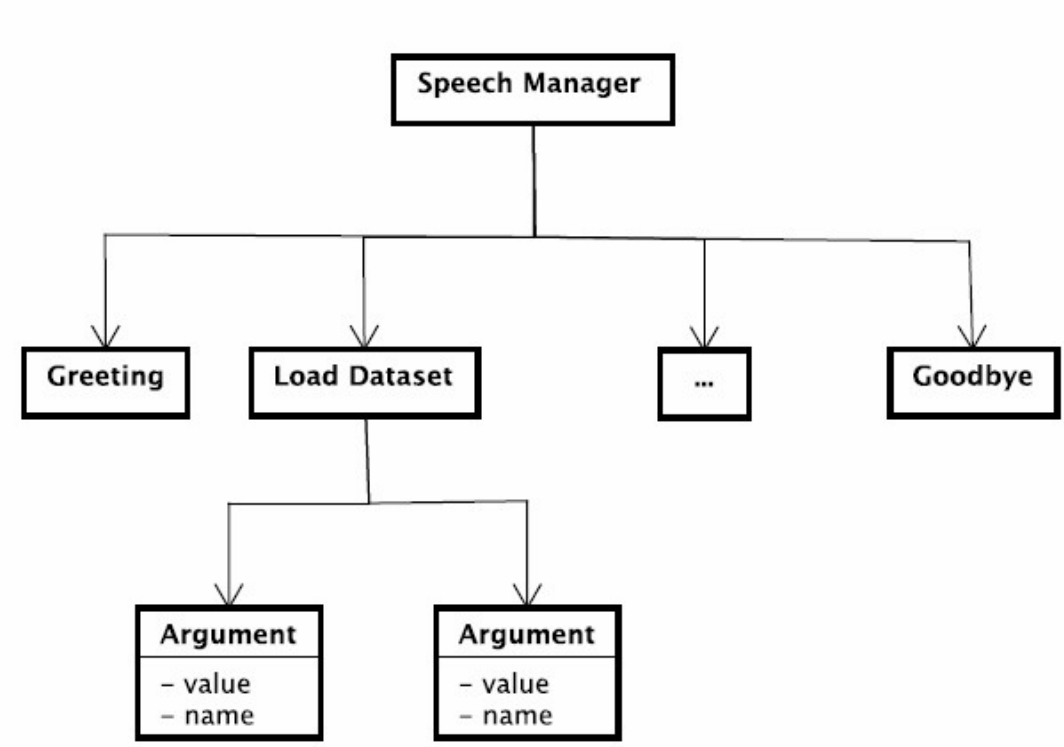


Рис. 2.12. Приклад розмови представленої в деревовидній структурі

У цьому прикладі перший рівень дерева має SpeechManager, другий рівень містить команди GreetingsCommand, LoadDatasetCommand та GoodbyeCommand, а третій рівень має аргументи.

SpeechManager має контекст для збереження змінних виконання та даних. Цей контекст спочатку порожній, але як тільки команди виконуються, він зберігає змінні та результати. Кожна команда має властивість complete, і вона сповіщає SpeechManager, коли вона готова до виконання. Виконання команд завжди відбувається спочатку для дітей і зліва направо. Наприклад, коли користувач каже "Привіт", згенерований GreetingCommand не має

аргументів, тому він готовий до виконання. Однак LoadDatasetCommand готовий лише тоді, коли всі його діти готові. Отже, це залежить від завершення їхніх дочірніх елементів Argument. У цьому випадку аргументи знаходяться в одному вводі користувача. У випадку, коли потрібні аргументи відсутні, бот запитує їх, а SpeechManager зберігає останніх неповнолітніх дітей. Бот відновить виконання, коли потрібні аргументи матимуть правильні значення. Коли LoadDatasetCommand завантажує набір даних у пам’ять, він зберігає посилання в об’єкті контексту для майбутніх команд.

2.3.4 Опис об’єктів конвеєрів

У цьому підрозділі ми описуємо об’єкти конвеєрів. По-перше, ми показуємо, як ми групуємо команди в конвеєри. Потім ми показуємо, як конвеєри представлені в структурі дерева мовлення, і, нарешті, деякі деталі виконання. У [30] автори представляють інтерактивний інструмент науки про дані та аргумент, що масштабні завдання науки про дані вимагатимуть структурованих конвеєрів команд. Масштабні завдання науки про дані можуть зайняти кілька днів. Конвеєри забезпечують миттєве подальше виконання після кожної команди та візуалізують лише часткові або кінцеві результати.

Команди конвеєрів є колекцією інших команд, і за визначенням конвеєр не може бути рекурсивним, тому він не може мати дитину того самого типу класу. Наприклад, у випадку виявлення викидів ми створили команду OutlinePipe. Команда OutlinePipe має такі дії, представлені на рисунку 2.13.

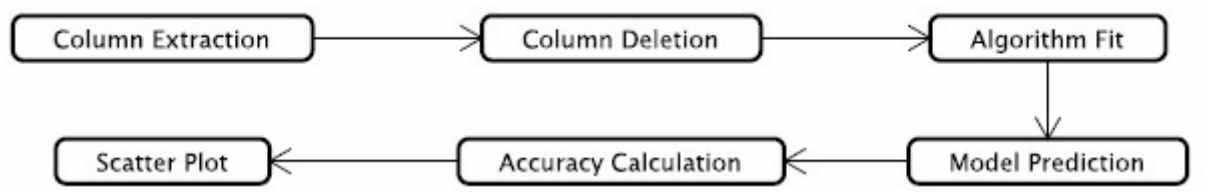


Рис. 2.13. Дії команди OutlierPipeCommand

Усі дії, згруповані всередині команди OutlierPipe, є командами (рис. 2.10). Команди ColumnExtraction і ColumnDeletion належать до групи Preprocessing, а Algorithm і Prediction знаходяться всередині групи Algorithms. Accuracy належить до групи Metrics. Нарешті, ScatterPlot розташований у групі Visualization.

Користувач: Привіт

Бот: Привіт

Користувач: Давайте завантажимо набір даних outliers_sample_data в data/outliers.csv

Бот: Готово

Користувач: Зробіть аналіз виявлення викидів у наборі даних outliers_sample_data

Бот: Готово

Користувач: Бувай

Бот: Бувай

Наведений вище діалог є прикладом використання команди конвеєра. Коли користувач каже «Зробіть аналіз виявлення викидів у наборі даних outliers_sample_data», SpeechManager розпізнає намір outline_pipe і створює екземпляр команди OutliersPipe. Команда OutliersPipe розташована між командами LoadDataset і Goodbye, як показано на рисунку 2.14.

Команда OutliersPipe містить шість дочірніх вузлів, по одному для кожної дії, переліченої на рисунку 2.13. Зверніть увагу, що з додаванням конвеєрів рівні, визначені для структури дерева мовлення, змінюються. Перший рівень залишається лише з SpeechManager, а другий рівень тепер може містити екземпляри Command, CommandWithArgs і конвеєри. Тому третій рівень, який мав лише екземпляри Argument, може мати Argument, CommandWithArgs, Command та інші конвеєри. Зверніть увагу, що конвеєр не може містити себе як дитину, але він може містити інші конвеєри.

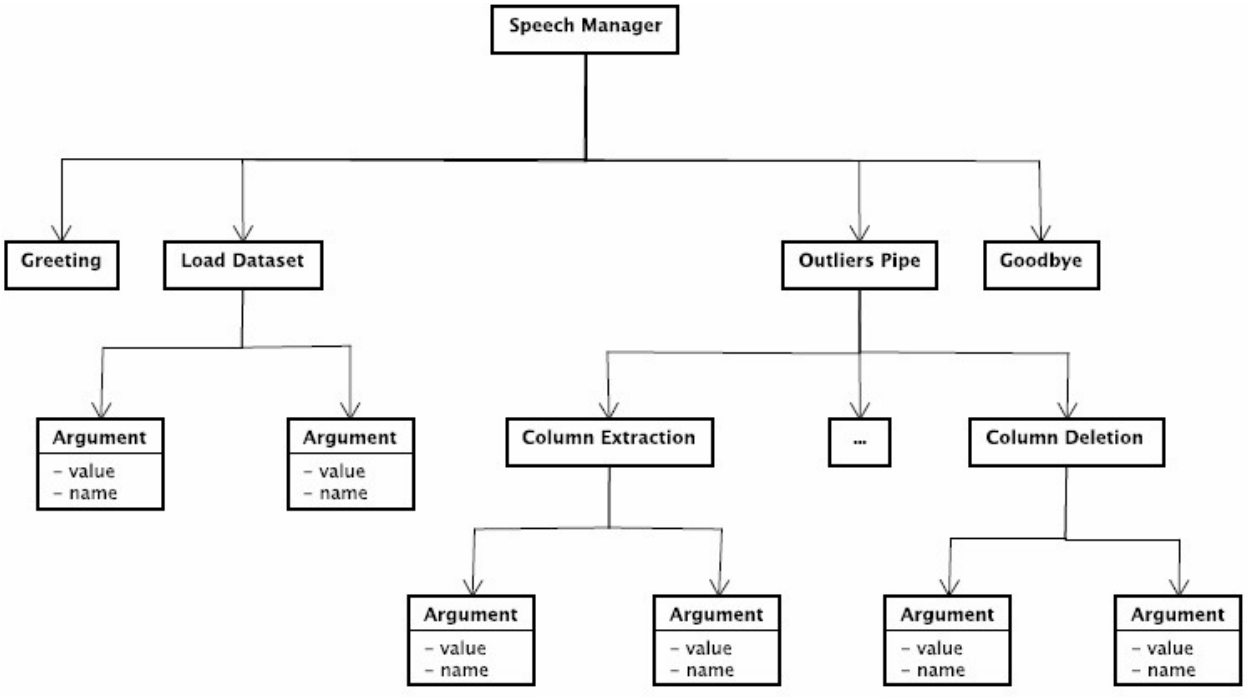


Рис. 2.14. Приклад дерева мовлення з конвеєром

Виконання конвеєра залежить від його дочірніх вузлів. Однак аргументи команд усередині конвеєра не задаються користувачем. У цьому випадку конвеєр генерує значення команд своїх дочірніх елементів. Наприклад, Algorithm Fit і Model Prediction є послідовними діями команди OutliersPipe. Після виконання команди Algorithm, OutliersPipe зберігає результат у контексті виконання та передає посилання на модель команді Model Prediction. Таким чином, команда OutliersPipe зв’язує свої дочірні команди для прямого виконання.

Конвеєри можуть тестувати кілька алгоритмів і вибирати той, який дає найкращі результати. Але нам також потрібно було спробувати кілька комбінацій параметрів. Щоб досягти нашої мети, ми підтримуємо деякі завдання AutoML всередині системи, щоб доповнити конвеєри. Ми знайшли кілька можливих пакетів AutoML, доступних в Інтернеті: TPOT, auto-sklearn, autoPyTorch і Hyperopt-Sklearn. Однак усі вони є специфічними і базуються на існуючих бібліотеках. Наприклад, Auto-sklearn виконує завдання AutoML з алгоритмами бібліотеки scikit-learn, тоді як autoPyTorch використовує

мережі PyTorch. Нам потрібен інструмент, здатний розширювати функціональність за межами цих бібліотек і адаптуватися до нових алгоритмів. Ми вибрали бібліотеку TPOT для виконання завдань AutoML у системі. Бібліотека TPOT базується на бібліотеці scikit-learn, але її можна адаптувати до інших алгоритмів, які підтримують однаковий програмований інтерфейс і методи. Таким чином, ми могли адаптувати бібліотеку TPOT до інших алгоритмів за межами бібліотеки scikit-learn, таких як бібліотека PyOD.

Ми створили TPOTCommand, який має доступ до алгоритмів оптимізації TPOT і розширює базовий клас конвеєра. Таким чином, команда TPOT може інтегруватися з системою одночасно з виконанням алгоритмів TPOT для пошуку найкращої комбінації аргументів.

2.3.5. Менеджер завдань

Менеджер завдань займається потоком виконання в додатку. Деякі команди вимагають високих обчислювальних ресурсів, і час виконання став більшим за прийнятний для утримання користувача в очікуванні відповіді. У цьому випадку ми вирішили відокремити один потік для розмови та інші потоки для виконання екземплярів команд. Екземпляр команди іноді залежить від раніше виконаних команд і використовує результати інших команд. Тому ми створили пул виконання, де всі завдання збираються та організовуються для виконання. Рисунок 2.15 деталізує членів функції TaskManager.

Менеджер завдань має колекцію працівників (Workers), здатних виконуватись як фоновий потік виконання. Він повертає результат після завершення завдання. Клас Task є визначенням завдання, яке потрібно виконати. Він містить функцію для виконання, аргументи та зворотний виклик для повідомлення результатів. TaskManager успадковує від базової черги і містить чергу завдань і групу працівників. SpeechManager містить екземпляр TaskManager, доступний для всіх його дочірніх команд.

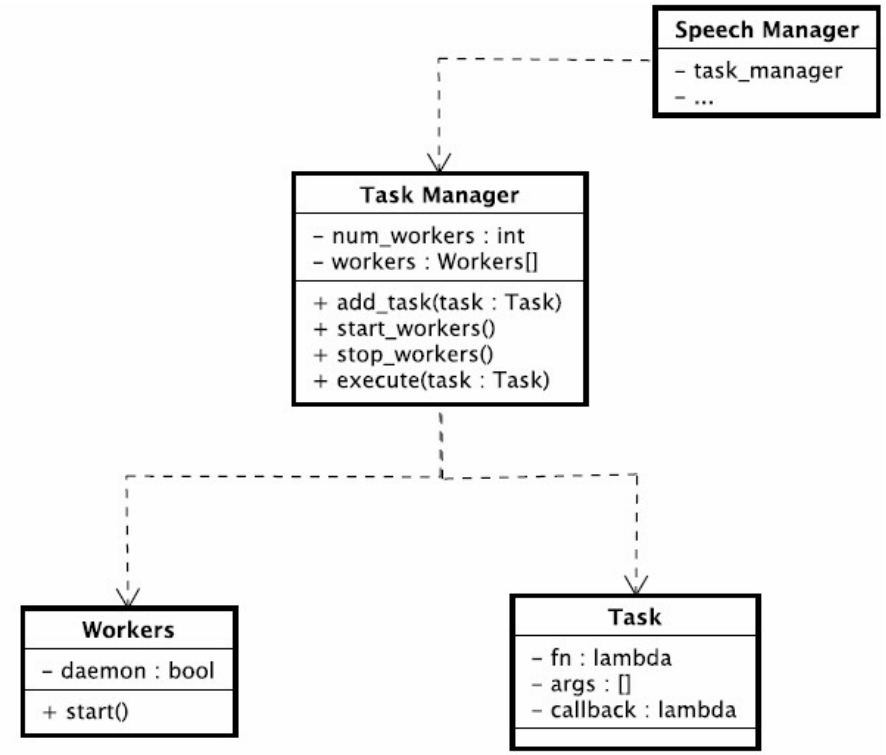


Рис. 2.15. Діаграма класів менеджера завдань

Послідовність подій, що запускається після нового повідомлення користувача, на рисунку 2.5 трохи змінюється. З цього моменту SpeechManager створює екземпляр Task з вбудованим виконанням дочірньої команди та надсилає асинхронне повідомлення TaskManager для виконання завдання. TaskManager додає отриманий екземпляр Task до своєї черги обробки, а працівники виконають його в якийсь момент. Зверніть увагу, що SpeechManager повертається користувачеві з результатом обіцянки. Коли завдання завершиться, SpeechManager отримає сповіщення через зворотний виклик завдання та представить фактичні результати користувачеві.

2.3.6. Реалізація функції генерації коду

У цьому розділі ми представляємо функцію генерації коду. Функція генерації коду дозволяє системі генерувати придатний код Python зі структури дерева мовлення. Користувач ініціює генерацію коду за допомогою команди. Команда CodeGenerationCommand успадковує базовий клас CommandWithArgs і містить один аргумент. Необхідним аргументом є

шлях для збереження файлу згенерованого коду. Коли користувач просить згенерувати код, SpeechManager розпізнає намір code_generation і встановлює завдання, як це зазвичай робиться. Генерація коду починається з виконання команди CodeGeneration. Ця команда піднімається по структурі дерева мовлення і викликає метод generate_code з вершини дерева. Рисунок 2.16 представляє об’єкт CodeGenerator.

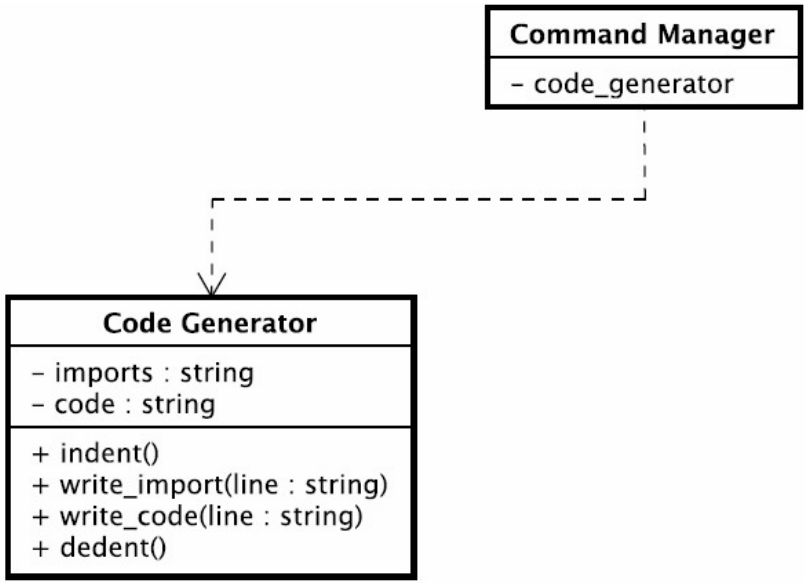


Рис. 2.16. Об’єкт CodeGenerator

CodeGenerator має код і імпорти як текстовий атрибут і підтримує операції для обробки коду Python (відступ і зменшення відступу). Модуль CommandManager містить екземпляр CodeGenerator, переданий як параметр у метод generate_code кожній команді для генерації їх блоків коду. Генерація коду дотримується того самого порядку виконання і використовує контекст для доступу до змінних і даних. Наприклад, генерація коду LoadDatasetCommand застосовує наступні кроки, детально описані на рисунку 2.17. Спочатку команда пише коментар до коду, щоб налаштувати блок коду завантаження набору даних. Усі команди генерують коментарі та розділяють свій код на блоки. Далі йдуть загальні імпорти використання, такі як numpy і pandas. LoadDatasetCommand перевіряє, чи набір даних є відомим

набором даних, доступним у пакеті, чи набором даних, завантаженим з зовнішнього файлу.

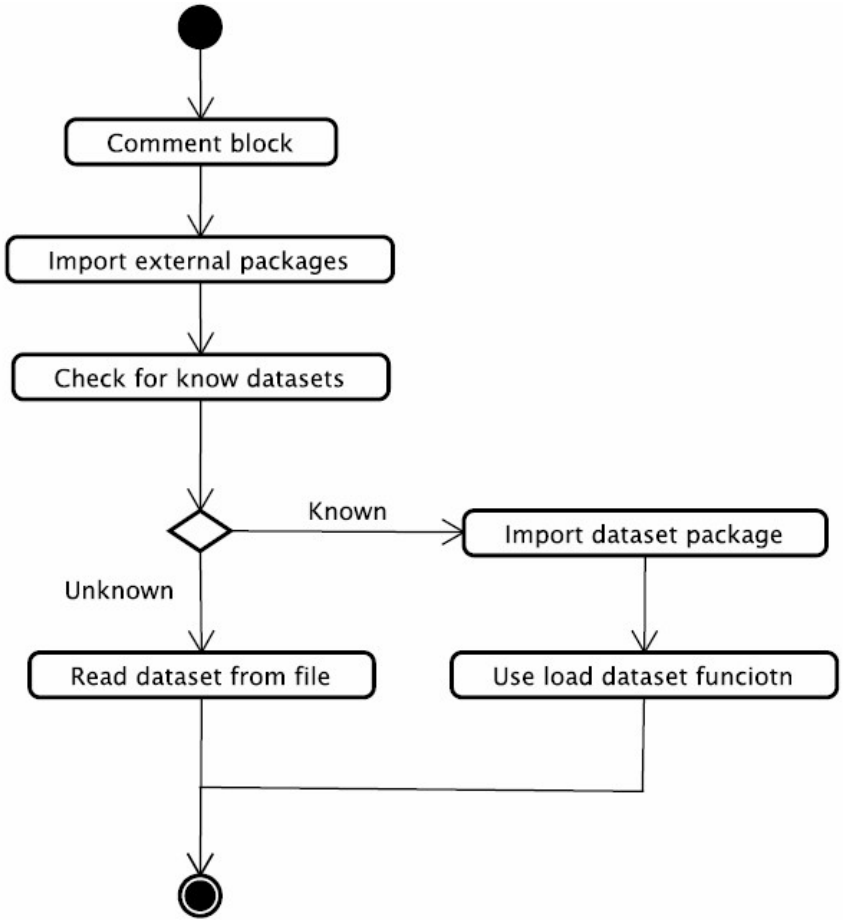


Рис. 2.17. Приклад генерації коду команди LoadDataset

У випадку відомого набору даних команда генерує необхідні імпорти набору даних і використовує функцію завантаження, надану пакетом, для використання набору даних. При роботі із зовнішніми файлами команда використовує функцію читання, надану бібліотекою pandas, для читання файлу. Зверніть увагу, що шлях до набору даних є аргументом dataset_path команди LoadDatasetCommand.

2.3.7. Функція профілю користувача

У цьому підрозділі ми представляємо функцію профілю користувача. Ми пояснюємо структуру, яка використовується для збереження профілю

користувача, і як ми оновлюємо його після кожної взаємодії з системою. Ми категоризували всі доступні команди на вісім груп, як показано на рисунку 2.18: попередня обробка, алгоритм, візуалізація, конвеєр, метрики, генерація коду, мовлення та інші. Група попередньої обробки містить команди, пов'язані з маніпуляціями даними, такі як ColumnsExtractionCommand, а категорія алгоритмів містить команди, пов'язані з використанням алгоритмів, наприклад KNNCommand. Категорія візуалізації збирає команди, пов'язані з діаграмами, статистичними графіками та іншими командами з візуальним зворотним зв'язком. Група конвеєрів містить усі екземпляри PipelineCommand, а група метрик містить команди, пов'язані з метриками. Команда CodeGenerationCommand має свою категорію, а група мовлення містить команди, необхідні для підтримки розмови з ботом. Категорія інші охоплює всі команди, які не входять до жодної іншої категорії.

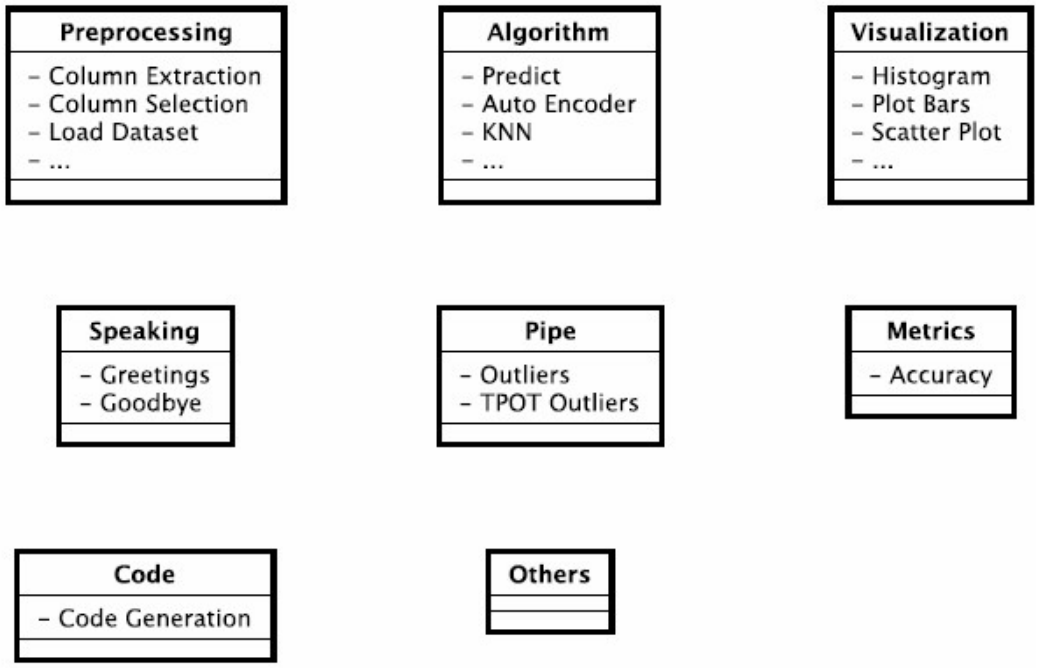


Рис. 2.18. Типи профілів користувачів та їхні пов'язані команди

Створені категорії узагальнюють групу дій. При постійному використанні системи кожен користувач може застосовувати команди всередині категорії частіше, ніж інші. У цьому випадку ми припускаємо, що

користувач має переважну категорію. Ми створюємо структуру для реєстрації використання команд і обмірковуємо категорії, представлені на рисунку 2.19.

$$user\_profile \leftarrow \begin{cases} Preprocessing = 0.125 \\ Algorithm = 0.125 \\ Visualization = 0.125 \\ Metrics = 0.125 \\ Speaking = 0.125 \\ Pipe = 0.125 \\ Code = 0.125 \\ Other = 0.125 \end{cases}$$

Рис. 2.19. Приклад аргументів у введенні користувача

Спочатку всі категорії отримують однакове початкове значення, а сума дорівнює 1. Ця структура зберігається під час кількох виконань системи. Таким чином, уподобання користувача відображають постійне використання користувачем. Коли користувач виконує команду, система оновлює профіль користувача з попередньо визначеною швидкістю. Алгоритм 1 детально описує кроки оновлення профілю користувача.

Algorithm 1 Update user profile with a new entry

```
1: procedure UPDATE_PROFILE(user_profile, last_command)
2:     last_command_category ← last_command.category
3:     user_history[last_command_category] += 0.01
4:     index ← 0
5:     total_sum ← sum(user_history)
6:     while index < len(user_history) do
7:         current_command ← user_history[index]
8:         current_command.value ← current_command.value/total_sum
9:     end while
10: end procedure
```

Алгоритм 1 отримує поточний профіль користувача та останню виконану команду. У першому рядку (2) алгоритм зберігає команду, а в

рядку 3 він збільшує значення категорії на швидкість оновлення (0,01). Алгоритм нормалізує значення з рядків 6 по 9, щоб сума всіх категорій знову дорівнювала 1,0.

Наприклад, після створення профілю користувача припустимо, що першою командою є команда LoadDataset. Алгоритм (у рядку 3) оновлює значення 0,125 до 0,126. Рядки 6-9 нормалізують решту категорій, тому сума балів становить один. Рисунок 2.20 представляє значення кожної категорії після оновлення.

$$user\_profile \leftarrow \begin{cases} Preprocessing = 0.1257... \\ Algorithm = 0.1237... \\ Visualization = 0.1237... \\ Metrics = 0.1237... \\ Speaking = 0.1237... \\ Pipe = 0.1237... \\ Code = 0.1237... \\ Other = 0.1237... \end{cases}$$

Рис. 2.20. Приклад профілю користувача після першого оновлення командою LoadDataset

Зверніть увагу, як виконання однієї команди впливає на значення інших категорій.

2.3.8. Процес персоналізації контенту бота до користувача

У цьому підрозділі ми описуємо процес персоналізації бота для конкретного користувача. Ідея полягає в тому, щоб навчатися на історії користувача.

Наша мета - досягти рівня автоматизації L2 у проектах з науки про дані та машинного навчання. Тому бот не має автономії для самостійного виконання команд, а лише виконує команди, задані користувачем. Це означає, що бот не ініціюватиме дії, але він може вивчати найчастіші послідовності команд і пропонувати продовження аналізу. Наприклад,

користувач часто використовує команду PredictCommand після навчання моделі за допомогою команди AlgorithmCommand. У цьому випадку бот може представити результати прогнозування, коли користувач запитує про команду AlgorithmCommand. Однак наша система не виконує команди проактивно і не виконуватиме команду PredictCommand у цьому випадку. Бот може використовувати ці знання по-іншому. Система може надавати пропозиції користувачам. Пропозиції виконуються, коли користувач запитує їх відповідно до рівня автоматизації L2. Різниця полягає в тому, що бот залишається реактивним на команди користувача. Якщо користувач хоче, щоб обидві команди виконувалися разом, він об'єднає обидві дії в одну команду через конвеєр. Наприклад, користувач може створити команду конвеєра з командами AlgorithmCommand і PredictCommand.

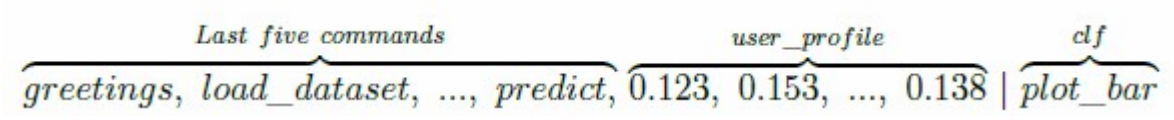


Рис. 2.21. Набір даних історії користувача

Система будує набір даних з історією команд і профілем користувача, щоб генерувати пропозиції. Цей набір даних містить найновіші ланцюжки команд розмови. Рисунок 2.21 показує рядок набору даних. Рядок складається з фіксованої кількості попередніх команд, поточного профілю користувача та поточної команди. Він читається так: користувач з таким профілем виконує поточну команду після цих попередніх команд. Таким чином, кожна нова команда перетворюється на рядок набору даних. Цей набір даних зберігає персоналізовану інформацію кожного користувача та взаємодію з системою. Ми створили модель DecisionTree з набором даних користувача і зберегли її для подальших прогнозів.

Ця модель дерева рішень використовується в класі PersonalizationManager, як показано на рисунку 2.22.

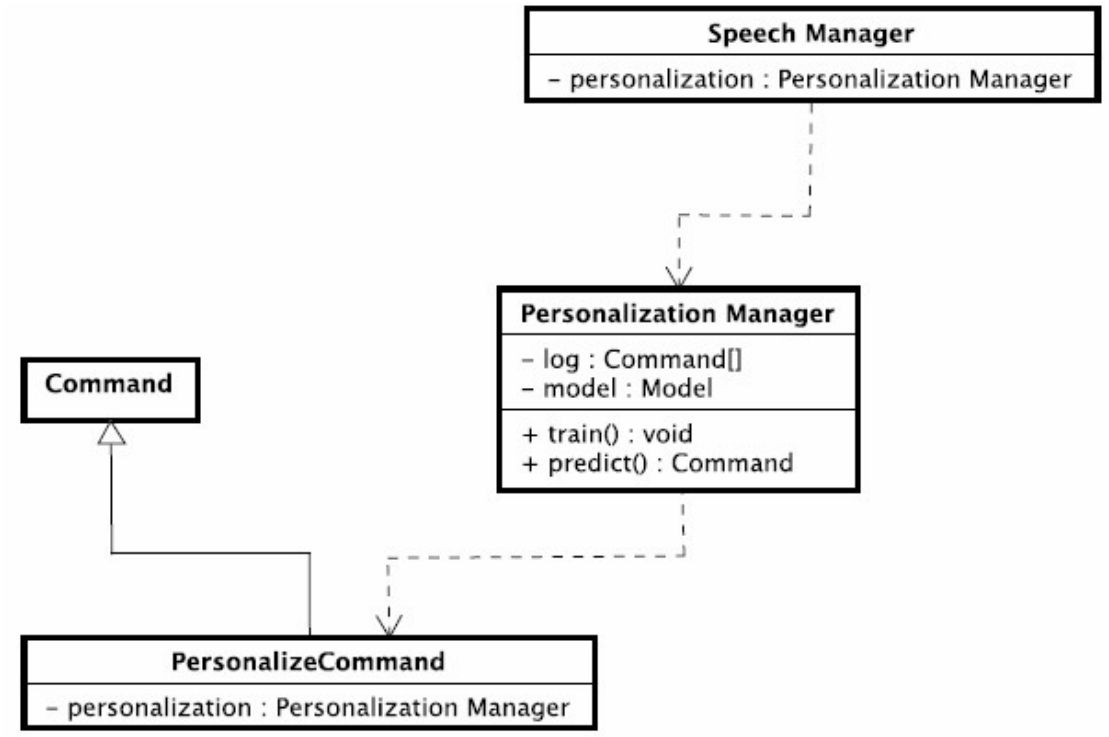


Рис. 2.22. Типи менеджера персоналізації та їхні пов’язані команди

SpeechManager передає екземпляр PersonalizationManager до деревоподібної структури в контексті виконання. PersonalizationManager відповідає за надання персоналізованих пропозицій користувачеві за допомогою прогнозів моделі. Ми додаємо PersonalizeCommand, який успадковується від класу Command. PersonalizeCommand є призначеним для користувача способом доступу до PersonalizationManager. Наприклад, у наступній розмові користувач запитує пропозицію.

Користувач: Привіт

Бот: Вітаю

Користувач: Що я можу зробити?

Бот: Завантажити набір даних.

...

Коли користувач запитує: "Що я можу зробити?", бот виявляє намір personalized_suggestion і створює екземпляр PersonalizedCommand. PersonalizedSuggestionCommand успадковується безпосередньо від Command,

а не від CommandWithArgs, тому він не потребує аргументів. Він створює рядок для прогнозування з попередніми командами та профілем користувача. У цьому випадку ми представляємо рядок наступним чином:

$$\overbrace{null,\ null,\ ...,\ greetings}^{Last\ five\ commands},\overbrace{0.123,\ 0.153,\ ...,\ 0.138}^{user\_profile}$$

Рис. 2.23. Набір даних історії користувача

Ми використовуємо рядок, описаний на рисунку 2.23, як вхідний параметр для прогнозування подальших команд. У цьому випадку модель прогнозує намір load_dataset. Намір load_dataset належить до LoadDatasetCommand, і PersonalizedCommand може нарешті запропонувати користувачеві команду LoadDatasetCommand.

2.3.9. Інтеграція з Telegram

Цей підрозділ показує, як ми підключаємо наш розмовний агент до інтерфейсу API Telegram. Telegram API (https://core.telegram.org/bots/api) пропонує кілька сервісів для запуску чат-бота всередині середовища Telegram. Це середовище включає багатоплатформові інтерфейси користувача для мобільних, комп'ютерних і планшетних додатків. Для реалізації нашого рішення ми використовували бібліотеку python-telegram, доступну на https://pypi.org/project/python-telegrambot/.

Серед сервісів, що пропонуються Telegram API, ми можемо знайти ігри, редагування повідомлень тощо. У нашому чат-боті ми реалізуємо такі служби: отримання тексту, надсилання зображення, надсилання тексту та вбудовану клавіатуру.

Текстові служби (отримати та надіслати текст) є основою нашого розмовного інтерфейсу та вирішують більшість потреб користувача. Поєднуючи ці дві служби, боти можуть отримувати вхідні дані користувача,

обробляти їх і відповідати текстовим повідомленням. Однак деякі вхідні дані користувача повинні повертати діаграми. У цих випадках бот використовує службу надсилання зображення, щоб надіслати зображення. Вбудована клавіатура дозволяє здійснювати множинний вибір через інтерфейс Telegram і може запитувати діалогові вікна підтвердження.

Бот також містить список спеціальних команд. Команди - це слова, що передують слешем, наприклад, /start. Команди дозволяють користувачеві взаємодіяти з ботом в іншій області та налаштовувати його параметри. Зверніть увагу, що команди не впливають на історію поточної розмови. Ми реалізуємо такі команди:

/hi, щоб перевірити, чи бот онлайн.

/spelling, щоб увімкнути/вимкнути коректор орфографії.

/learn: зіставити фразу з новим наміром. У цьому випадку фраза додається до набору даних для подальшого навчання.

/suggest: попросити бота рекомендацій щодо того, як продовжити аналіз.

Усі попередні команди сприятимуть більш гнучкому розмовному інтерфейсу.

2.4. Представлення архітектури запропонованого рішення

У цьому підрозділі представлено запропоноване рішення як фреймворк та виділено фіксовані та змінні компоненти. На рисунку 2.24 зображено огляд фреймворку рішення.

Компоненти з білим фоном є статичними, а ті, що з червоним фоном, є змінними. У цьому випадку базові класи команд (Command, CommandWithArgs, Argument та PipelineCommand) є частиною ядра рішення. Однак нові команди можуть розширювати ці класи відповідно до ситуації. Інші основні функціональності - це модулі TaskManager, UserProfile, CodeGenerator, ExecutionContext, Personalization та IdentifyIntention. У

випадку модуля IdentifyIntention модель, яку використовує бот для ідентифікації намірів, потрібно адаптувати до нових команд. Модель, що використовується в модулі Personalization, знаходиться в тій самій ситуації, і ми можемо дослідити інші техніки для покращення пропозицій бота.

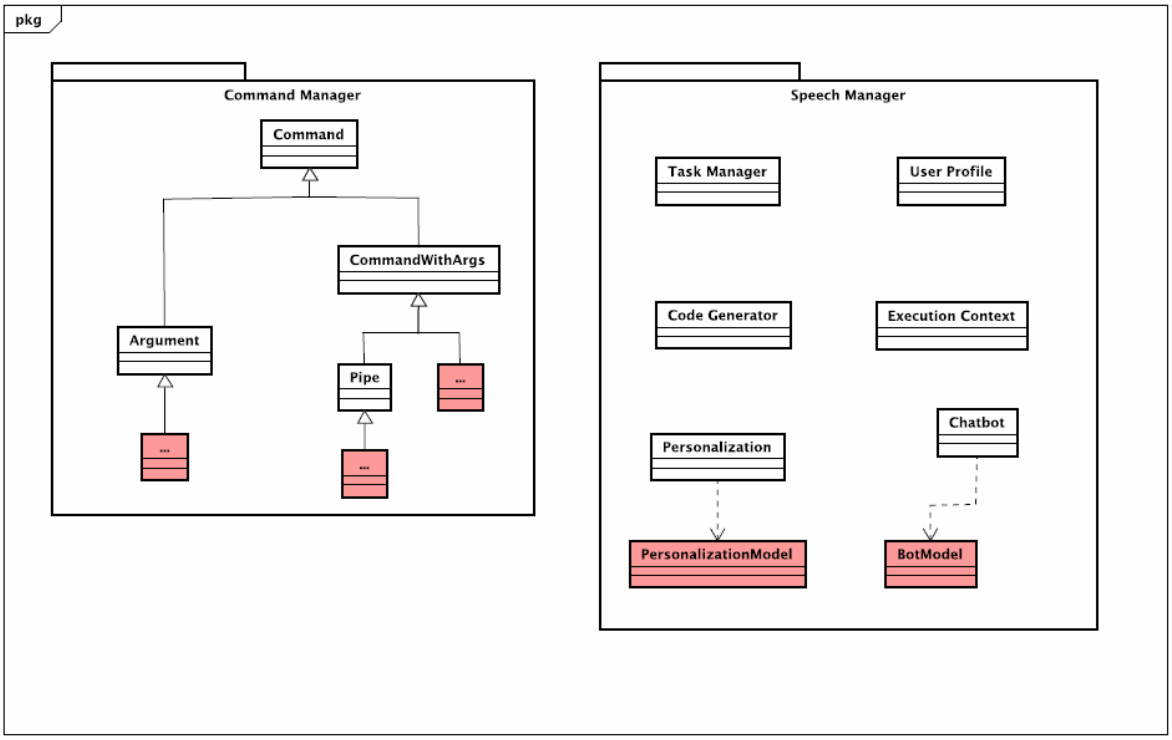


Рис. 2.24. Архітектура пропонованого рішення

Нарешті, ми можемо змінити категорії модуля UserProfile відповідно до нових команд, і модуль UserProfile адаптується до нових категорій.

Підсумовуючи, базою модифікацій є нові команди, похідні від базових класів Command. Однак інші компоненти, такі як UserProfile, IdentifyIntention та Personalization, можуть бути розширені, щоб бути узгодженими з новими командами.

Висновки до розділу

У цьому розділі було розглянуто підходи до генерування персоналізованого контенту за допомогою розмовних інтерфейсів, зокрема

чат-ботів, і представлено власне рішення, що враховує сучасні вимоги до інтерактивності та адаптивності таких систем.

Аналіз літератури показав, що ефективна персоналізація контенту значно залежить від врахування соціальних характеристик користувача. Для цього необхідно створювати концептуальні моделі, які включають такі аспекти, як мовленнєві уподобання, стиль спілкування та специфіку взаємодії. Ці моделі визначають основу для побудови чат-ботів, здатних забезпечувати гнучку та природну комунікацію.

Запропоноване рішення включає багаторівневу архітектуру, що складається з модулів, кожен із яких відповідає за певну функцію бота. Рішення передбачає ідентифікацію намірів користувача, побудову ієрархії команд, використання деревовидної структури даних для оптимізації мовлення та управління завданнями через спеціалізований менеджер.

Особливу увагу приділено функції генерації коду, яка дозволяє боту виконувати складні завдання, та функції профілю користувача, що відповідає за збір даних і налаштування бота відповідно до індивідуальних потреб користувача. Процес персоналізації контенту реалізовано таким чином, щоб бот міг динамічно адаптуватися до змін у поведінці та вподобаннях користувача.

Інтеграція з платформою Telegram забезпечує зручність використання та доступність рішення для широкої аудиторії. Використання популярного месенджера дозволяє спростити впровадження бота та забезпечити користувачам знайомий інтерфейс.

Розроблена архітектура вирізняється модульністю та масштабованістю, що дозволяє легко додавати нові функції або адаптувати бот під специфічні завдання. Запропонований підхід до побудови чат-ботів демонструє практичну реалізацію сучасних концепцій персоналізації та інтерактивності, сприяючи розвитку інтелектуальних розмовних інтерфейсів.

РОЗДІЛ 3. ОЦІНКА ЗАПРОПОНОВАНОГО ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ РІШЕННЯ ЗАДАЧ DATA SCIENCE

3.1. Екземпляри платформи

Цей розділ описує, як ми оцінюємо запропоноване рішення, презентуємо отримані результати та обговорюємо їх наслідки. Було проведено два експерименти. В першому експерименті, ми інстанціюємо нашу платформу для двох різних ситуацій: виявлення аномалій та очищення даних. У другому експерименті, ми виконуємо завдання з Data Science, використовуючи компоненти платформи.

3.1.1. Виявлення аномалій

Поширеною проблемою в науці про дані є виявлення аномалій, широко досліджувана проблема в літературі. Існує багато методик та алгоритмів для вирішення таких ситуацій. Також існує декілька застосувань у виявленні шахрайства з кредитними картками, безпеці мережі тощо. Проте, всі методики спираються на аналіз даних для виявлення дивних патернів. Виявлення аномалій спрямоване на ідентифікацію відхилень у даних. Ці відхилення можуть бути людськими помилками, шахрайством або просто природним відхиленням розподілів даних. Простими прикладами аномальних спостережень є негативний вік через людські помилки або нестандартні транзакції з клонованих кредитних карток.

Ми обрали набір дій, пов'язаних із виявленням аномалій:

а) витягти стовпець із набору даних,

б) видалити стовпець із набору даних,

в) навчити певний алгоритм виявлення аномалій,

г) зробити прогноз за допомогою створеної моделі

д) візуалізувати діаграми розсіювання.

Ми співставляємо кожну дію з наміром користувача наступним чином:

- Витягти стовпець із набору даних: column_extraction
- Видалити стовпець із набору даних: column_deletion
- Навчити певний алгоритм виявлення аномалій: outlier_algorithm
- Зробити прогноз за допомогою створеної моделі: outlier_prediction
- Візуалізувати діаграми розсіювання: scatter_plot

Використовуючи активні точки запропонованого рішення, ми створюємо одну команду для кожного нового наміру, розширюємо модель персоналізації та створюємо нову модель чат-бота. Нові команди потребують аргументів, тому ми успадковуємось від базового класу CommandWithArgs, а не від базового класу Command.

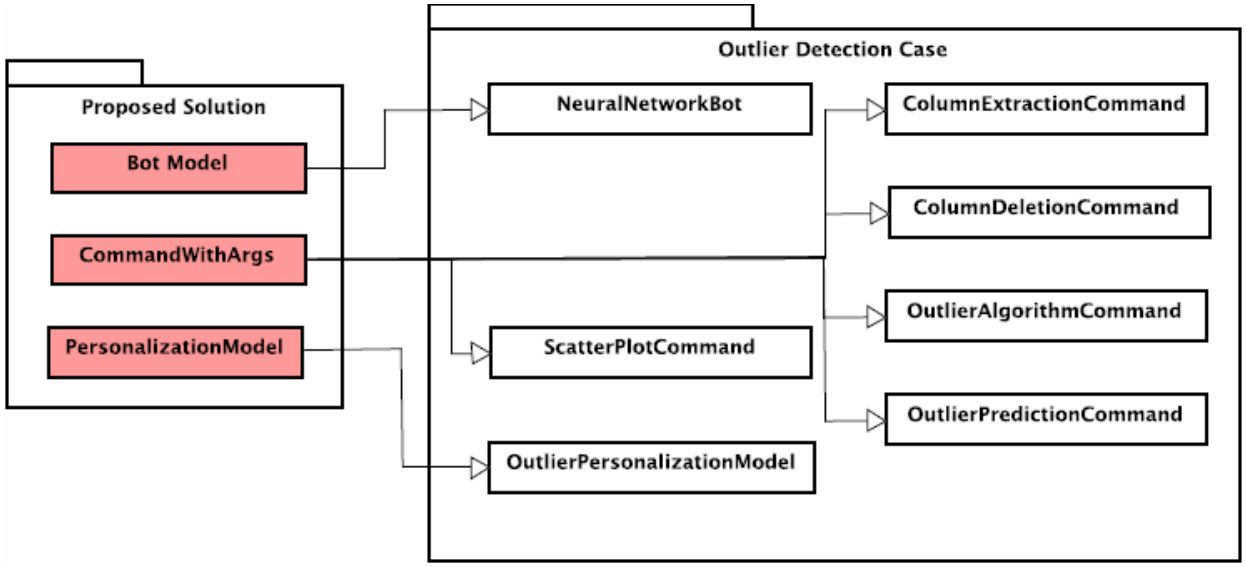


Рис. 3.1. Реалізація сценарію використання виявлення аномалій

На рисунку 3.1 показано команди, створені для цього прикладу: ColumnExtractionCommand, ColumnDeletionCommand, OutlierAlgorithm Command, OutlierPredictionCommand та ScatterPlotCommand. Команда OutlierAlgorithmCommand отримує алгоритм, який потрібно навчити. Ми включили такі алгоритми: Головний компонентний аналіз (PCA), Однокласові опорні векторні машини (OCSVM), Локальний фактор аномалій (LOF), Локальний фактор аномалій на основі кластерів (CLOF), Оцінка аномалій на основі гістограми (HBOS), K-найближчих сусідів (KNN),

Виявлення аномалій на основі кута (ABOD), Ізольований ліс (IForest), Виявлення аномалій на основі екстремального посилення (XGBOD), Легкий онлайн-детектор аномалій (LODA) та Автокодер з повним зв'язком (AutoEncoder). NeuralNetworkBot використовує ту саму архітектуру, що деталізовано на рисунку 2.8, тоді як OutlierPersonalizationModel використовує модель персоналізації за замовчуванням, доступну в запропонованому рішенні. Ми розширили обидві моделі, щоб додати журнали відладки їхніх операцій.

Для того, щоб представити цей сценарій, ми генеруємо простий діалог виявлення аномалій:

Користувач: Привіт

Користувач: Завантажимо набір даних outlier_sample_data з data/outlier.csv

Користувач: Виберіть цільовий стовпець і збережіть його в outlier_target

Користувач: Видаліть цільовий стовпець із набору даних outlier_sample_data

Користувач: Запустіть алгоритм LODA і збережіть його в autoencoder_model

Користувач: Зробіть прогноз за допомогою autoencoder_model і збережіть його в model_predictions

Користувач: Побудуйте діаграму розсіювання з model_predictions як кольорами

Користувач: Згенеруйте код в data/sample.py

Користувач: До побачення

Попередній діалог містить можливі дії користувача. Однак ми опустили відповіді бота та переривання через відсутні параметри, щоб підсумувати сценарій. Зверніть увагу, що в прикладі сценарію користувач

викликає всі створені команди та використовує деякі інші загальні команди, такі як LoadDatasetCommand або GenerateCodeCommand.

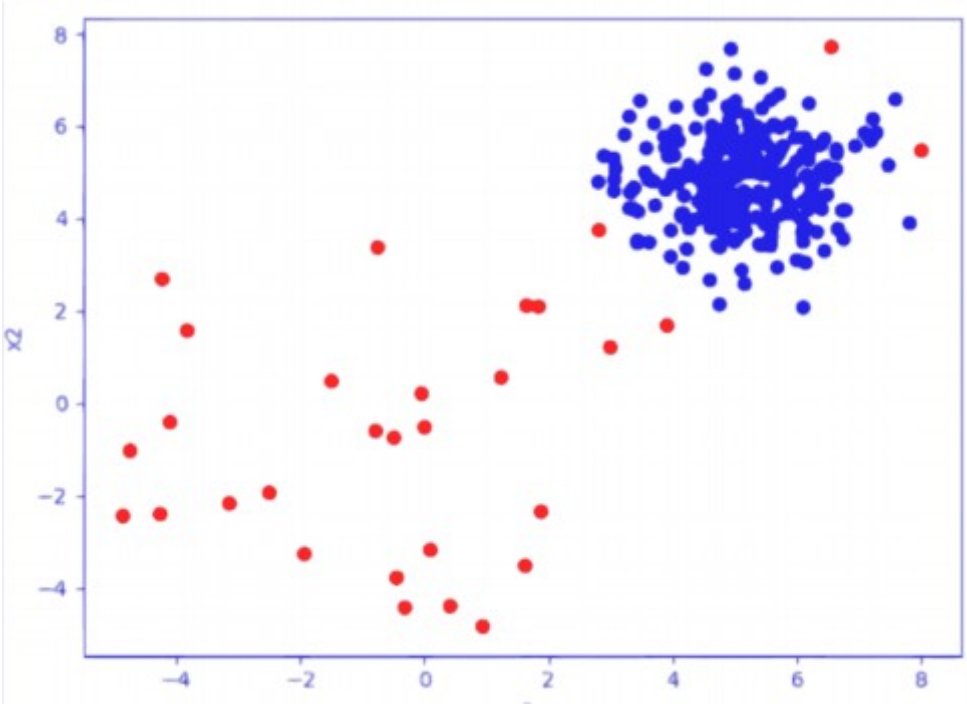


Рис. 3.2. Результат діаграми розсіювання аномалій

На рисунку 3.2 показано результат команди ScatterPlotCommand. Діаграма використовує передбачення, згенеровані командою OutlierPrediction Command, щоб позначити аномалії червоним кольором, а звичайні дані - синім. Команда OutlierPredictionCommand використовувала модель, згенеровану командою OutlierAlgorithmCommand та алгоритмом LODA. Команди ColumnExtractionCommand та ColumnDeletionCommand використовувалися для підготовки набору даних. Алгоритм LODA використовував аргументи за замовчуванням для створення моделі. Однак ми хотіли знайти кращу модель. Щоб подолати цю ситуацію, ми створили конвеєр для пошуку більш точної моделі.

Нова команда конвеєра, створена OulierPipeCommand, розширює базовий клас BasePipeCommand і визначає послідовності команд для виконання та порядок виконання. У нашому випадку ми створили наступну послідовність команд: ColumnExtractionCommand, ColumnDeletionCommand і

TPOTCommand. Команда TPOT визначає пошук у сітці з кількома алгоритмами та їхніми параметрами. У нашому прикладі TPOT повернув комбінацію CBLOF(input_matrix, contamination=0.1, n_clusters=8) з точністю 98%.

3.1.2. Очищення даних

Очищення даних є одним із ранніх етапів процесу обробки даних і міститься на етапі попередньої обробки даних. Процес обробки даних можна розділити на десять етапів. Одним із десяти етапів є попередня обробка даних, яка містить очищення даних як один із підходів до обробки даних. Очищення даних включає в себе всі техніки, стратегії та методи перетворення неправильних або суперечливих даних. Таким чином дані повинні бути перетворені, щоб задовольнити вимоги конкретного завдання науки про дані. Типовими прикладами очищення даних є відсутні клітинки, текст у числових стовпцях або дубльовані спостереження.

Щоб реалізувати запропоновану структуру рішення, ми вибрали набір дій з очищення даних:

а) дублювати в стовпці,

б) перетворювати стовпець на нижній/верхній регістр,

в) видаляти додаткові пробіли,

г) замінювати Null/NaN спостереження,

д) нормалізувати стовпець

е) сортувати стовпець.

Всі вибрані дії зручні в різних ситуаціях. Перетворення на нижній регістр корисне, коли дані надходять з тією самою сутністю, написаною за допомогою комбінації нижнього та верхнього регістрів. Наприклад, такі назви країн, як (Бразилія, БРАЗИЛІЯ та Бразилія). Усі попередні спостереження слід звести до одного виразу. Спостереження Replace Null/Nan стають у пригоді, коли в даних є нульова комірка, яка потенційно може знизити продуктивність на наступних етапах процесу обробки даних.

Нульові/NaN спостереження можна замінити середнім значенням, медіаною, значенням за замовчуванням тощо відповідно до кожної специфікації проблеми або контексту домену. Однак у нашій реалізації ми дозволяємо носіям і медіанним значенням замінити порожнє спостереження. Нормалізація стовпця є ключовою, коли виникають проблеми з масштабуванням даних і може ввести в оману решту процесу обробки даних. Підводячи підсумок, можна сказати, що всі вибрані завдання з очищення даних дуже специфічно допомагають дослідницькому процесу вихідних даних. Ми зіставляємо кожне завдання з наміром користувача та генеруємо нову команду для кожного наміру:

- дублювати в стовпці: column_duplication
- перетворити стовпець на нижній/верхній регістр: lower_upper_case
- видалити додаткові пробіли: remove_white_spaces
- замінити Null/NaN спостереження: replace_empty
- нормалізувати стовпець: column_normalization
- сортувати стовпець: sort_column.

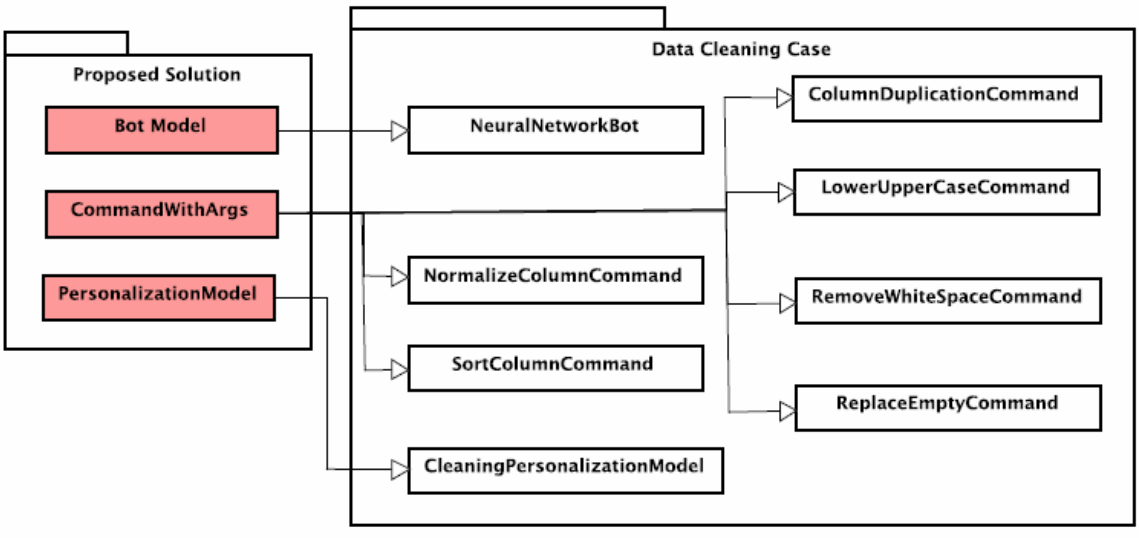


Рис. 3.3. Реалізація фреймворку сценарію використання очищення даних

Рисунок 3.3 демонструє, як кожен створений намір перетворювався на нову системну команду, викликану введенням користувача. Щоб розширити

архітектуру фреймворку, ми успадковуємо відповідні базові класи, перенавчаємо бота намірів і модель персоналізації. Для оцінки створеної команди генеруємо приклад сценарію:

- *Користувач: Привіт*
- *Користувач: завантажуємо набір даних cleaning_sample_data в data/data_cleaning.csv*
- *Користувач: видаліть повторювані рядки зі стовпця x2*
- *Користувач: перетворіть на нижній регістр стовпець x1 із набору даних cleaning_sample_data*
- *Користувач: Замініть порожні значення середнім значенням стовпця x1*
- *Користувач: нормалізуйте стовпець x1 вибіркових даних очищення набору даних*
- *Користувач: відсортуйте стовпець x1 набору даних cleaning_sample_data*
- *Користувач: До побачення*

Попереднє діалогове вікно було створено для запуску всіх реалізованих команд і обробки тестового набору даних. Зауважте, що ми опустили відповіді ботів і переривання параметрів, щоб спростити приклад.

На рисунку 3.4 (a) показано набір даних до процесу очищення, а на рисунку 3.4 бb) – набір даних після завершення останньої команди сортування. Кожна команда отримує набір даних із контексту виконання, обробляє дані та замінює те саме посилання оновленим набором даних. Таким чином, очищення виконується на тому самому наборі даних замість дублювання даних для кожної команди. Крім того, використовуючи те саме посилання, команди можуть установлювати послідовність, подібну до шаблону конструктора, де одна команда отримує результат попередньої команди очищення та генерує вхідні дані для наступної команди очищення. Наприкінці процесу ми видаляємо непотрібні пробіли, нормалізуємо

стовпець x1, замінюємо записи null/nan середнім значенням стовпця та сортуємо стовпець x1. Зверніть увагу, як сортування псує ідентифікатори спостережень.

	x1	x2
0	4.008780414158585437e+00	4.875245777243090828e+00
1	6.659941198333887868e+00	3.776461834377619731e+00
2	4.910271640185587039e+00	6.145712733467220801e+00
3	5.076319429876489536e+00	
4	6.500727364543041453e+00	4.879499841615219324e+00
5	3.888119031544130522e+00	4.623883074311895491e+00
6	4.611637611170428031e+00	3.803856355819496216e+00
7		3.925670017337897999e+00
8	6.102074501084556957e+00	3.498066480873270656e+00
9	NaN	3.482103922782001870e+00
10	NaN	4.448994311108052102e+00

а) до

	x1	x2
280	0.000000	-2.435304519972205650e+00
279	0.000925	-1.026126124762823721e+00
288	0.046668	-2.396337462174522592e+00
291	0.049046	2.688089640539730682e+00
290	0.058914	-4.084455996308253845e-01
275	0.080510	1.573295326844569075e+00
299	0.133189	-2.167208367657778103e+00
296	0.183241	-1.934377887070082291e+00
289	0.227191	-3.260185843517168003e+00
295	0.261056	4.786206966893038484e-01
292	0.316443	-5.983108124258205152e-01

б) після

Рис. 3.4. Набір даних до та після очищення даних

3.2. Використання розробленого фрейворку для рішення задач Data Science

Дослідження складалося з чотирьох етапів:

1. Встановлення,
2. Навчання,
3. Завдання,
4. Анкетування.

Кожен етап має свою специфічну мету і залежить від завершення попередніх етапів. Метою цього дослідження є перевірити, чи може розробник даних використовувати фреймворк.

Етап встановлення полягає у встановленні фреймворку та його залежностей на комп'ютері учасника. На цьому етапі ми встановили фреймворк на різних операційних системах (Windows, Ubuntu, Debian та Mac-OS) та версіях Python. Метою цього етапу є налаштування середовища, і

він більшу частину часу керувався учасниками. Аналітик міг усунути помилки встановлення, які учасники не змогли виправити.

Після того, як учасник створив відповідне середовище, ми перейшли до етапу Навчання. Етап навчання складався з кількох блоків коду для виконання та відтворення. Навчання показує, як розширити фреймворк новою командою ColumnNamesCommand. Команда ColumnNamesCommand перераховує стовпці набору даних, вже завантаженого фреймворком. Кожен блок коду супроводжується поясненням та покращує повний, але простий приклад використання. Метою цього етапу є надання базових прикладів використання та розуміння фреймворку. Аналітик керував цим етапом, а учасник виконував інструкції, щоб фреймворк з прикладами працював. Цей етап являє собою діалог, де учасники задають питання про фреймворк, а аналітик відповідає на всі запитання, навіть ті, що відхиляються від вмісту навчання. Наприклад, команда ColumnNamesCommand має параметр для вибору набору даних та відображення стовпців. Однак багато учасників хотіли зрозуміти, як фреймворк внутрішньо витягує назву набору даних.

Після того, як учасники заявили, що вони можуть створювати власні команди, ми перейшли до етапу Завдання. Етап Завдання складався з двох завдань: по-перше, створити команду для відображення заголовка набору даних, вже завантаженого у фреймворк. Нашою метою цього завдання було отримати загальне уявлення про те, наскільки добре користувач засвоїв навчання. Зверніть увагу, що команда навчання ColumnNamesCommand показує стовпці набору даних. Користувачеві слід внести незначні зміни, щоб виконати це завдання. Друге завдання полягало в тому, щоб учасник застосував логарифмічну функцію до першого стовпця набору даних. Ця команда потребує глибшого розуміння фреймворку та підвищує складність завдання. Нарешті, ми попросили учасників виконати їхній код та протестувати дві попередні команди на наборі даних IRIS. Нашою метою було оцінити, чи команди були реалізовані правильно, та виправити помилки

виконання. Нарешті, після успішного тестування коду учасники перейшли до етапу Анкетування.

Анкета складається з розділів: завдання та код. Розділ "Завдання" містить загальні питання щодо фреймворку. Він також містить конкретні питання щодо кожного окремого завдання (кожної запрошеної команди). Нарешті, у розділі коду учасники мали завантажити свій код Python для аналізу.

Запропоноване рішення призначене для допомоги науковцям даних, які хочуть автоматизувати деякі етапи робочого процесу. Допомогти початківцям науковцям даних, які можуть скористатися командами, створеними іншими науковцями даних. Генерувати сценарії аналізу даних, якими можна ділитися та вручну покращувати іншими науковцями даних. Іншими словами, науковець даних доходить до часткового аналізу даних і може генерувати сценарій, щоб поділитися результатами, отриманими в запропонованому рішенні, з іншими колегами. Користувачі повинні розуміти предметну область з точки зору науки про дані та повинні визначити, змоделювати та сформулювати проблему.

З іншого боку, користувачам, які розширюють запропоноване рішення та створюють нові команди, знадобиться наступний бекграунд: - знання Python, фреймворків та об'єктно-орієнтованого програмування (ООП) - знання процесу науки про дані та основних стратегій, використовуваних на кожному етапі.

3.2.1. Отримані результати

Усі учасники виконали обидва завдання, відповідаючи на наше питання з першого розділу - Чи зможуть розробники даних розширити запропоновану структуру для конкретних сценаріїв?

У першому завданні 75% учасників повідомили, що вони завершили його з невеликими зусиллями, тоді як решта 25% стверджували, що витратили помірну кількість зусиль для завершення першого завдання. Ніхто

з учасників не вважав перше завдання дуже складним. Більше учасників 82% вважали друге завдання дуже легким, але вони вже виконали перше завдання. Однак один учасник (3,6%) вважав друге завдання дуже складним для виконання, а решта 14% вважали, що друге завдання вимагало помірної кількості роботи.

Перша команда була дуже схожа на приклад з навчального посібника і вимагала незначних змін для завершення. Друга була більш віддаленою і вимагала глибшого розуміння фреймворку. Однак більше учасників заявили, що друге завдання було дуже легким для виконання (82%) проти (75%) першого завдання. З іншого боку, один учасник вважав друге завдання складнішим, ніж перше. Загалом, межа похибки коливається від +-16% у першому завданні до +-15% у другому завданні.

Ми створили набір змінних для характеристики профілю кожного учасника та його результатів у дослідженні, показаних у таблиці 3.1.

Таблиця 3.1.

Змінні, створені для кількісної оцінки важливих аспектів під час дослідження

Variable Name	Description
participant	This variable identifies the participants with a unique code.
crawl_debugged	This variable identifies the users who investigated the source code by crawling the git files or debugging the tutorial examples.
code_generation	This variable identifies the participants who asked questions, did not understand or skipped the code generation methods.
left_super	This variable identifies the participants who left the tutorial superclass in their code.

imports	This variable identifies the participants who asked questions or missed the import numpy package during the activities.
dataset_reference	This variable identifies the participants who omitted the dataset reference during the execution of the activities.
bot_extension	This variable identifies the participants who did not register their commands in the framework.
unused_code	This variable identifies the participants who noticed the unnecessary code chunk in the tutorial examples.
suggest_updates	This variable identifies the participants who suggested at least one relevant improvement on the framework. This variable means that the participant achieved a reasonable level of understanding of the framework limits and proposed an improvement of some kind.
first_activity difficulty	This variable identifies the level of difficulty to complete the first activity. The possible values are: (1) very easy, (2) reasonable work, (3) very hard, (4) I couldn't do it.

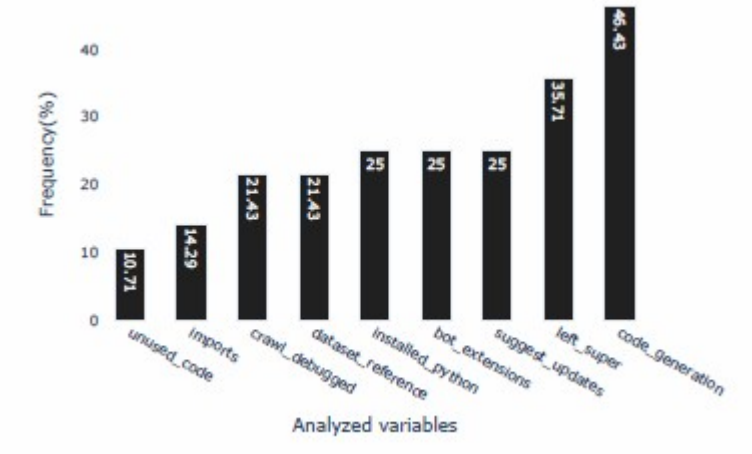


Рис. 5.5. Частота подій у наборі даних

На рисунку 3.5 ми можемо спостерігати частоту (у відсотках) булевих змінних, що цікавлять, зібраних у дослідженні. Майже половина учасників -

(50%) виявляли труднощі з генерацією коду під час дослідження. Проте всі вони завершили дії після кількох спроб. Зазначимо, що 25% учасників зробили відповідні пропозиції, які підтверджують глибше розуміння фреймворку. 20% прочитали або налагодили вихідний код, щоб прояснити поведінку деяких фреймворків. 25% учасників не мали на комп'ютері встановленої версії python, і лише 10% помітили невикористані рядки коду в посібнику. Таким чином, 90% учасників прийняли ці рядки коду та не ставили під сумнів їх роль у посібнику.

Ми створили кореляційну матрицю, щоб проаналізувати, як змінні пов'язані між собою. На рисунку 3.6 представлена кореляційна матриця.

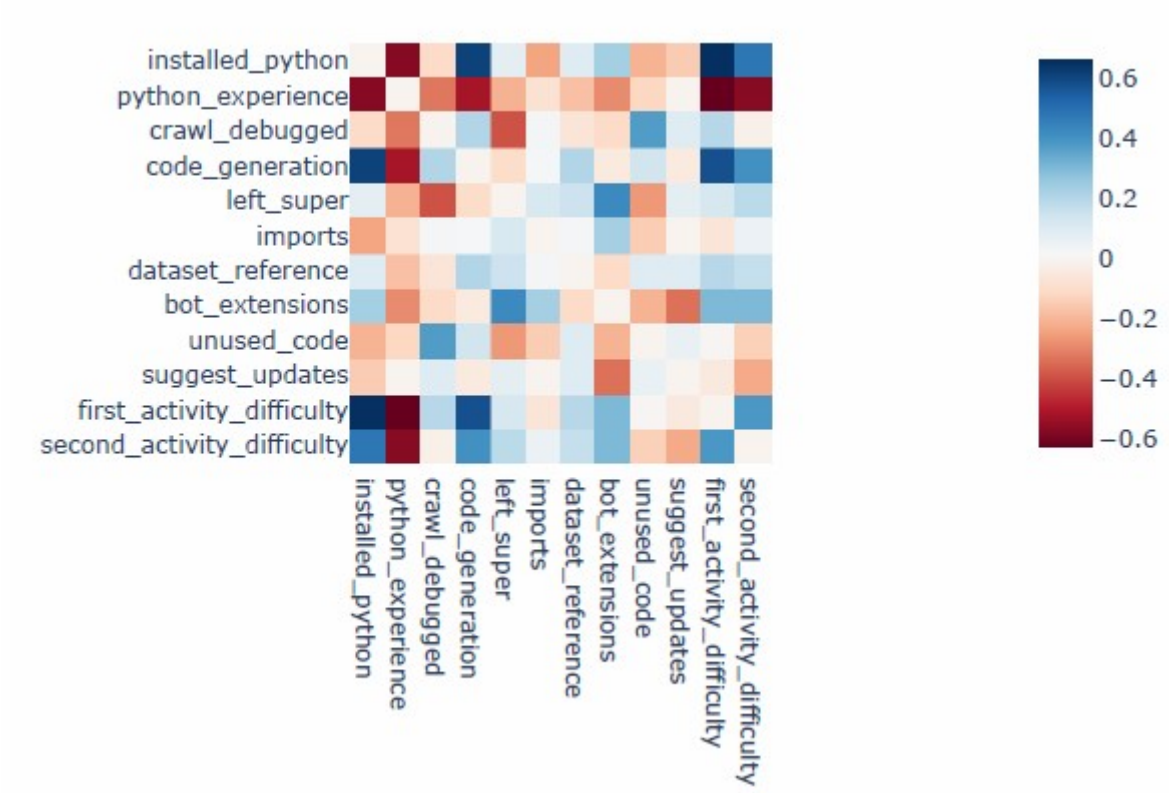
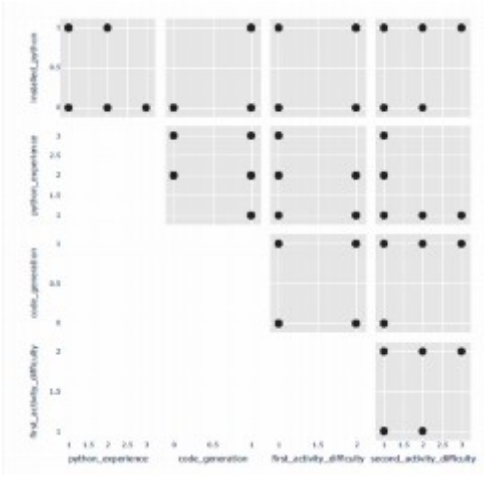


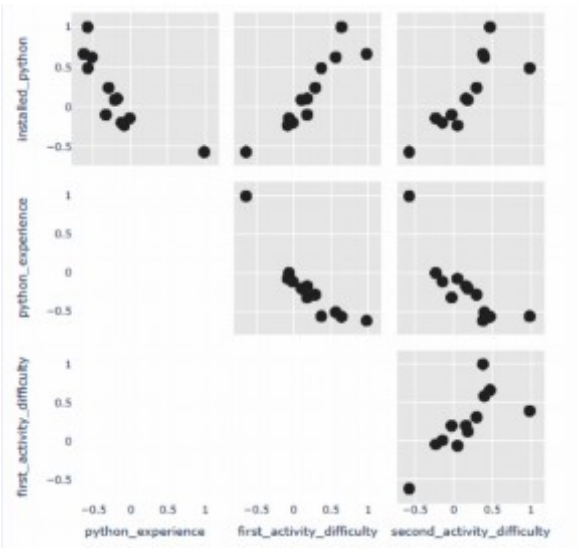
Рис. 3.6. Кореляція змінних

Ми можемо помітити, що найвища кореляція спостерігається між досвідом python та інсталяцією щодо передбачуваної складності діяльності зі значенням -0,6. Ця негативна кореляція, як і очікувалося, свідчить про те, що користувачі, які мають менший досвід роботи з Python, вважали дії більш

складними, ніж користувачі з більшим досвідом роботи з Python. Інша сильна кореляція відбулася між встановленням Python і генерацією коду зі значенням +0,6. Це означає, що користувачі без інстальованого Python на своїй машині мали більше проблем із завершенням створення коду, ніж користувачі з інстальованим Python. Очікується, що люди, у яких встановлено Python, частіше користуватимуться Python і будуть краще знайомі з кодуванням. Щоб дослідити це більш детально, ми створили діаграму розсіювання (рис. 3.7).



а) Діаграма розсіювання одиничних значень



б) Матриця кореляції розкиду python_installed, python_experience та складність діяльності

Рис. 3.7. Матриця розкиду найсильніше корельованих змінних

На рисунку 3.7(а) ми можемо побачити появу комбінацій значень - більш корельованих змінних. Наприклад, у випадку змінних code_generation і python_installed ми бачимо, що комбінація python_installed = 0 і code_generation = 1 не відбулася для жодного з учасників. Тому всі учасники, у яких не було встановлено python, відчувають труднощі з генерацією коду. Більше того, ми бачимо, що всі учасники, які мають досвід роботи з Python більше одного року, вважали другу вправу дуже легкою для виконання. Лише користувачі, які стверджували, що мають менше року досвіду роботи з Python, вважали другу дію складнішою. Крім того, усі учасники, які знайшли другу діяльність із помірним або високим рівнем складності, мали проблеми під час генерації коду. Тому всі учасники, які не мали труднощів із генерацією коду, вважали друге завдання дуже легким для виконання.

На рисунку 3.7(б) представлено кореляційну матрицю між python_installed, python_experience, first_activity_difficulty і second_activity_ difficulty. Як і очікувалося, ми можемо спостерігати позитивну лінійну кореляцію між installed_python і складністю діяльності. Існує негативна лінійна кореляція між python_experience і складністю діяльності. Ці кореляції узгоджуються з результатами матриці розсіювання. Крім того, ми можемо помітити, що існує сильна кореляція між учасниками, які стверджували, що мають більше ніж один рік досвіду роботи з Python, і учасниками, які повинні були встановити Python для дослідження.

Ми відфільтрували зібраний набір даних за змінною python_experience, щоб зберегти учасників із досвідом роботи з python більше одного року. Ми вважаємо, що користувачі з більш ніж однорічним досвідом роботи з Python навчилися з першого завдання та вважали, що друге легше виконати. Зауважте, що чотири учасники з досвідом роботи на Python більше одного року визнали перше завдання середньої складності, а всі учасники вважали друге завдання дуже легким для виконання. Однак ми очікували, що все буде інакше, оскільки перше завдання було схоже на приклад підручника, а друге було далі від підручника.

3.2.2. Обговорення результатів

У цьому підрозділі ми спочатку описуємо деякі повторювані сценарії під час застосування дослідження, а потім представляємо деякий аналіз зібраних змінних.

Декілька учасників пропустили генерацію коду для команд або вставили приклад генерації коду підручника. Деякі учасники неправильно зрозуміли мету дослідження. Інші не вважали за потрібне завершувати заходи. У учасників може виникнути помилкове відчуття завершення, якщо вони пропустять генерацію коду, оскільки бот виконує команди та показує результати. Однак структура не генерувала код для цих команд, і це може вплинути на майбутні блоки генерації коду. Майже всі учасники залишили суперклас ColumnNameCommand для своїх команд і отримали помилку виконання під час тестів. Усі учасники швидко зрозуміли та виправили помилку.

Декілька учасників припустили, що немає необхідності імпортувати свої пакети в свої команди. Для другої команди потрібна бібліотека numpy, і деякі учасники стверджували, що фреймворк уже завантажив бібліотеку numpy. Ця помилка може виникати через те, що бібліотеки numpy та pandas часто з’являються разом, а фреймворк використовує бібліотеку pandas. Отже, учасники могли використовувати об’єкт фрейму даних pandas без імпорту бібліотеки pandas. Деякі користувачі сказали: «У мене вже є об’єкт dataframe, і я не імпортував бібліотеку pandas». Параметр force_training визначає, чи може фреймворк використовувати старі серіалізовані файли, чи він повинен навчити нового чат-бота. Семеро учасників залишили для параметра force_training значення true навіть під час завантаження старого чат-бота для перевірки кількох ідей. Цей параметр впливав лише на час завантаження чат-бота при кожному виконанні. Однак, тренуючи важкі моделі, тренування слід виконувати лише на вимогу. Майже кожен учасник використовував свої команди без вказівки значення набору даних. Для виконання всіх команд у посібнику та діях потрібен аргумент набору даних. Однак майже ніхто не

усвідомлював, що вони не вказували значення набору даних, а фреймворк виконував дію правильно. Фреймворк розв’язав останній набір даних у діалозі та використав цей набір даних для завершення дії. Тим не менш, учасники не знали про це, і ніхто з них не ставив під сумнів поведінку рамок. Деякі учасники стверджували: «Природно, структура використовує набір даних, про який ми говоримо».

Деякі учасники запропонували, щоб фреймворк мав функцію, яка отримує всі гарячі точки та створює внутрішній клас. Цей підхід повинен усунути залежність від об’єктно-орієнтованого програмування (ООП) і зменшити кількість коду, необхідного для розширення структури . Однак це буде більш загадковим. Інші учасники запропонували, щоб бот підтримував завантажений об’єкт через декілька виконання. Цей випадок стався, коли учасники спробували використати набір даних зі старих розмов.

У цій роботі описано структуру для автоматизації завдань науки про дані за допомогою персоналізованих чат-ботів. Для цього ми створили фреймворк у сценарії виявлення викидів. Ми також провели дослідження користувачів, щоб продемонструвати використання інструменту.

Основні результати даної роботи наступні. По-перше, був запропонований робочий процес для досягнення рівня автоматизації L2. Ми досягли своєї мети завдяки розмовним інтерфейсам і персоналізованим пропозиціям. По-друге, представлена структура для автоматизації деяких процесів DS/ML.

Висновки до розділу

У третьому розділі роботи проведено оцінку запропонованого фреймворку автоматизації задач Data Science через практичне застосування до різних типів задач, зокрема виявлення аномалій та очищення даних. Цей підхід дозволив продемонструвати універсальність і ефективність

фреймворку, а також оцінити його здатність адаптуватися до потреб спеціалістів у галузі Data Science.

На етапі виявлення аномалій фреймворк успішно застосував методи аналізу даних для ідентифікації невідповідностей у вхідних наборах даних. Цей процес показав високу точність і швидкість виявлення відхилень, що робить його корисним для виявлення ризиків або аномальних подій у великих масивах даних.

У підрозділі, присвяченому очищенню даних, фреймворк продемонстрував здатність ефективно усувати пропуски, дублікати та інші помилки, які є типовими для "брудних" даних. Результати цього кейсу підкреслили значний потенціал розробленого інструменту в забезпеченні якості даних, що є критично важливим для подальшого аналізу.

Розглянувши результати використання фреймворку, встановлено, що запропоноване рішення дозволяє не тільки автоматизувати рутинні завдання Data Science, але й забезпечує масштабованість та інтеграцію з іншими інструментами. Отримані результати демонструють, що фреймворк підходить як для початківців, так і для досвідчених спеціалістів у сфері аналізу даних.

ВИСНОВКИ

Магістерська робота присвячена дослідженню підходів автоматизації задач Data Science та створенню персоналізованих рішень за допомогою розмовних інтерфейсів. У межах дослідження виконано аналіз предметної області, розроблено та оцінено власне рішення, яке базується на інтеграції сучасних концепцій автоматизованого машинного навчання (AutoML) і методологій створення чат-ботів.

У першому розділі роботи виконано глибокий аналіз стану автоматизації в Data Science. Визначено, що, попри прогрес у розробці інструментів AutoML, існує низка проблем, пов’язаних із недостатньою персоналізацією, складністю інтеграції та обмеженнями в адаптації під потреби користувачів. Запропонована методологія дослідження ґрунтується на поєднанні технологій AutoML та чат-ботів, які здатні автоматизувати не лише технічні, але й комунікаційні процеси.

У другому розділі представлено архітектуру та функціональні модулі розробленого рішення. Система базується на чат-боті з інтеграцією в Telegram, який може генерувати персоналізований контент, виконувати завдання Data Science, ідентифікувати наміри користувачів та надавати індивідуальні рекомендації. Зокрема, реалізовано такі модулі, як менеджер завдань, функція генерації коду, ієрархія команд, деревовидна структура даних мовлення, а також модуль персоналізації контенту. Це дозволило створити систему, здатну адаптуватися до специфічних потреб користувачів і забезпечити ефективну комунікацію.

У третьому розділі проведено оцінку запропонованого фреймворку на практичних задачах. Тестування включало виявлення аномалій та очищення даних, що продемонструвало його універсальність і високу точність. Результати використання підтвердили ефективність автоматизації задач Data Science та скорочення часу їх виконання. У процесі обговорення виявлено,

що якість результатів суттєво залежить від вхідних даних, проте гнучкість налаштувань дозволяє компенсувати ці обмеження.

Запропоноване рішення успішно поєднує технології автоматизації та персоналізації, забезпечуючи інтуїтивний інтерфейс і широкий функціонал. Воно має потенціал для застосування в практичних сценаріях завдяки здатності масштабуватися та адаптуватися до потреб користувачів. Робота робить внесок у розвиток інструментів автоматизації Data Science, водночас відкриваючи перспективи для подальших досліджень у напрямку інтеграції більш складних сценаріїв персоналізації та розширення функціональності розмовних агентів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Abdellatif, A., Costa, D., Badran, K., Abdalkareem, R., and Shihab, E. (2020). Challenges in chatbot development: A study of stack overflow posts.
2. Alemu, E. N. and Huang, J. (2020). Healthaid: Extracting domain targeted high precision procedural knowledge from on-line communities. Information Processing & Management, 57(6):102299.
3. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
4. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.
5. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
7. Chollet, F. (2018). Deep Learning with Python. Manning Publications.
8. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson.
9. Mitchell, T. (1997). Machine Learning. McGraw-Hill Education.
10. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.
11. VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media.
12. McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.
13. Provost, F., & Fawcett, T. (2013). Data Science for Business. O'Reilly Media.
14. Alpaydin, E. (2020). Introduction to Machine Learning. MIT Press.
15. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

16. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer.
17. Aggarwal, C. C. (2015). Data Mining: The Textbook. Springer.
18. Abadi, M., et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.
19. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.
20. Chollet, F. (2015). Keras: The Python Deep Learning Library.
21. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature.
22. McAfee, A., & Brynjolfsson, E. (2017). Machine, Platform, Crowd: Harnessing Our Digital Future. Norton.
23. Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM.
24. Domingos, P. (2015). The Master Algorithm. Basic Books.
25. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System.
26. Silver, D., et al. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature.
27. Witten, I. H., Frank, E., & Hall, M. A. (2016). Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann.
28. Stone, P. (2016). Artificial Intelligence and Life in 2030.
29. Oliphant, T. E. (2006). A Guide to NumPy.
30. Baydin, A. G., et al. (2018). Automatic Differentiation in Machine Learning: A Survey. Journal of Machine Learning Research.
31. Jordan, M. I., & Mitchell, T. M. (2015). Machine Learning: Trends, Perspectives, and Prospects. Science.
32. Berthold, M. R. (2019). What does it take to be a successful data scientist? Harvard Data Science Review, 1(2).
33. Brandtzaeg, P. B. and Følstad, A. (2018). Chatbots: changing user needs and motivations. Interactions, 25(5):38–43.

34. Chandola, V., Banerjee, A., and Kumar, V. (2007). Outlier detection: A survey. ACM Computing Surveys, 14:15.
35. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems.
36. Brown, T., et al. (2020). Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems.
37. Chaves, A. and Gerosa, M. (2019). How should my chatbot interact? a survey on human-chatbot interaction design.(2019). arXiv preprint arXiv:1904.02743.
38. Breiman, L. (2001). Random Forests. Machine Learning Journal.
39. Vapnik, V. (1998). Statistical Learning Theory. Wiley.
40. Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research.
41. Chittò, P., Baez, M., Daniel, F., and Benatallah, B. (2020). Automatic generation of chatbots for conversational web browsing. In International Conference on Conceptual Modeling, pages 239–249. Springer.
42. Costa, C. and Santos, M. Y. (2017). A conceptual model for the professional profile of a data scientist. In World Conference on Information Systems and Technologies, pages 453–463. Springer.
43. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization.
44. Dey, S. and Zhang, P. (2019). Estimating personalized drug responses from real world evidence. US Patent App. 15/855,314.
45. Ed-douibi, H., Cánovas Izquierdo, J. L., Daniel, G., and Cabot, J. (2021). A model-based chatbot generation approach to converse with open data sources. In International Conference on Web Engineering, pages 440–455. Springer.
46. Fast, E., Chen, B., Mendelsohn, J., Bassen, J., and Bernstein, M. S. (2018).

47. Iris: A conversational agent for complex tasks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pages 1–12.
48. Gogoi, P., Bhattacharyya, D., Borah, B., and Kalita, J. K. (2011). A survey of outlier detection methods in network anomaly identification. The Computer Journal, 54(4):570–588.
49. Guyon, I., Bennett, K., Cawley, G., Escalante, H. J., Escalera, S., Ho, T. K., Macià, N., Ray, B., Saeed, M., Statnikov, A., et al. (2015). Design of the 2015 chalearn automl challenge. In 2015 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE.
50. Harris, H., Murphy, S., and Vaisman, M. (2013). Analyzing the analyzers: An introspective survey of data scientists and their work. " O'Reilly Media, Inc.".
51. He, X., Zhao, K., and Chu, X. (2019). Automl: A survey of the state-of-the-art. arXiv preprint arXiv:1908.00709.
52. Hellerstein, J. M. (2008). Quantitative data cleaning for large databases. United Nations Economic Commission for Europe (UNECE), 25.