Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Магістерська робота

освітній ступінь – магістр

на тему “**Аналіз BigData за допомогою методів машинного навчання**“

Виконав: студент 2-го року навчання,

Спеціальності

121 Інженерія програмного забезпечення

Фещенко Кирил Юрійович

Керівник Жежерун О.П.

кандидат наук, доцент

Рецензент _____

Магістерська робота захищена

з оцінкою _____

Секретар ЕК _____

“___” _____2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

Доктор фіз.-мат. наук,

_____ Гороховський С. С.

(підпис)

«____» _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломний проект

студенту 2 р. н. магістерської програми Інженерія програмного забезпечення

Фещенко Кирилу Юрійовичу

Дослідити Аналіз BigData за допомогою методів машинного навчання

Зміст текстової частини до магістерської роботи:

Перелік умовних позначень, Анотація, Вступ, Огляд предметної області, Аналіз архітектурних та технічних рішень, Реалізація системи обробки повідомлень, Висновки, Список використаних джерел, Додатки

Дата видачі: “ ”_____ 2024

Керівник:

к.н., доцент Жежерун О.П._____

Завдення отримав

Фещенко К.Ю._____

Тема: Аналіз BigData за допомогою методів машинного навчання

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1	Отримання завдання на дипломну роботу	28.10.2023	
2	Огляд технічних джерел	29.10.2023 - 29.11.2023	
3	Аналіз альтернативних рішень	30.11.2023 - 30.12.2023	
4	Розробка архітектури системи	31.12.2024 - 16.01.2024	
5	Проектування основних модулів	17.01.2024 - 01.02.2024	
6	Реалізація основних модулів	02.02.2024 - 28.02.2024	
7	Інтеграція модулів	29.02.2024 - 28.03.2024	
8	Інтеграційне тестування	29.03.2024 - 12.04.2024	
9	Рефакторинг	13.04.2024 - 27.04.2024	
10	Підготовка технічної документації	28.04.2024 - 10.05.2024	
11	Огляд та оновлення документації	11.05.2024 - 16.05.2024	
12	Попередній захист	17.05.2024	
13	Остаточний захист	10.06.2024	

Керівник:

к.н. доцент Жежерун О.П._____

Студент:

Фещенко К.Ю._____

Анотація

Дипломна робота робота на тему ‘Аналіз BigData за допомогою методів машинного навчання’ присвячена дослідженню проблеми обробки великої кількості даних у подійно-орієнтованій архітектурі. В даній роботі були використані найновіші інструменти для обробки BigData та наведені найбільш популярні методи вирішення задачі. В даній роботі імплементовано систему обробки повідомлень, згідно зі стандартами індустрії. Особлива увага приділена теоретичній алгоритмічній частині. Робота містить інструкції з проводження та використання системи.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Кафка – Apache Kafka

Івент – запис у топіку в системі Apache Kafka

Процессінг – обробка даних

Зтригеритися - зреагувати на отримання повідомлення з топіку

Топік – місце зберігання повідомлень у системі Apache kafka

Хайлоуд – робота системи під високим навантаженням

МЛ – машинне навчання

Event-driven архітектура – подійно-орієнтована архітектура

ВСТУП

Актуальність.

В сучасному світі, обробка та аналіз великих обсягів даних (Big Data) стає все більш актуальною та необхідною для розвитку різноманітних галузей промисловості та науки. Швидкий розвиток інформаційних технологій, зростання кількості доступної інформації в Інтернеті та збільшення кількості сенсорів і пристроїв Інтернету речей (IoT) призвели до виникнення великих обсягів даних, які потребують ефективної обробки та аналізу. Особливо важливим стає завдання обробки потокових даних, які надходять у реальному часі та величезних обсягах.

У цьому контексті, методи машинного навчання (Machine Learning) виявляються дуже ефективними інструментами для аналізу та обробки великих обсягів даних. Вони дозволяють автоматизувати процеси обробки даних, виявляти закономірності та залежності у них, роблячи цінні висновки та прогнози. Одним з таких методів є аналіз потокових даних за допомогою класифікаторів, який дозволяє автоматично відбирати та обробляти вхідні дані в реальному часі.

У даній роботі буде розглянуто використання методів машинного навчання для аналізу потокових даних з використанням Kafka як системи обміну повідомленнями. Висвітлено методи та техніки, що дозволяють ефективно відбирати та аналізувати вхідні дані, використовуючи класифікатори та різноманітні алгоритми машинного навчання. Розглянуті практичні приклади та результати використання запропонованих методів у реальних умовах.

Мета дослідження.

Метою даного дослідження є розробка та дослідження методів аналізу потокових даних за допомогою методів машинного навчання у системі Kafka. Конкретні цілі дослідження включають:

1. Розробка системи збору та обробки потокових даних з використанням Kafka як системи обміну повідомленнями.
2. Використання класифікаторів та алгоритмів машинного навчання для автоматичного відбору та аналізу вхідних даних.

3. Визначення ефективних методів виявлення та класифікації даних у реальному часі.
4. Проведення експериментів з різними моделями машинного навчання для оцінки їхньої ефективності та точності.
5. Визначення можливостей та обмежень використання методів машинного навчання для аналізу потокових даних у системі Kafka.

Мета цього дослідження полягає у виявленні оптимальних підходів до аналізу потокових даних з використанням методів машинного навчання в системі Kafka та визначенні їхнього потенціалу для практичного застосування.

Завдання дослідження.

1. Розробка системи збору потокових даних: реалізація механізму зчитування даних з Kafka та їхнього подальшого перетворення та обробки.
2. Вибір та налаштування класифікаторів: визначення найбільш ефективних алгоритмів машинного навчання для класифікації даних в системі Kafka.
3. Розробка методів аналізу даних: реалізація алгоритмів для виявлення аномалій, визначення патернів та інших особливостей у потокових даних.
4. Експерименти та оцінка ефективності: проведення серії експериментів для порівняння різних методів аналізу даних та визначення їхньої ефективності та точності.
5. Вдосконалення та оптимізація: виявлення можливостей для вдосконалення та оптимізації розроблених методів та алгоритмів для забезпечення їхньої оптимальної роботи.
6. Аналіз результатів та висновки: обробка та аналіз отриманих даних, формулювання висновків та рекомендацій для подальшого використання методів машинного навчання для аналізу потокових даних у системі Kafka.

Об’єкт дослідження:

Об'єктом дослідження є потокові дані, які надходять у реальному часі до системи Kafka. Ці дані можуть бути різної природи, такої як логи, метрики,

транзакції тощо, та мають великий обсяг та високу швидкість зміни. Основною метою дослідження є розробка та застосування методів машинного навчання для аналізу цих потокових даних з метою виявлення аномалій, патернів та інших цікавих особливостей. Об'єкт дослідження також включає в себе розроблену систему збору та обробки даних, яка забезпечує стабільну та ефективну роботу з потоковими даними в середовищі Kafka.

Предмет дослідження:

Предметом дослідження є методи машинного навчання та їх застосування для аналізу потокових даних у системі Kafka. Конкретно, досліджується ефективність та точність різних алгоритмів класифікації, кластеризації та аналізу аномалій в потоках даних. Предметом дослідження є також розробка та оптимізація системи обробки потокових даних для забезпечення швидкості, масштабованості та надійності в реальному часі. Важливим аспектом є також вивчення впливу різноманітних факторів, таких як обсяг даних, швидкість потоку, складність даних та інші, на ефективність застосованих методів машинного навчання. Використання методів машинного навчання дозволить покращити ефективність обробки даних, шляхом автоматичного виявлення та класифікації повідомлень. В результаті, знижуються витрати часу та ресурсів на обробку даних, що сприяє підвищенню продуктивності та точності аналізу.

Джерела дослідження:

Веб-сайти компаній, що пропонують автоматизовані системи, форуми, друковані видання, матеріали конференцій, довідкова література, а також вихідний код програм та бібліотеки.

Наукова новизна полягає у використанні методів машинного навчання для аналізу великих обсягів даних, що надходять з системи консьюмерів

Кафки. Цей підхід дозволяє автоматизувати процес обробки та аналізу даних, швидко виявляти та класифікувати необхідні повідомлення, що має важливе значення для підтримки ефективної роботи бізнесу та прийняття обґрунтованих рішень.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Огляд предметної області включає в себе комплексний аналіз та оптимізацію обробки великого обсягу даних за допомогою методів машинного навчання. Предметна область охоплює терміни, концепції, процеси та бізнес-аспекти, що пов'язані з аналізом та обробкою даних в режимі реального часу.

В контексті розробки програмного забезпечення, предметна область включає в себе такі аспекти:

1. Основні терміни та концепції, пов'язані з аналізом даних та методами машинного навчання.
2. Структура та відношення між різними об'єктами та процесами в області аналізу даних.
3. Бізнес-правила та обмеження, які впливають на функціонування системи аналізу даних.
4. Сценарії використання системи аналізу даних, які відображають різні аспекти діяльності користувачів у контексті великого обсягу даних.

Для глибокого розуміння предметної області, розробники співпрацюють з експертами з області аналізу даних, вивчають наукову літературу та аналізують існуючі рішення та методики в цій галузі. Це допомагає виявити та проаналізувати ключові аспекти аналізу та обробки даних, що дозволяє ефективно розробляти та впроваджувати програмні рішення в області аналізу BigData.

1.1 Теоретичні відомості

Event-Driven архітектура стає все більш популярною у сфері розробки програмного забезпечення, оскільки вона дозволяє ефективно та швидко реагувати на події та зміни у системі. Одним із ключових аспектів цієї архітектури є обробка івентів, які можуть бути пушнуті у реальному часі в кафку або інший системи повідомлень.

Event-Driven системи базуються на концепції подій (івентів), які виникають у системі та спричиняють зміни у стані системи або викликають реакції в програмному забезпеченні. Ці події можуть бути різноманітні - від кліків користувачів на веб-сторінці до змін у базі даних або запуску додатку.

Одним з ключових викликів Event-Driven систем є швидка та ефективна обробка івентів у реальному часі. Це означає, що система повинна бути здатна реагувати на івенти миттєво та без затримок, що може бути складною задачею, особливо при великому обсязі даних.

У деяких випадках івенти можуть бути неоднозначними або непередбачуваними, і не завжди очевидно, як їх потрібно обробляти. В таких ситуаціях використання методів машинного навчання може бути корисним. За допомогою моделей класифікації можна навчити систему визначати, чи потрібно обробляти конкретний івент та як саме його опрацьовувати.

Існують певні переваги та виклики використання машинного навчання у системах обробки івентів. До переваг відносяться можливість автоматизації процесу обробки івентів, покращення швидкості та ефективності обробки, а також можливість реагувати на складні та непередбачувані сценарії. Однак використання машинного навчання може ставити виклики щодо необхідності навчання моделей на великих обсягах даних, потреби у великій обчислювальній потужності та складності налаштування та підтримки моделей.

Застосування методів машинного навчання для обробки івентів у Event-Driven системах відкриває нові можливості для автоматизації та оптимізації процесів обробки даних у реальному часі. Однак це вимагає глибокого розуміння як технологій Event-Driven систем, так і методів машинного навчання, а також уваги до викликів та обмежень цих підходів.

Event-Driven архітектура

є парадигмою, яка набула значного розповсюдження в останні роки через здатність вирішувати виклики масштабованості та реактивності в сучасних програмних системах. У її основі лежить концепція подій, які відображають значущі події або зміни стану в системі. Ці події захоплюються, обробляються та реагуються в реальному часі, що дозволяє системам динамічно реагувати на змінні умови та вимоги.

Однією з ключових переваг є вбудована масштабованість. Шляхом роз'єднання компонентів та сервісів за допомогою подій, архітектура дозволяє більшій гнучкості та агільності в проектуванні системи. Це роз'єднання означає, що окремі компоненти можуть масштабуватися незалежно, що дозволяє системам ефективніше вирішувати різні завдання. Крім того, сприяє вільному зв'язку між

сервісами, що спрощує модифікацію та розширення систем без впливу на інші компоненти.

Іншою перевагою є здатність покращити реактивність системи. Шляхом обробки подій асинхронно, системи можуть реагувати на події практично в реальному часі, надаючи користувачам швидку зворотню інформацію та зменшуючи затримки. Ця реактивність особливо цінна в сценаріях, де часове прийняття рішень є критичним, таких як фінансові торговельні платформи або системи реального часу.

Більше того, архітектура дозволяє системам бути більш стійкими та стійкими до відмов. Шляхом розподілу логіки обробки по всіх компонентах, архітектури ПОА можуть продовжувати функціонувати навіть у випадку відмов компонентів або відмов мережі. Ця стійкість до відмов досягається за допомогою механізмів, таких як буферизація подій, реплікація та стратегії повтору, які забезпечують надійну та послідовну обробку подій.

Отже, подійно-орієнтована архітектура пропонує потужний фреймворк для побудови масштабованих, реактивних та стійких програмних систем. Шляхом прийняття принципів вільного зв'язку, асинхронної обробки та стійкості до відмов, ПОА дозволяє організаціям проектувати та розгортати системи, які можуть адаптуватися до змінних вимог та надавати користувачам значну користь. Оскільки попит на обробку даних в реальному часі та інтелектуальне прийняття рішень продовжує зростати, Подійно-орієнтована архітектура готова відігравати все більш важливу роль у формуванні майбутнього розробки програмного забезпечення.

Обробка повідомлень у event-driven архітектурі

є ключовим аспектом, який дозволяє системам ефективно реагувати на події та зміни стану. Цей процес полягає у виявленні, обробці та відповіді на різноманітні події, що відбуваються у системі або надходять ззовні.

Першим етапом обробки повідомлень є виявлення та отримання подій. Це може включати прийом подій з різних джерел, таких як вхідні потоки даних, зовнішні інтеграції або інші компоненти системи. Для цього часто використовуються

асинхронні механізми передачі повідомлень, такі як черги подій або брокери повідомлень, які дозволяють ефективно передавати та приймати великі обсяги даних.

Після отримання подій вони піддаються обробці, що може включати аналіз та інтерпретацію їх змісту, перевірку на валідність та вирішення, чи потрібно їх подальша обробка. Важливою частиною цього процесу є класифікація подій та прийняття рішення про те, як їх обробляти. Тут можуть використовуватися методи машинного навчання для автоматизації цього процесу та підвищення ефективності.

Коли події класифіковані та оброблені, вони можуть бути надіслані наступним крокам обробки або взаємодії з іншими компонентами системи. Це може включати виклик інших сервісів чи системних функцій для виконання певних дій на основі отриманих даних.

Таким чином, обробка повідомлень у подійно-орієнтованій архітектурі дозволяє системам ефективно реагувати на зміни в середовищі та взаємодіяти з ними за допомогою асинхронного та гнучкого підходу. Цей процес є ключовим для побудови високопродуктивних та реактивних систем, які можуть ефективно вирішувати різноманітні завдання та задачі.

Задача классифікації

у машинному навчанні полягає в призначенні кожному елементу вхідного набору даних однієї або декількох категорій або класів на основі його характеристик чи ознак. Це важлива задача, яка знаходить застосування у багатьох галузях, включаючи розпізнавання образів, аналіз тексту, медичну діагностику та багато інших.

У контексті подійно-орієнтованої архітектури задача класифікації може бути використана для визначення того, які події потрібно обробляти та як їх обробляти. Наприклад, на основі аналізу характеристик події, таких як тип, зміст чи джерело, можна вирішити, чи вона є важливою та потребує обробки, або може бути проігнорована.

Для вирішення задачі класифікації у машинному навчанні застосовуються різноманітні алгоритми, такі як лінійна регресія, дерева рішень, метод опорних векторів та нейронні мережі. Вони навчаються на підготовленому наборі даних, де кожний зразок має відомий клас чи категорію, і потім застосовуються до нових даних для прогнозування їх класу.

У контексті подійно-орієнтованої архітектури, для вирішення задачі класифікації можна навчити модель на підготовлених даних про події та їх характеристики. Наприклад, модель може визначати, чи є певна подія критичною для системи та потребує негайної обробки, чи може бути відтермінована для обробки в майбутньому. Використання машинного навчання для розв'язання задачі класифікації дозволяє автоматизувати процес прийняття рішень та забезпечити ефективне управління подіями у подійно-орієнтованих системах.

1.2 Огляд існуючих систем для обробки повідомлень

Подібній підхід до обробки повідомлень за допомогою машинного навчання можна розглядати в контексті альтернативних систем, які також працюють з шиною данних. Ось декілька можливих альтернатив:

1. Правила на базі даних: Одним з альтернативних підходів до обробки повідомлень може бути використання правил на базі даних. В цьому випадку, замість використання машинного навчання для класифікації повідомлень, ви можете налаштовувати правила в базі даних, які визначають, які повідомлення слід обробляти та як їх обробляти.
2. Правила на базі потоків даних: Іншою альтернативою може бути використання потокових обробників даних, таких як Apache NiFi або StreamSets. Ці інструменти дозволяють вам створювати потокові обробники даних, які можуть перевіряти умови та застосовувати дії до вхідних даних на основі цих умов.
3. Потужність спеціалізованих систем: Деякі спеціалізовані системи для обробки повідомлень, такі як Apache Kafka з Apache Flink або Apache Kafka з Apache Spark, можуть включати в себе розширені можливості обробки даних за допомогою машинного навчання. Наприклад, ви можете використовувати моделі машинного навчання в Apache Flink для аналізу потокових даних та прийняття рішень на основі цього аналізу.
4. Інтеграція з ML-платформами: Деякі платформи машинного навчання, такі як TensorFlow або PyTorch, можуть мати інтеграцію з Apache Kafka або іншими системами потокового обробки даних. Це дозволяє вам використовувати моделі машинного навчання для аналізу та обробки вхідних даних з Kafka у реальному часі.

Виходячи з популярності інструментів та простоти налаштування найбільш вірогідними кандидатами для вирішення такоєї задачі є:

Apache Flink

це потужна та розширювана система потокової обробки даних, яка дозволяє аналізувати та обробляти великі обсяги даних у реальному часі. У контексті обробки повідомлень з Apache Kafka, Apache Flink може бути використаний для ефективного аналізу потокових даних та прийняття рішень на основі цього аналізу.

Ось деякі ключові переваги використання Apache Flink для обробки повідомлень з Apache Kafka:

1. Низька затримка: Apache Flink пропонує низьку затримку обробки даних, що дозволяє аналізувати та відповідати на повідомлення з Kafka майже в реальному часі. Це особливо важливо для додатків, які вимагають швидкої реакції на події.
2. Скальованість: Apache Flink може ефективно масштабуватись горизонтально, що дозволяє обробляти великі обсяги даних та забезпечує високу доступність системи.
3. Підтримка ML: Apache Flink має вбудовану підтримку машинного навчання, що дозволяє вам використовувати моделі машинного навчання для аналізу потокових даних та прийняття рішень на основі цього аналізу. Це особливо корисно в контексті вашої задачі, де необхідно класифікувати повідомлення та вирішувати, як їх обробляти.
4. Широкий функціонал: Apache Flink надає багато функціоналу для обробки потокових даних, включаючи віконні операції, агрегацію, обробку подій у вікні та інші. Це дозволяє ефективно аналізувати та обробляти дані з Kafka за допомогою різних технік.
5. Інтеграція з Kafka: Apache Flink має вбудовану підтримку Apache Kafka, що спрощує його інтеграцію з вашою системою Kafka та дозволяє легко отримувати дані з Kafka для подальшої обробки.

Загалом, використання Apache Flink для обробки повідомлень з Apache Kafka може допомогти вам ефективно аналізувати та обробляти потокові дані у

реальному часі, використовуючи різноманітні техніки та функціонал, що пропонує Apache Flink.

Apache Spark

це потужна та розподілена система обробки даних, яка надає широкі можливості для аналізу, обробки та обчислення великих обсягів даних у реальному часі. У контексті обробки повідомлень з Apache Kafka, Apache Spark може бути використаний для ефективної обробки потокових даних та аналізу їх з використанням різноманітних методів та алгоритмів.

Ось деякі ключові переваги використання Apache Spark для обробки повідомлень з Apache Kafka:

1. Висока швидкодія: Apache Spark пропонує високу швидкодію обробки даних, що дозволяє ефективно аналізувати та обробляти потокові дані у реальному часі. Використання розподілених обчислень та оптимізацій пам'яті дозволяє досягти високої продуктивності навіть при великому обсязі даних.
2. Багатофункціональність: Apache Spark має широкий набір функцій та бібліотек для обробки даних, включаючи підтримку SQL-запитів, машинного навчання, графових алгоритмів та інших. Це дозволяє вам ефективно аналізувати та обробляти дані з Apache Kafka за допомогою різних методів та алгоритмів.
3. Машинне навчання: Apache Spark має вбудовану підтримку машинного навчання, що дозволяє використовувати моделі машинного навчання для аналізу потокових даних та прийняття рішень на основі цього аналізу. Це особливо корисно в контексті вашої задачі, де потрібно класифікувати повідомлення та приймати рішення щодо їх обробки.
4. Легка інтеграція з Kafka: Apache Spark має вбудовану підтримку Apache Kafka, що спрощує інтеграцію з вашою системою Kafka та дозволяє легко отримувати дані з Kafka для подальшої обробки.
5. Скальованість: Apache Spark може ефективно масштабуватись горизонтально, що дозволяє обробляти великі обсяги даних та забезпечує високу доступність системи.

Загалом, використання Apache Spark для обробки повідомлень з Apache Kafka може допомогти вам ефективно аналізувати та обробляти потокові дані у

реальному часі, використовуючи різноманітні методи та алгоритми, що пропонує Apache Spark.

Apache NiFi
це потужна та гнучка система для обробки та пересилання даних в реальному часі. Вона розроблена для ефективного керування потоками даних, включаючи збирання, маршрутизацію, трансформацію та надсилання даних між різними джерелами та призначеннями.

У контексті обробки повідомлень з Apache Kafka, Apache NiFi може бути використаний для реалізації комплексних потоків обробки даних. Ось деякі ключові переваги використання Apache NiFi:

1. Гнучкість та розширюваність: Apache NiFi надає гнучкі можливості для налаштування потоків даних згідно зі специфічними потребами вашої системи. Вона має велику кількість готових компонентів та процесорів, які можна легко поєднати для створення складних потоків обробки даних.
2. Підтримка багатьох джерел та призначень: Apache NiFi підтримує різноманітні джерела даних, включаючи Apache Kafka, бази даних, файлові системи, API та інші. Ви можете легко інтегрувати різні джерела даних та призначення в один потік обробки даних.
3. Моніторинг та керування: Apache NiFi надає інтерфейс для моніторингу та керування потоками даних в реальному часі. Ви можете відстежувати стан та продуктивність вашого потоку даних, а також вносити зміни у реальному часі.
4. Безпека: Apache NiFi має різноманітні можливості безпеки, включаючи аутентифікацію, авторизацію та шифрування даних. Ви можете забезпечити безпеку вашого потоку даних та захистити дані від несанкціонованого доступу.
5. Легка інтеграція з Kafka: Apache NiFi має вбудовану підтримку Apache Kafka, що спрощує інтеграцію з вашою системою Kafka та дозволяє легко отримувати дані з Kafka для подальшої обробки.

Загалом, використання Apache NiFi для обробки повідомлень з Apache Kafka може допомогти вам створити потужні та гнучкі потоки обробки даних, що відповідають потребам вашої системи.

Таблиця 1 – Характеристики програмних інструментів для обробки повідомлень

Інструмент	Інтеграція з Apache Kafka	Скальованість	Підтримка ML	Обробка подій у реальному часі
Apache Flink	Так	Так	Так	Так
Apache Spark	Так	Так	Так	Ні
Apache Nifi	Так	Ні	Ні	Ні

Підсумовуючи.

можна сказати що було розглянуто різні інструменти для обробки повідомлень в реальному часі з використанням методів машинного навчання. Apache Flink, Apache Spark та Apache NiFi виявилися потужними інструментами з великим функціоналом та можливостями. Вони надають широкі можливості для обробки потокових даних та інтеграції з Apache Kafka.

Проте використання цих інструментів вимагає значного налаштування та кастомізації для вирішення конкретних завдань обробки повідомлень за допомогою методів машинного навчання. Наявність системи, яка була би легко розгорталась та мала можливість підтримувати різні сценарії обробки повідомлень з використанням методів машинного навчання, стала би великим покроком у напрямку розробки більш ефективних та гнучких систем обробки даних в реальному часі.

Таким чином, подальші дослідження та розробка систем, які поєднують в собі потужність та гнучкість інструментів обробки даних з методами машинного навчання, мають великий потенціал для вирішення складних завдань аналізу даних в реальному часі.

1.3 Постановка задачі

Призначення системи: інформаційна та програмна підтримка процесу обробки повідомлень з нечітким крітерієм типу повідомлення за умови задавання довільного сценарію обробки.

Цілі створення системи: оптимізація часу та ресурсів необхідних для реалізації обробки повідомлень з шини даних з простою конфігурацією бажаного процесу обробки повідолень.

Для досягнення поставлених цілей програмна система має реалізувати такі задачі:

1. Налаштування інтеграції з шиною данних
2. Збереження конфігурації процесу обробку повідомлень
3. Тренування моделі критерія прийняття рішень за допомогою методів машинного навчання
4. Обробки усіх повідомленнь з шини данних
5. Виконання запрограмованого сценарію для певних типів повідомлень

Постановка задачі для цієї системи полягає в розробці програмного забезпечення, яке забезпечить оптимізацію обробки повідомлень з шини даних з урахуванням умов та критеріїв, заданих користувачем. Головними цілями є забезпечення ефективного та швидкого оброблення повідомлень, враховуючи їх тип та характеристики, а також використання методів машинного навчання для автоматизації процесу прийняття рішень. Система повинна бути гнучкою та легко налаштовуватися, щоб задовольняти різноманітні вимоги та потреби користувачів. Крім того, важливо забезпечити надійність та безпеку обробки даних, а також зручний інтерфейс для користувачів, що дозволить легко керувати та моніторити процеси роботи системи.

Функціональна структура системи, що розроблятиметься, подана IDEF0 діаграмами за методологією SADT (Structured Analysis and Design Technique). На рис.1. подана контекстна діаграма.

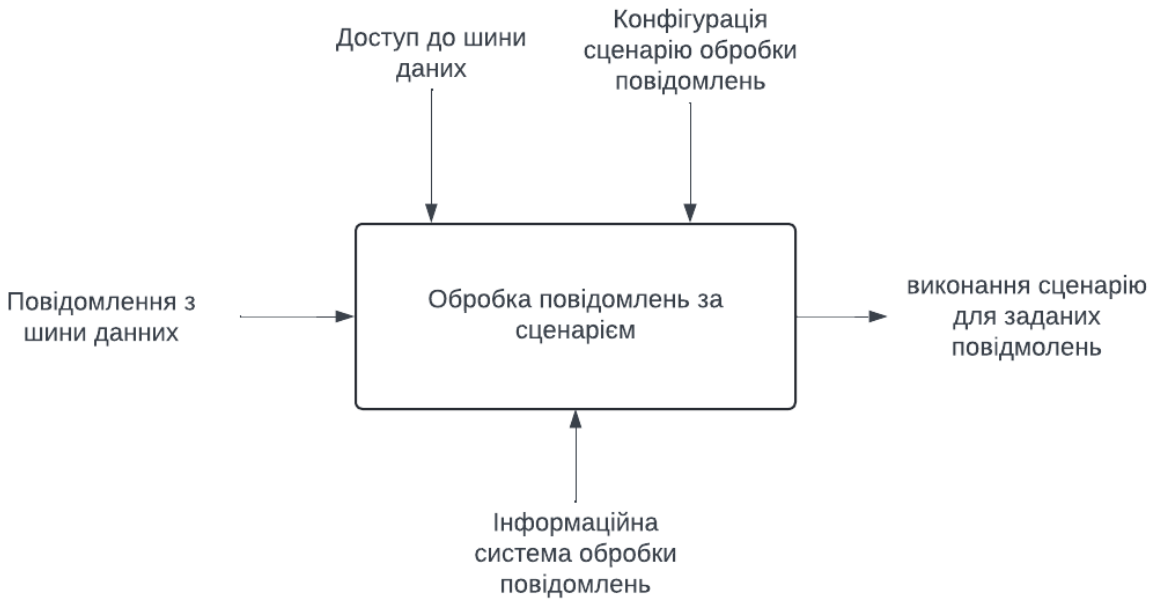

Рисунок 1 – IDEF0 контекстна діаграма інформаційної системи обробки повідомлень

1.4 Математична формалізація задачі

Математична формалізація задачі полягає в моделюванні процесу обробки повідомлень з нечітким крітерієм типу повідомлення за умови задавання довільного сценарію обробки. Для цього можна використовувати методи теорії масового обслуговування та імовірнісних моделей.

1.4.1 Математична модель задачі

Нехай X - множина усіх можливих повідомлень, T - множина усіх можливих типів повідомлень, C - множина усіх можливих умов обробки повідомлень, а S - множина усіх можливих сценаріїв обробки повідомлень. Для кожного повідомлення $x \in X$, типу $t \in T$, умови $c \in C$, існує сценарій $s \in S$, який визначає дії, що потрібно виконати з цим повідомленням.

Нехай $P(x,t,c,s)$ - ймовірність того, що повідомлення x типу t, що відповідає умовам c, буде оброблено сценарієм s. Метою системи є максимізація функції корисності U, яка враховує ефективність обробки повідомлень, з урахуванням обмежень ресурсів та вимог користувачів. Таким чином, задача може бути сформульована як задача оптимізації:

$$\max_{s \in S} U = \sum_{x \in X} \sum_{t \in T} \sum_{c \in C} P(x,t,c,s) \cdot U(x,t,c,s)$$

де $U(x,t,c,s)$ - функція корисності обробки повідомлення x типу t, що відповідає умовам c сценарієм s.

1.4.2 Методи машинного навчання, що можуть бути застосовані для оцінки співбесід

Для задачі обробки повідомлень з нечітким крітерієм типу повідомлення за умови задавання довільного сценарію обробки, можна використовувати різноманітні методи машинного навчання. Ось декілька можливих підходів:

1. Класифікація: За допомогою моделей класифікації можна визначати типи повідомлень на основі їх характеристик. Наприклад, можна використовувати моделі класифікації тексту для визначення теми або сутності повідомлення.

2. Кластеризація: Цей метод дозволяє групувати схожі повідомлення разом. За допомогою кластеризації можна виявити структури в наборі даних та визначити подібність між повідомленнями.
3. Асоціативні правила: Цей метод дозволяє виявляти часті комбінації у великих наборах даних. За допомогою аналізу асоціативних правил можна виявити зв'язки між різними типами повідомлень та умовами їх обробки.
4. Загальна залежність (Generalized Additive Models, GAM): Цей метод дозволяє моделювати залежності між наборами даних та визначити, як вони впливають на результати. GAM може бути корисним для аналізу взаємодії між типами повідомлень та умовами їх обробки.
5. Згорткові нейронні мережі (Convolutional Neural Networks, CNN): Цей тип нейронних мереж спеціалізується на обробці зображень або послідовностей даних з використанням згорткових шарів. CNN може бути застосований для виявлення паттернів у текстових або числових даних.
6. Рекурентні нейронні мережі (Recurrent Neural Networks, RNN): Ці мережі підходять для обробки послідовних даних, таких як текст або часові ряди. RNN може бути використана для аналізу послідовностей повідомлень та визначення їх зв'язків.

Ці методи машинного навчання можуть бути застосовані для вирішення вашої задачі обробки повідомлень з нечітким крітерієм типу повідомлення за умови задавання довільного сценарію обробки. Вибір конкретного методу залежить від характеру даних та постановки задачі.

Висновки розділу 1

У розділі 1 проаналізовано переваги використання системи обробки повідомлень з методами машинного навчання для автоматизації процесу обробки повідомлень з нечітким критерієм типу повідомлення. Визначено основні функції системи, які включають збереження конфігурації обробки повідомлень, тренування моделі критерія прийняття рішень та обробку усіх повідомлень з шини даних.

Для досягнення мети системи - оптимізації витрат часу та ресурсів на обробку повідомлень - важливо мати гнучку та адаптивну систему, яка забезпечить легкість у використанні та можливість інтеграції з іншими програмними засобами компанії. Крім того, необхідно забезпечити допустимий рівень безпеки та конфіденційності даних, щоб забезпечити захист інформації від несанкціонованого доступу.

Для забезпечення ефективної роботи системи рекомендується створити чіткий план конфігурації обробки повідомлень, який включає в себе як базові, так і кастомні сценарії обробки повідомлень, які відповідають потребам конкретного випадку використання. Такий підхід дозволить забезпечити ефективну та об’єктивну обробку повідомлень з нечітким критерієм типу повідомлення з урахуванням різних сценаріїв та потреб користувачів.

РОЗДІЛ 2. АНАЛІЗ АРХІТЕКТУРНИХ ТА ТЕХНІЧНИХ РІШЕНЬ

Сучасний розвиток технологій надає організаціям потужні інструменти для підвищення ефективності та конкурентоспроможності. Важливою складовою успішної інформаційної системи є обґрунтовані архітектурні та технічні рішення, що визначають структуру системи, взаємодію компонентів та відповідність вимогам замовника.

Мета даного розділу полягає в аналізі різноманітних архітектурних та технічних рішень з метою визначення найбільш ефективних та оптимальних з точки зору надійності, масштабованості та продуктивності. Вибрані рішення стануть основою для планування та розробки інформаційної системи, а також важливим етапом у формуванні проектної документації та визначенні технічних ресурсів.

У підсумку, аналіз архітектурних та технічних рішень є ключовим етапом у розробці інформаційної системи, спрямованим на вибір оптимальних шляхів досягнення мети проекту та забезпечення високої якості кінцевого продукту.

2.1 Огляд архітектурних підходів

Архітектура програмного забезпечення є основою будь-якої інформаційної системи, оскільки вона визначає структуру та взаємодію компонентів. Метою будь-якої архітектури є спрощення розробки, розгортання та підтримки програмної системи. У даному розділі ми розглянемо чотири основні підходи до архітектури програмного забезпечення: монолітну, мікросервісну, доменну та чисту архітектуру.

Монолітна архітектура представляє собою систему як єдиний модуль, що поставляється через єдине розгортання. Вона відзначається простотою розробки та розгортання, але може бути складно масштабованою та повільно оновлюватись. Мікросервісна архітектура, натомість, складається з невеликих незалежних служб, що мають свої точки розгортання. Вона забезпечує більшу гнучкість та масштабованість, але може бути складною у впровадженні та управлінні.

Доменна архітектура зосереджується на бізнес-логіці та доменних правилах, що сприяє створенню модульної структури з незалежними компонентами. Вона підвищує зрозумілість системи, але може бути складною у застосуванні для простих проєктів. Чиста архітектура, нарешті, є концепцією проектування

програмного забезпечення, що покладається на розділення логіки додатку на чітко визначені компоненти. Вона дозволяє зберегти гнучкість та легкість змін у системі, але може бути складною у розробці та налаштуванні.

Вибір архітектурного підходу для інформаційної системи має вирішальне значення і повинен базуватися на потребах та вимогах конкретного проекту, а також на доступних ресурсах та досвіді розробників. Тільки правильний вибір може забезпечити ефективність розробки та стабільність інформаційної системи.

2.2 Аналіз вимог та визначення ключових характеристик програмної системи

Оцінка потреб і вимог до інформаційної системи є вирішальним етапом у процесі її створення, оскільки дозволяє визначити основні функції та характеристики, необхідні для досягнення бізнес-цілей та врахування потреб користувачів. Давайте розглянемо основні функціональні вимоги та ключові аспекти для інформаційної системи, яка забезпечує процессінг повідомлень:

1. Користувач конфігурує систему
   a) Заповнює дані для підключення до шини даних
   b) Надає датасет для тренування классифікатору
   c) Описує потрібний тип повідомлень та сценарій реагування на нього
2. Користувач активує систему з однієї точки запуску

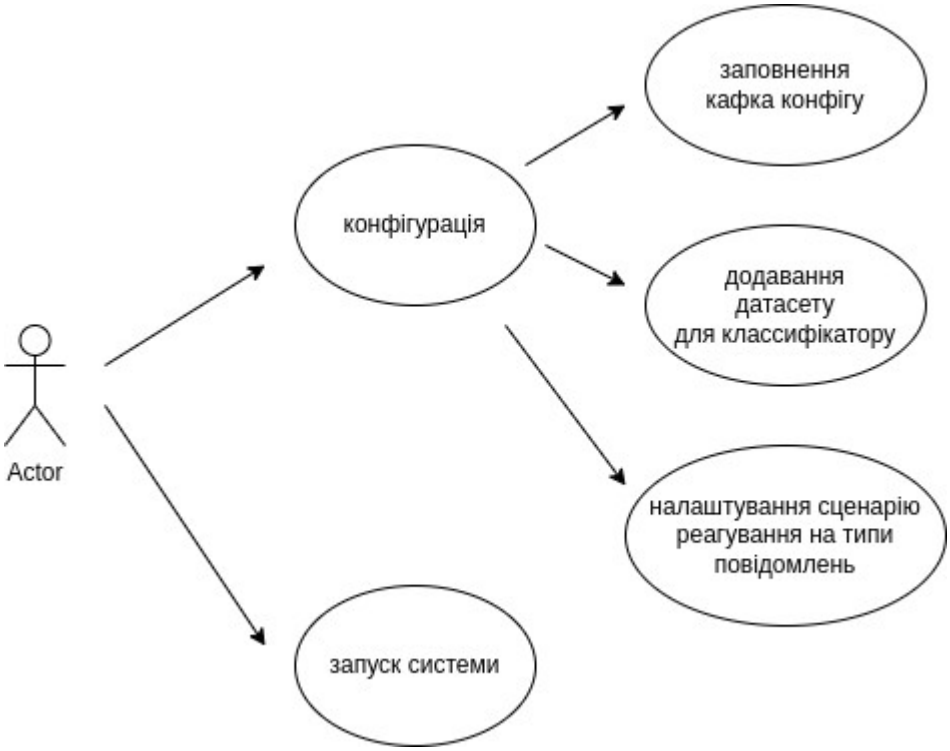

Рисунок 2 – The Use-Case View діаграма

Деякі аспекти важливі не лише за їх функціональністю, але й за їх якістю та здатністю працювати. Ці аспекти, відомі як нефункціональні вимоги, визначають стандарти та умови, які система повинна відповідати:

- швидкодія у реальному часі: система повинна забезпечувати швидке завантаження та обробку даних;
- масштабованість: важливо, щоб система була здатна розширюватися, щоб забезпечувати ефективну роботу при зростанні обсягу даних
- надійність: система повинна мати високу надійність, забезпечуючи стабільну працездатність та захист від відмов, забезпечуючи майже постійну доступність з урахуванням можливостей розбудови на 99,9% часу.
- легкість конфігурування: конфігурація має бути прописана в одному місці та мати інтуітивно зрозуміле призначення
- модульність: архітектура системи повинна бути гнучкою та модульною, щоб забезпечити зручність у розширенні та зміні функціоналу системи у майбутньому, не порушуючи стабільність та ефективність роботи.

- підтримка та оновлення: підтримка та оновлення системи повинні бути легкими та забезпечувати можливість швидкої виправлення помилок та впровадження поліпшень відповідно до змін у потребах користувачів та ринкових вимог.
- документація та навчання: документація та навчання є важливою складовою інформаційної системи. Вона повинна включати зрозумілу та повну документацію для користувачів та розробників, що сприяє розумінню принципів роботи системи та навчанню її використанню. Крім того, важливо передбачити підтримку та навчальні матеріали для співробітників компаній, які будуть користуватися системою.

Враховуючи вищезазначені вимоги, інформаційна система буде готовою до викликів сучасного бізнес-середовища. Забезпечуючи ефективність, стабільність та безпеку, вона стане надійним інструментом для користувачів. Важливо забезпечувати постійний моніторинг та оцінку роботи системи, щоб оперативно виявляти та вирішувати потенційні проблеми. Також необхідно постійно вдосконалювати функціональність системи, відповідаючи на зміни у потребах користувачів та вимогах ринку.

2.3 Огляд технологій, методів та інструментів розробки

Найвагоміші чинники при виборі технологій включають наступне: вимоги проєкту, тенденції розвитку технологій та наявність готових до використання рішень.

Таким чином було обрано наступний стек: Python як основну мову та технології Apache Kafka, Docker, MakeFile

Такий набір є типовим для ds та ml розробки та гарно себе показав у багатьох схожих проектах.

Python - це потужна мова програмування, яка відома своєю простотою, гнучкістю та розширюваністю. Використання Python у сферах машинного

навчання (ML) та обробки великих обсягів даних (Big Data) може мати декілька важливих переваг:

1. Простота використання: Python має простий синтаксис, що дозволяє швидко та легко розробляти програми. Це особливо важливо для розвитку ML-моделей та аналізу великих наборів даних, де швидкість розробки може бути критичною.
2. Велика екосистема: Python має велику кількість бібліотек та фреймворків, призначених для ML та Big Data, таких як TensorFlow, PyTorch, scikit-learn, Pandas, NumPy тощо. Це робить Python популярним вибором серед розробників, оскільки вони можуть легко використовувати готові рішення та інструменти для своїх проектів.
3. Висока продуктивність: Python забезпечує високу продуктивність завдяки своїм оптимізованим бібліотекам та інструментам, що дозволяють ефективно обробляти великі обсяги даних та навчати ML-моделі.
4. Легка інтеграція: Python легко інтегрується з іншими мовами програмування та технологіями, що дозволяє використовувати його у складних екосистемах та проєктах.
5. Активна спільнота: Python має велику та активну спільноту розробників, яка постійно працює над розвитком мови та її екосистеми. Це забезпечує підтримку, навчальні ресурси та швидке вирішення проблем через обмін знаннями та досвідом.

Git є одним з найпопулярніших та найбільш надійних інструментів для керування версіями коду в проектах розробки програмного забезпечення. Використання Git при розробці інформаційної системи має ряд переваг. Зокрема, ефективне керування версіями дозволяє відслідковувати зміни у коді та перевіряти історію змін. Можливість створення гілок для окремих завдань полегшує роботу над різними функціями чи виправленнями помилок без впливу на основний код. Використання хеш-функції SHA-1 для ідентифікації змін забезпечує надійність та цілісність даних. Крім того, розробники можуть використовувати відмітки (теги) для позначення важливих версій, що спрощує контроль версій та розгортання. Отже, враховуючи ці переваги, Git демонструє свою ефективність у розробці інформаційних систем.

2.4 Розробка стратегії реалізації програмної системи

Для втілення розробки системи такого рівня необхідно визначити основні кроки, які необхідно пройти:

1. Аналіз вимог: Ретельне вивчення потреб користувачів і визначення цілей проєкту для формулювання основних вимог.

2. Проектування архітектури: Розроблення загального плану дій для проекту, включаючи вибір технологій та платформи, створення інфраструктури та структури додатку.

3. Розробка: Втілення функціональності проєкту шляхом написання програмного коду та розробки компонентів.

4. Тестування: Проведення різних тестів для перевірки функціональності та надійності системи.

5. Розгортання: Підготовка системи до впровадження та його реалізація.

6. Експлуатація та підтримка: Після впровадження системи надання постійної підтримки та вдосконалення функціоналу.

7. Оцінка проекту: Аналіз результатів проєкту для визначення його успішності та можливостей подальшого розвитку.

Висновки розділу 2

У другому розділі розглянуті та проаналізовані переваги та недоліки різних архітектурних підходів, що застосовуються для створення програмних систем. Були сформульовані функціональні та нефункціональні вимоги до системи, визначені основні користувачі та описані сценарії їх використання системи. На основі аналізу вимог було визначено необхідні технології та інструменти для реалізації інформаційної системи, а також надані рекомендації

стосовно подальших кроків у процесі розробки, які включають планування, проектування, програмування, тестування та впровадження системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ОБРОБКИ ПОВІДОМЛЕНЬ

У сучасному світі неможливо представити real-time систему без шини даних. Обробка правильних івентів з цієї шини даних є ключем до побудови надійної асинхроної архітектури. Така архітектура де-факто стає стандартом для проектів подібної спеціфики та задля її имлпементації проблема критерія прийняття рішення про обробку івенту набуває критичної важливості

У даному розділі розглянутий підхід до побудови системи розпізнавання івентів для процессингу. Наведені технічні рішення до побудови та обгрунтування їх доцільності.

3.1 Проєктування архітектури

Проєктування архітектури є одним з найважливіших кроків у створенні програмного забезпечення, оскільки воно визначає структуру системи, полегшує розробку та підтримку, а також забезпечує масштабованість. Правильна архітектура допомагає ефективно використовувати ресурси, оптимізувати роботу системи та уникнути проблем з кодом. Важливість проєктування архітектури полягає в тому, що воно дозволяє створювати системи, які легко адаптуються до змін та вимог ринку, мають високий рівень надійності, пропускної здатності та доступності. Завдяки грамотно спроектованій архітектурі, код стає більш зрозумілим, модульним та легко тестованим, що сприяє ефективній роботі розробників.

Проєктування архітектури системи розпізнавання триггерних івентів потребує врахування певних концептів необхідних для подібних систем. В цьому розділі будуть розглянуті ці концепти та їх взаємодія.

3.1.1 Структура проекту

Так як мовою програмування є python, то структура проекту буде зроблена з умовою найкращих практик проектування проектів на цій мові.

Тобто одна точка входу в систему, розгортання інфрастурктури в докері, розбиття функціональності по пакетам-файлам-классам; зони відповідальності проекту:

- requirements.txt – залежності проекту
- README.md – документація
- Makefile – аліаси для docker-compose
- main.py – точка входу
- enums.py – енами
- docker-compose.yml – інфраструктура
- config.py – конфігурація в коді
- classification.py – классифікатор
- kafka_messaging – модуль шини даних(так як кафка де факто є лідером шин даних в мл, то імплементація залежить саме від неї)
- data – папка з метаданими

Виходячи з того, що проект не є великим, частина логіки знаходиться у корні проекту. Сам проект розгортається у венві.

3.1.2 Модульність та розділення відповідальності

Гарним підходом у полібних системах є модульність. В проекті реалізовано розбиття коду за функціональним навантаженням та забезпеченно пряму передачу залежностей згори донизу. Для цього активно використовуються інтерфейси, та паттерни сінглтон та фабричний метод. Такий підіхід дозволяє швидко та безболісно міняти імплементації, подмінювати їх для тестування чи впроваджувати стратегії за якими буде обрано той чи інший об’єкт в якості імплементації інтерфейсу.

В коді також імплементовано розбиття за архітектурними шарами, наразі їх 4, але у випадку розширення системи їх може стати більше. Наявні архітектури шари реалізують:

1. Шар месседжингу: Цей шар забезпечує отримання даних консьюмером з шиною даних. Його зона відповідальності це дто та його єдина задача це передача даних на наступний шар.

2. Шар бізнес логіки: бізнес логіка імплементує обробку даних отриманих від шару месседжингу та запускає дії в залежності від висновків від шару мл.
3. Шар ml: ключова функціональність системи, місце де імплементовано критерій прийняття рішень системи. Цей шар має лише одну задачу: визначати тип повідомлень, які йому передають. Тренування моделі прийняття рішень цього шару відбувається при запуску системи.
4. Шар конфігурації: конфігурація описана кодом, тож вона описується окремим шаром. З розширенням системи, шар конфігурації можно буде легко доповнити отриманням доступів з енвайроменту чи хмарних сховищ даних.

Таким чином фактично бізнес логіка, классифікатор, конфігурація та метадані є частиною однією компомненти, яка імплементує процесор обробки івентів.

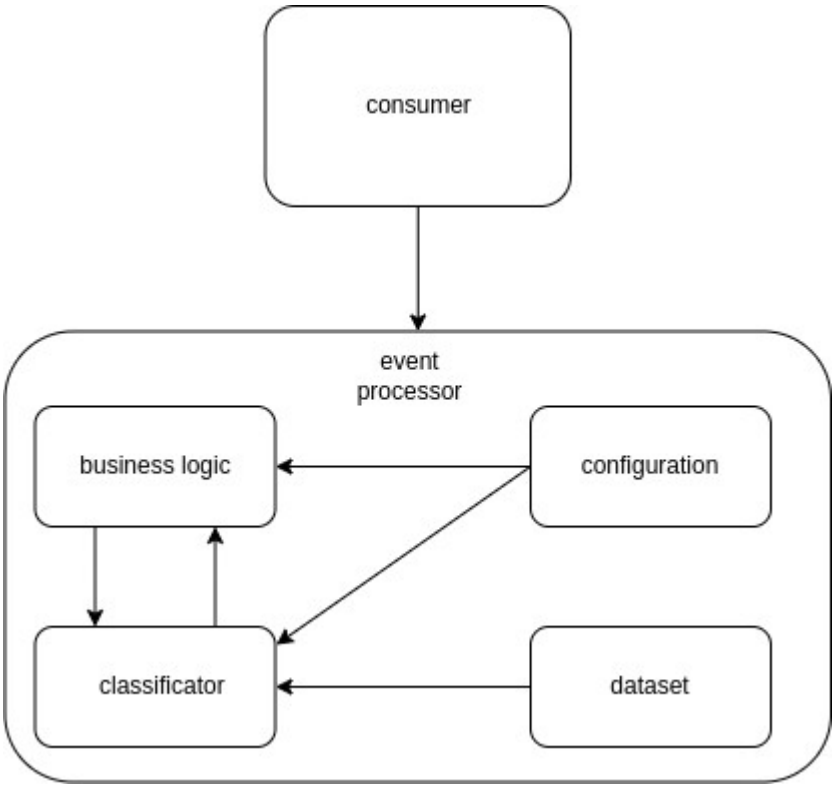

Рисунок 3 – Схема архітектури ПЗ

3.2 Розробка модуля месседжингу

3.2.1 Аналіз вимог до модуля месседжингу

Єдиною зоною відповідальності модуля месседжингу є отримання повідомлень. Але задля тестування було би добре також реалізувати логіку відправки повідомлень в шину даних. Тобто модуль повинен складатися з продьюсера та консьюмера.

3.2.2 Проектування модуля месседжингу

Згідно вимог було обрано розробити модуль месседжингу з двох файлів, один з классом консьюмера та інший з классом продьюсера. Конфігурації для обох сутностей знаходяться в конфіг файлі, там же і задається консьюмер група.

3.2.3 Реалізація модуля месседжингу

Для роботи з кафкою було обрано бібліотеку kafka-confluent. Для консьюмера процесу консьюму організовано в вигляді безкінечного циклу, який постійно опитує шину даних на предмет нових повідомлень. На кожне повідомлення виконується коллбек, який передається конструктором в консьюмер. Такий підхід дозволяє передавати залежності згори донизу та робить консьюмера незалежним від логіки, яку потрібно виконати для кожного месседжу.

3.3 Розробка шара бізнес логіки

Шар з бізнес логікою є найлегшим та представляє собою оркестрацію дій інших модулів. Даний шар складається лише з одного файлу main.py та відповідає за порядок процесу обробки даних. В ньому же знаходиться точка запуску системи.

3.3.1 Функціональність шара бізнес логіки

Функціональність модуля це є алгоритмом процесінгу даних. Алгоритм можно представити у вигляді наступних кроків:

- зчитування конфігурації
- створення коллбеку для процесінгу повідомлень
- створення классифікатору
- створення консьюмера
- запуск консьюмера

3.3.2 Реалізація шара бізнес логіки

Реалізація повністю відповідає крокам з опису функціональності шару

3.4 Розробка шару ml

Саму в цьому шарі знахожиться core функціональність. Незважаючи на простоту функціональності, саме цей шар є найбільш складним.

3.4.1 Функціональність шара ml

Функціональність цього шару це классифікація повідомлень. Для классифікації повідомлень потрібно натренувати модель, яка зможе виконувати цю задачу. Тренування моделі доречно проводити під час запуску системи.

3.4.2 Реалізація шару ml

Для классифікації потрібна конфігурація та датасет. Ці дані передаються в классифікатор напряму з файлу конфігурації. Так як існує доволі багато алгоритмів для классифікації, код який користується классифікатором не повинен щось знати про особливості імплементації. Саме для цього було розроблено інтерфейс классифікатору, який реалізується різними алгоритмами классифікації. Вибір потрібного алгоритму здійснюється через конфігурацію, де

можна вказати потрібний алгоритм за допомогою певного єнаму. Наразі в системі представлені наступні алгоритми:

- K_NEAREST_NEIGHBOURS
- LOGISTIC_REGRESSION
- DECISION_TREE
- LINEAR_DISCRIMINANT
- GAUSSIAN_NAIVE_BAYES
- SUPPORT_VECTOR_MACHINE

Для отримання потрібного об’єкту в залежності від конфігурації реалізовано паттерн фабричний метод. Кожний класс імплементація алгоритму реалізує паттерн одинак.

3.5 Розгортання

Система розгортається в віртуальному середовищі venv за допомогою наступних кроків:

- завантаження залежностей
- налаштування конфігурації(за потреби активація інфраструктури за допомогою cmake білд системи)
- запуск main скрипту

3.6 Опис програмного продукту

Основні особливості та функціонал продукту:

- обробка кафка месседжей за сконфігурованим сценарієм
- підтримка 6 алгоритмів классифікації повідомлень
- легка та ізольована архітектура, розроблена для розширення системи

Також потрібно врахувати недоліки для об'єктивності, які можуть виникнути в ході подальшої розробки та використання системи:

- потреба в конфігурації через енв
- проблема вибору алгоритму

Загалом, система відповідає вимогам та повертає корректні дані під час тестування.

Як результат роботи, можна побачити скріншоти з виконання системи, розміщені у додатку.

Висновки розділу 3

У цьому розділі було описано реалізацію системи для процессінгу повідомлень за сконфігурованим сценарієм. Наведено сильні та слабкі сторони системи. Описана архітектура та основні архітектурні рішення, розбиття на шари, а також специцікація для кожного шару. Наведено структуру імплементації кожного шару.

ВИСНОВКИ

У даному дослідженні було вирішено проблему процесінгу повідмолень з неясним крітерієм прийняття рішень. Для досягнення цієї мети, ми сформулювали наступні дослідницькі завдання: огляд і аналіз сучасних методів процессінгу подібних повідмолень, аналіз цих методів, ознайомлення з існуючими інструментами та дослідження структури програмної системи.

Ми провели дослідження, використовуючи різноманітні джерела інформації, такі як наукові ресурси, конференції, а також програмні коди. Цей підхід дозволив нам отримати актуальну інформацію для докладного аналізу та досягнення поставлених цілей.

В результаті наших досліджень було розроблено програмну систему для процесінгу повідомлень з нечітким критерієм прийняття рішень.

Розроблена програмна система допомагає обробити потрібні повідомленння та легко конфігурується для підтримки зазначених сценаріїв обробки повідомлень.

З урахуванням отриманих результатів, можна зазначити, що наше дослідження відіграє важливу роль у розвитку асинхроних архітектур заточених на обробку подій. Впровадження розробленої системи дозволить ефективно обробити потрібні повідомлення та виконати сценарій заданий для конкретного

повідомлення. Кодова база системи має усе необхідне як для подальшої розробки так і для підтримки системи. Дослідження несе користь як з наукової, так і з практичної сторони та має потенціал для подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chen, M., Mao, S., & Liu, Y. Big Data: A Survey. *Mobile Networks and Applications*, 2014, vol. 19, no. 2, pp. 171–209.
2. Dean, J., & Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 2008, vol. 51, no. 1, pp. 107-113.
3. LeCun, Y., Bengio, Y., & Hinton, G. Deep Learning. *Nature*, 2015, vol. 521, no. 7553, pp. 436-444.
4. Chen, Q., Zeng, D., & Zhao, H. Big Data Deep Learning: Challenges and Perspectives. *IEEE Access*, 2014, vol. 2, pp. 514-525.
5. Hadoop.apache.org. Apache Hadoop. 2024. URL: https://hadoop.apache.org/.
6. Zaharia, M., et al. Apache Spark: A Unified Engine for Big Data Processing. *Communications of the ACM*, 2016, vol. 59, no. 11, pp. 56-65.
7. O'Reilly, C. What is Big Data? *O'Reilly Media*, 2015. URL: https://www.oreilly.com/library/view/what-is-big/9781449379310/.
8. Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016.
9. Hastie, T., Tibshirani, R., & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed., Springer, 2009.
10. Abadi, M., et al. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX Association, 2016.
11. DataBricks. Apache Spark. 2024. URL: https://databricks.com/apache-spark.
12. Gartner. Gartner Glossary. 2022. URL: https://www.gartner.com/en/information-technology/glossary.
13. Microsoft. Azure Machine Learning. 2024. URL: https://azure.microsoft.com/en-us/services/machine-learning/.
14. Amazon Web Services. Amazon SageMaker. 2024. URL: https://aws.amazon.com/sagemaker/.

15. IBM. IBM Watson Machine Learning. 2024. URL: https://www.ibm.com/cloud/watson-studio.

16. Google Cloud. Google Cloud Machine Learning. 2024. URL: https://cloud.google.com/products/ai.

17. Apache Software Foundation. Apache Flink. 2024. URL: https://flink.apache.org/.

18. McKinsey & Company. Big Data: The Next Frontier for Innovation, Competition, and Productivity. 2011. URL: https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/big-data-the-next-frontier-for-innovation.

19. Beyer, M., et al. The Importance of Big Data: A Definition. *Gartner*, 2011. URL: https://www.gartner.com/en/newsroom/press-releases/2011-10-12-gartner-highlights-key-predictions-for-it-organizations-and-users-in-2012-and-beyond.

20. Halevy, A., Norvig, P., & Pereira, F. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 2009, vol. 24, no. 2, pp. 8-12. https://doi.org/10.1109/MIS.2009.36.

ДОДАТКИ

Додаток А:

Репозиторій з кодом: https://github.com/laarchenko/consumer-resolver

Код системи:

main.py

```
import json

import config
from classification import get_classifier
from kafka_messaging.consumer import DataConsumer

def process_data(data_json):
    data_dict = json.loads(data_json)

    data_list = [data_dict[i] for i in config.CLASSIFICATION_PROPERTY_NAMES]

    data_type = get_classifier().serve(data_list)

    action = config.TYPE_ACTION_DICT.get(data_type)
    if action is not None:
        action(data_dict)

if __name__ == '__main__':
```

```
data_consumer = DataConsumer(config.KAFKA_CONFIG, config.KAFKA_TOPIC_NAME, process_data)
    data_consumer.consume()
```

requirements.txt

```
confluent_kafka
pandas
scikit-learn
```

enums.py

```
from enum import Enum, auto

class DATA_CLASSIFIER_METHOD_TYPE(Enum):
    K_NEAREST_NEIGHBOURS = auto(),
    LOGISTIC_REGRESSION = auto(),
    DECISION_TREE = auto(),
    LINEAR_DISCRIMINANT = auto(),
    GAUSSIAN_NAIVE_BAYES = auto(),
    SUPPORT_VECTOR_MACHINE = auto()
```

docker-compose.yml

```
version: '3'

services:
  zookeeper:
    container_name: project_name-zookeeper
    image: confluentinc/cp-zookeeper:6.2.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - 22181:2181

  kafka:
    container_name: project_name-kafka
    image: confluentinc/cp-kafka:6.2.0
    ports:
      - "9092:9092"
    expose:
```

```
    - "29092"
  depends_on:
    - zookeeper
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
                                KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_INTERNAL:PLAINTEXT
                                                KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL://kafka:29092
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1


 init-kafka:
  container_name: project_name-init-kafka
  image: confluentinc/cp-kafka:6.2.0
  depends_on:
    - kafka
  entrypoint: [ '/bin/sh', '-c' ]
  command: |
```

"

blocks until kafka is reachable

echo -e 'Creating kafka topics'

kafka-topics --bootstrap-server kafka:29092 --create --if-not-exists --topic fruits --replication-factor 1 --partitions 1

echo -e 'Created kafka topics:'

kafka-topics --bootstrap-server kafka:29092 –list

"

config.py

```
import os
import random

from enums import DATA_CLASSIFIER_METHOD_TYPE

ROOT_DIR = os.path.dirname(os.path.abspath(__file__))
```

```
CLASSIFICATION_TRAINING_DATASET_PATH =
'/data/fruit_data_with_colors.txt'

TYPE_PROPERTY_NAME = 'fruit_name'
CLASSIFICATION_PROPERTY_NAMES = ['mass', 'width', 'height', 'color_score']

TYPE_ACTION_DICT = {
    'apple': lambda x: print('apple = ' + str(x)),
    'mandarin': lambda x: print('mandarin = ' + str(x))
}

KAFKA_CONFIG = {
        'bootstrap.servers': 'localhost:9092',
        'group.id': 'mygroup' + str(random.randrange(0, 10000)),
        'auto.offset.reset': 'earliest',
        'security.protocol': 'PLAINTEXT'
}

KAFKA_TOPIC_NAME = 'fruits'
```

DATA_CLASSIFIER_METHOD =
DATA_CLASSIFIER_METHOD_TYPE.K_NEAREST_NEIGHBOURS

classification.py

```
import abc

import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

import config
from enums import DATA_CLASSIFIER_METHOD_TYPE
```

```
def get_classifier():

                    if(config.DATA_CLASSIFIER_METHOD == DATA_CLASSIFIER_METHOD_TYPE.K_NEAREST_NEIGHBOURS):
        return DataClassifierKnn.get_instance()
                    if(config.DATA_CLASSIFIER_METHOD == DATA_CLASSIFIER_METHOD_TYPE.LOGISTIC_REGRESSION):
        return DataClassifierLogisticRegression.get_instance()
                    if (config.DATA_CLASSIFIER_METHOD == DATA_CLASSIFIER_METHOD_TYPE.DECISION_TREE):
        return DataClassifierDecisionTree.get_instance()
                    if (config.DATA_CLASSIFIER_METHOD == DATA_CLASSIFIER_METHOD_TYPE.LINEAR_DISCRIMINANT):
        return DataClassifierLinearDiscriminant.get_instance()
                    if (config.DATA_CLASSIFIER_METHOD == DATA_CLASSIFIER_METHOD_TYPE.GAUSSIAN_NAIVE_BAYES):
        return DataClassifierGaussianNaiveBayes.get_instance()
                    if (config.DATA_CLASSIFIER_METHOD == DATA_CLASSIFIER_METHOD_TYPE.SUPPORT_VECTOR_MACHINE):
        return DataClassifierSupportVectorMachine.get_instance()
```

```
    raise Exception(f"Data classifier {config.DATA_CLASSIFIER_METHOD} is not supported")

class DataClassifierAbc(abc.ABC):

    def serve(self, data_dict):
        pass

    @staticmethod
    def _generate_instance():
        pass

    @classmethod
    def get_instance(cls):
        if not hasattr(cls, "_instance"):
            cls._instance = cls._generate_instance()
        return cls._instance


class DataClassifierKnn(DataClassifierAbc):
```

```
def __init__(self, properties):
    self.model = None
    self.properties = properties

def serve(self, data_dict):
    scaled_properties = self.scaler.transform([data_dict])
    return self.model.predict(scaled_properties)[0]

def _train(self, data_path, type_property_name):

    data = pd.read_table(data_path)

    X = data[self.properties]
    y = data[type_property_name]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    self.scaler = MinMaxScaler()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)
```

```
        self.model = KNeighborsClassifier()
        self.model.fit(X_train, y_train)

                        print('Accuracy of K-NN classifier on training set: {:.2f}'.format(self.model.score(X_train, y_train)))
                            print('Accuracy of K-NN classifier on test set: {:.2f}'.format(self.model.score(X_test, y_test)))

    @staticmethod
    def _generate_instance():
                                                        classifier = DataClassifierKnn(config.CLASSIFICATION_PROPERTY_NAMES)
                                                    classifier._train(config.ROOT_DIR + config.CLASSIFICATION_TRAINING_DATASET_PATH,
config.TYPE_PROPERTY_NAME)
        return classifier

class DataClassifierLogisticRegression(DataClassifierAbc):

    def __init__(self, properties):
        self.model = None
        self.properties = properties
```

```
def serve(self, data_dict):
    scaled_properties = self.scaler.transform([data_dict])
    return self.model.predict(scaled_properties)[0]

def _train(self, data_path, type_property_name):

    data = pd.read_table(data_path)

    X = data[self.properties]
    y = data[type_property_name]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    self.scaler = MinMaxScaler()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)

    self.model = LogisticRegression()
    self.model.fit(X_train, y_train)
    print('Accuracy of Logistic regression classifier on training set: {:.2f}'
```

```
            .format(self.model.score(X_train, y_train)))
        print('Accuracy of Logistic regression classifier on test set: {:.2f}'
            .format(self.model.score(X_test, y_test)))

    @staticmethod
    def _generate_instance():
                                                        classifier =
DataClassifierLogisticRegression(config.CLASSIFICATION_PROPERTY_NAMES)
                                                    classifier._train(config.ROOT_DIR +
config.CLASSIFICATION_TRAINING_DATASET_PATH,
config.TYPE_PROPERTY_NAME)
        return classifier

class DataClassifierDecisionTree(DataClassifierAbc):

    def __init__(self, properties):
        self.model = None
        self.properties = properties

    def serve(self, data_dict):
        scaled_properties = self.scaler.transform([data_dict])
```

```
    return self.model.predict(scaled_properties)[0]

def _train(self, data_path, type_property_name):

    data = pd.read_table(data_path)

    X = data[self.properties]
    y = data[type_property_name]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    self.scaler = MinMaxScaler()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)

    self.model = DecisionTreeClassifier().fit(X_train, y_train)
    print('Accuracy of Decision Tree classifier on training set: {:.2f}'
         .format(self.model.score(X_train, y_train)))
    print('Accuracy of Decision Tree classifier on test set: {:.2f}'
         .format(self.model.score(X_test, y_test)))
```

```
    @staticmethod
    def _generate_instance():
                                                        classifier          =
DataClassifierDecisionTree(config.CLASSIFICATION_PROPERTY_NAMES)
                                                    classifier._train(config.ROOT_DIR       +
config.CLASSIFICATION_TRAINING_DATASET_PATH,
config.TYPE_PROPERTY_NAME)
        return classifier

class DataClassifierLinearDiscriminant(DataClassifierAbc):

    def __init__(self, properties):
        self.model = None
        self.properties = properties

    def serve(self, data_dict):
        scaled_properties = self.scaler.transform([data_dict])
        return self.model.predict(scaled_properties)[0]

    def _train(self, data_path, type_property_name):
```

```
    data = pd.read_table(data_path)

    X = data[self.properties]
    y = data[type_property_name]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    self.scaler = MinMaxScaler()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)

    self.model = LinearDiscriminantAnalysis()
    self.model.fit(X_train, y_train)
    print('Accuracy of LDA classifier on training set: {:.2f}'
         .format(self.model.score(X_train, y_train)))
    print('Accuracy of LDA classifier on test set: {:.2f}'
         .format(self.model.score(X_test, y_test)))

@staticmethod
def _generate_instance():
```

```
                                                    classifier =
DataClassifierLinearDiscriminant(config.CLASSIFICATION_PROPERTY_NAMES
)
                                                    classifier._train(config.ROOT_DIR +
config.CLASSIFICATION_TRAINING_DATASET_PATH,
config.TYPE_PROPERTY_NAME)
        return classifier

class DataClassifierGaussianNaiveBayes(DataClassifierAbc):

    def __init__(self, properties):
        self.model = None
        self.properties = properties

    def serve(self, data_dict):
        scaled_properties = self.scaler.transform([data_dict])
        return self.model.predict(scaled_properties)[0]

    def _train(self, data_path, type_property_name):

        data = pd.read_table(data_path)
```

```
    X = data[self.properties]
    y = data[type_property_name]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    self.scaler = MinMaxScaler()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)

    self.model = GaussianNB()
    self.model .fit(X_train, y_train)
    print('Accuracy of GNB classifier on training set: {:.2f}'
          .format(self.model .score(X_train, y_train)))
    print('Accuracy of GNB classifier on test set: {:.2f}'
          .format(self.model .score(X_test, y_test)))

@staticmethod
def _generate_instance():
```

```
                                                    classifier =
DataClassifierGaussianNaiveBayes(config.CLASSIFICATION_PROPERTY_NAME
S)
                                                    classifier._train(config.ROOT_DIR +
config.CLASSIFICATION_TRAINING_DATASET_PATH,
config.TYPE_PROPERTY_NAME)
        return classifier

class DataClassifierSupportVectorMachine(DataClassifierAbc):

    def __init__(self, properties):
        self.model = None
        self.properties = properties

    def serve(self, data_dict):
        scaled_properties = self.scaler.transform([data_dict])
        return self.model.predict(scaled_properties)[0]

    def _train(self, data_path, type_property_name):

        data = pd.read_table(data_path)
```

```
    X = data[self.properties]
    y = data[type_property_name]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    self.scaler = MinMaxScaler()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.transform(X_test)

    self.model = SVC()
    self.model.fit(X_train, y_train)
    print('Accuracy of SVM classifier on training set: {:.2f}'
          .format(self.model.score(X_train, y_train)))
    print('Accuracy of SVM classifier on test set: {:.2f}'
          .format(self.model.score(X_test, y_test)))

@staticmethod
def _generate_instance():
```

```
                                                    classifier =
DataClassifierSupportVectorMachine(config.CLASSIFICATION_PROPERTY_NA
MES)
                                                    classifier._train(config.ROOT_DIR +
config.CLASSIFICATION_TRAINING_DATASET_PATH,
config.TYPE_PROPERTY_NAME)
    return classifier
```

Makefile

```
infra-up:
    docker-compose up
infra-down:
    docker-compose down
infra-rm:
    docker-compose rm
```

kafka/consumer.py

```
from confluent_kafka import Consumer

class DataConsumer:

    def __init__(self, kafka_config, topic_name, process_function):
        self.process_function = process_function
        self.consumer = Consumer(kafka_config)
        self.consumer.subscribe([topic_name])

    def consume(self):
        while True:
            msg = self.consumer.poll(1.0)

            if msg is None:
                continue
            if msg.error():
                print("Consumer error: {}".format(msg.error()))
                continue
            decoded_msg = msg.value().decode('utf-8')
```

```
        print('Received message: {}'.format(decoded_msg))
        self.process_function(decoded_msg)

def close(self):
    self.consumer.close()
```