КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: «***Автоматизація тестування програмного забезпечення на основі машинного навчання***»

Мацієвський Олексій Олегович

Київ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

інформаційних технологій проектування та прикладної математики
(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО
„22” Червня 2023 року

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: " Автоматизація тестування програмного забезпечення на основі машинного навчання "

Виконав: студент 2 – го курсу, групи КНм – 22

Спеціальності: 122 «Комп’ютерні науки .
(шифр і назва напряму підготовки, спеціальності)

Магістрант Мацієвський Олексій Олегович
(прізвище та ініціали)

Керівник Горда Олена Володимирівна
(прізвище та ініціали)

Рецензент Доля Олена Вікторівна
(прізвище та ініціали)

Київ, 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій .
Кафедра: інформаційних технологій .
Освітній рівень: «магістр за ОПП» .
Спеціальність: 122 «Комп'ютерні науки» .

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
Тетяна ГОНЧАРЕНКО
„22" Червня 2023 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

1. Тема роботи: Автоматизація тестування програмного забезпечення на основі машинного навчання
   затверджена наказом ректора КНУБА № 1280/2 від «26» Червня 2023р.
2. Керівник роботи: Горда Олена Володимирівна.
3. Строк подання студентом роботи до захисту: 9 Грудня 2023р.
4. Зміст пояснювальної записки за розділами:

Р.1. Аналіз сучасних тенденцій та методів автоматизації тестування у сфері програмного забезпечення з використанням машинного навчання.

Р.2. Інструменти та технології для автоматизації тестування на основі машинного навчання.

Р.3. Опис програмної реалізації методу.

Р.4. Реалізація програмного забезпечення для автоматизації тестування на основі машинного навчання.

5. Інформаційні слайди:

С.1. Актуальність дослідження

С.2. Порівняння видів тестування

С.3. Методи тестування

С.4. Архітектура програмного забезпечення

6. Календарний план виконання атестаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Р.1. Аналіз сучасних тенденцій та методів автоматизації тестування у сфері програмного забезпечення з використанням машинного навчання.	01.09.23
Р.2. Інструменти та технології для автоматизації тестування на основі машинного навчання.	24.09.23
Р.3. Опис програмної реалізації методу.	02.10.23
Р.4. Реалізація програмного забезпечення для автоматизації тестування на основі машинного навчання.	14.11.23
Остаточне оформлення роботи	27.11.23
Направлення роботи на рецензування, перевірку на плагіат	04.12.23
Попередній захист роботи на кафедрі	07.12.23

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1.	Рябчун Ю.В.	05.09.2023	
Розділ 2.	Гончаренко Т.А.	01.10.2023	
Розділ 3.	Голенков В.Г.	06.10.2023	
Розділ 4.	Горда О.В.	20.11.2023	

8. Дата видачі завдання: «22» Червня 2023р.

Керівник _____ Горда О.В.
(підпис) (прізвище та ініціали)

Магістрант _____ Мацієвський О.О.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ

Київський національний університет будівництва і архітектури

Мацієвський Олексій Олегович

факультет автоматизації і інформаційних технологій, група КНм-22

Тема атестаційної випускної роботи: «Автоматизація тестування програмного забезпечення на основі машинного навчання»

освітній рівень: магістр,

спеціальність: 122 «Комп'ютерні технології»,

Науковий керівник: Горда Олена Володимирівна

Обсяг роботи. Атестаційна випускова робота магістра складається: з 4 розділів, 86 стор., 3 таблиць, 45 рисунків, завдання, анотації, вступу, висновків, списку використаних джерел та додатків.

Актуальність теми. Впровадження машинного навчання в автоматизацію тестування призводить до покращення якості програмного забезпечення та сприяє більш швидкому впровадженню нових функцій і виправленню помилок в тестуванні програмного забезпечення.

У вступі визначені основні напрямки дослідження, обґрунтовано актуальність теми, сформульовано мету та основні завдання системи.

У першому розділі розглянуто базову теорію про штучний інтелект, машинне навчання, сучасні тенденції та рівень розвитку машинного навчання в штучному інтелекті.

У другому розділі розглянуто сучасні інструменти для автоматизації тестування програмного забезпечення, їх основні функції, дослідження можливості машинного навчання, алгоритми машинного навчання.

У третьому розділі представлено практичне забезпечення засобів для роботи з інформацією, архітектура програмного забезпечення, аналіз бази даних, та вибір мови програмування.

У четвертому розділі представлена реалізація програмного продукту, моніторинг тестів, прогнозування тестів, автоматичні тестові набори, та метод створення тестових сценаріїв.

Ключові слова автоматизація тестування, машинне навчання, тестові сценарії, ефективність тестування, технічна інтеграція, генерація тестових даних, аномалії в програмному забезпеченні, методологія тестування

Якість оформлення проекту. Атестаційна випускна робота магістра оформлена у відповідності до діючих нормативних документів та методичних вказівок до виконання дипломних робіт для студентів спеціальності 126 «Інформаційні системи і технології». Порушень та зауважень під час розробки та перевірки дипломної роботи не виявлено.

Загальний висновок стосовно роботи та присвоєння авторові освітньо-кваліфікаційного рівня «магістр». Робота виконана якісно та на високому рівні, студент продемонстрував достатній рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій. Заслуговує оцінки «Відмінно».

Науковий керівник _____/ Горда Олена Володимирівна./

(підпис)

Посада, місце роботи: к.т.н., доц. кафедри інформаційних технологій

«23» Листопада 2022р.

АНОТАЦІЯ

Дипломна робота присвячена дослідженню та розробці методів автоматизації тестування програмного забезпечення з використанням технологій машинного навчання.

Аналізуючи сучасні методи автоматизованого тестування та існуючі підходи до машинного навчання, дослідження розглядає можливості їхнього взаємодії. Визначається методологія, що дозволяє ефективно впроваджувати машинне навчання у процес тестування, а також створювати моделі для генерації тестових сценаріїв та виявлення аномалій.

Робота включає практичну реалізацію розроблених моделей та їхнє використання в реальних проектах. Експериментальна частина оцінює ефективність та придатність використання машинного навчання у тестуванні, зокрема у порівнянні з традиційними методами.

На основі отриманих результатів формулюються висновки та рекомендації для інтеграції машинного навчання в практику автоматизованого тестування. Робота спрямована на внесення вагомого вкладу у розвиток та вдосконалення технік тестування програмного забезпечення, забезпечуючи високу якість та стабільність програмних продуктів.

Ключові слова: автоматизація тестування, машинне навчання, тестові сценарії, ефективність тестування, технічна інтеграція, генерація тестових даних, аномалії в програмному забезпеченні, методологія тестування

ANNOTATION

This thesis focuses on the exploration and development of methods for automating software testing using machine learning technologies.

By analyzing contemporary methods of automated testing and existing approaches to machine learning, the research investigates the possibilities of their synergy. A methodology is defined that effectively integrates machine learning into the testing process and creates models for generating test scenarios and identifying anomalies.

The work includes the practical implementation of the developed models and their application in real-world projects. The experimental section evaluates the effectiveness and suitability of using machine learning in testing, particularly in comparison with traditional methods.

Based on the obtained results, conclusions and recommendations are formulated for the integration of machine learning into automated testing practices. The thesis aims to make a substantial

contribution to the advancement and refinement of software testing techniques, ensuring high quality and stability in software products.

Key words: automated testing, machine learning, test scenarios, testing efficiency, technical integration, test data generation, anomalies in software, testing methodology.

РЕЦЕНЗІЯ

на атестаційну випускну роботу

Студента Мацієвського Олексія Олеговича

Факультет автоматизації і інформаційних технологій

Спеціальності 122 «Інформаційних технологій»

Тема роботи: Автоматизація тестування програмного забезпечення на основі машинного навчання.

Обсяг роботи: атестаційна випускова робота магістра складається: з 4 розділів, 82 стор., 2 таблиць, 46 рис., завдання, анотація, вступу, висновків, списку використаних джерел та додатків.

Висновок про відповідність завданню: робота виконана у повній відповідності до завдання і у встановлений термін .

Актуальність обраної теми: використання машинного навчання дозволяє оптимізувати тестування, забезпечуючи ефективний аналіз даних та автоматизацію виявлення помилок.

Використання у роботі сучасних досягнень науки і техніки: розробка проекту базується на використанні сучасних інформаційних комп'ютерних технологій

Використання у роботі комп'ютерних технологій: MH, Big Data, MCTS

Практичне значення роботи: впровадження сучасних інформаційних технологій має забезпечувати виконання ряду вимог, у тому числі наявність зручного і дружнього інтерфейсу, забезпечення безпеки за допомогою різних методів контролю та розмежування доступу до інформаційних ресурсів, підтримку розподіленої обробки інформації.

Якість оформлення роботи: випускна робота оформлена у відповідності до діючих нормативних документів та методичних вказівок для студентів спеціальності 122 «Комп'ютерні науки»

Зауваження та побажання: Зауважень не виявлено

Загальний висновок стосовно роботи та надання авторові освітнього ступеня "магістр": робота виконана на високому рівні, студент продемонстрував високий рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій. Заслуговує оцінки «відмінно».

Рецензент _____/ к.ф.-м.н., доцент Доля О.В.

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

Посада, місце роботи: доцент кафедри інформаційних технологій проєктування та прикладної математики КНУБА

«__» _____2023р.

ЗМІСТ

ВСТУП

Програмне забезпечення в сучасному світі стає все більш складним і обширним, вимагаючи від розробників та тестувальників ефективних інструментів для забезпечення його надійності та якості. Одним із ефективних підходів до автоматизації тестування програмного забезпечення є використання методів машинного навчання.

Машинне навчання (Machine Learning, ML) - це галузь штучного інтелекту, яка надає комп'ютерам здатність навчатися на основі даних і вдосконалювати свої навички без явного програмування. Використання машинного навчання в автоматизації тестування дозволяє створювати більш адаптивні та ефективні тести, а також автоматизувати процеси виявлення та виправлення помилок.

Однією з ключових переваг використання машинного навчання в автоматизації тестування є здатність моделей до аналізу великої кількості даних, ідентифікації патернів та прогнозування потенційних проблем. Моделі можуть виявляти аномалії, що допомагає покращити якість тестування та забезпечити більшу стабільність програмного продукту.

У цьому контексті важливо розглядати різноманітні аспекти використання машинного навчання в автоматизації тестування: від створення тестових сценаріїв на основі аналізу історії помилок до розробки моделей, які можуть передбачати критичні точки невірного функціонування програмного продукту.

Цей напрям автоматизації тестування є актуальним та перспективним, оскільки дозволяє зменшити затрати на тестування, підвищити ефективність та прискорити випуск програмних продуктів на ринок. У наступних розділах будуть розглянуті конкретні приклади використання машинного навчання в автоматизації тестування та їх вплив на розробку програмного забезпечення.

Мета кваліфікаційної роботи: детальне вивченні, аналіз та розробка практичних рішень для впровадження машинного навчання у процес автоматизованого тестування програмного забезпечення.

Об’єкт дослідження – автоматизована система тестування

Предмет дослідження – методи та моделі штучного інтелекту автоматизованого тестування програмного забезпечення.

Методи дослідження:

-Методи системного аналізу

- Методи об’єктно-орієнтованого проектування

- Методи побудови та навчання нейромереж

- Методи прийняття рішень

- Застосування алгоритму Байєса в аналізі даних

Ключовими аспектами мети є:

1) Дослідження сучасного стану автоматизованого тестування;
2) Вивчення основ машинного навчання;
3) Розробка методології використання машинного навчання в тестуванні;
4) Розробка та реалізація моделей машинного навчання для тестування;
5) Експериментальна перевірка ефективності;
6) Висновки та рекомендації;

РОЗДІЛ 1.
АНАЛІЗ СУЧАСНИХ ТЕНДЕНЦІЙ ТА МЕТОДІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ У СФЕРІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ

1.1. Поняття автоматизація тестування програмного забезпечення

Автоматизація тестування програмного забезпечення (АТ) – це процес використання спеціалізованих інструментів та програм для автоматичного виконання тестових сценаріїв та оцінки результатів. Основна мета автоматизації тестування полягає в покращенні ефективності тестового процесу, зменшенні часу виконання, підвищенні точності результатів та забезпеченні покриття коду тестами [1].

Основні поняття та складові автоматизації тестування включають [2]:

Тестові скрипти (Test Scripts). Це набір команд або інструкцій, написаних за допомогою автоматизованих інструментів, які виконують тести на програмному продукті.

***Інструменти автоматизації тестування (Test Automation Tools)*.** Програмні засоби, які дозволяють створювати, виконувати та аналізувати результати тестових скриптів автоматично. Приклади включають Selenium, JUnit, TestNG, Appium, і багато інших.

Тестові дані (Test Data). Інформація, яка використовується для виконання тестових сценаріїв. Це може включати в себе різні вхідні дані, конфігураційні файли, бази даних тощо.

Звітність (Reporting). Автоматизовані тести зазвичай створюють звіти про результати виконання, що дозволяє швидше виявляти помилки та відстежувати прогрес тестування.

Регресійне тестування (Regression Testing). Автоматизація часто використовується для виконання повторних тестів для підтвердження, що зміни

в програмному коді не призвели до появи нових помилок або не порушили існуючу функціональність.

Ключові аспекти автоматизації тестування програмного забезпечення включають:

Створення тестових сценаріїв. Автоматизовані тестові сценарії розробляються на основі вимог до програми. Це може включати тестування функціональності, навантажувальне тестування, тестування відмовостійкості тощо.

Вибір інструментів. Вибір правильних інструментів для автоматизації є критичним етапом. Інструменти повинні відповідати потребам проекту, забезпечувати підтримку різних типів тестів та бути інтегрованими з іншими інструментами у розробці.

Скриптинг та програмування. Створення автоматизованих тестів вимагає написання тестових скриптів або програмного коду. Ці скрипти визначають послідовність дій, які потрібно виконати для тестування конкретного функціоналу.

Виконання тестів. Інструменти автоматизації виконують створені тестові скрипти на програмному забезпеченні, ідентифікують помилки, записують результати тестів та генерують звіти.

Підтримка тестових наборів. Підтримка та розширення тестових скриптів у відповідь на зміни в програмному забезпеченні. Це важливо для забезпечення актуальності автоматизованих тестів у високодинамічних проектах.

Інтеграція з CI/CD. Автоматизація тестування пов'язана з процесом неперервної інтеграції та поставки (CI/CD), що дозволяє автоматизовано виконувати тести при кожній зміні коду та в межах автоматизованого процесу розгортання.

Переваги автоматизації тестування включають швидкість виконання, повторюваність тестових сценаріїв, раннє виявлення помилок, покращення покриття коду тестами та ефективне використання ресурсів тестування. Однак, автоматизація тестування також може стикатися з викликами, такими як висока

вартість впровадження та утримання, необхідність постійного оновлення тестових скриптів та обмеження в тестуванні графічного інтерфейсу.

1.2. Роль штучного інтелекту в тестуванні

Штучний інтелект (ШІ) грає значущу роль в тестуванні програмного забезпечення, вносячи інновації та покращення в цей процес. Важливі аспекти ролі ШІ в тестуванні включають автоматизацію тестів, підтримку у виявленні помилок, аналіз даних, створення реалістичних тестових середовищ, та покращення продуктивності тестувальників [3].

Для автоматизації тестів штучний інтелект дозволяє створювати тестові скрипти швидше та ефективніше. Це може бути досягнуто за допомогою різних інструментів автоматизації, таких як (Selenium) для веб-програм, (Appium) для мобільних додатків, або (JUnit/TestNG) для тестування на рівні коду. Застосування алгоритмів машинного навчання дозволяє ШІ створювати ефективні тестові сценарії на основі аналізу історії тестування та даних профілю програмного забезпечення.

Основним аспектом видалення помилок в штучному інтелекті є можливість застосовувати глибокий аналіз та проводити тестування реального часу.

Аналіз коду проводиться для виявлення можливих помилок, неправильного використання API та інших проблем в коді програмного продукту.

Застосування ШІ для моніторингу та аналізу роботи програмного забезпечення в реальному часі дозволяє вчасно виявляти та вирішувати проблеми, що виникають при реальному використанні.

Для глибокого аналізу використовується глибинний аналіз та метрики якості.

ШІ використовує глибинне навчання для обробки великих обсягів тестових даних, виявлення складних залежностей та автоматичної генерації тестових сценаріїв.

Алгоритми ШІ можуть використовувати дані про виконання тестів для автоматичного визначення ефективності тестових сценаріїв та забезпечення високої якості продукту.

В створенні реалістичних тестових середовищ відіграє важливу роль емуляція та створення тестових даних.

ШІ може емулювати реальні умови використання програмного продукту, включаючи різні типи мереж, обсяги даних та інші умови, що дозволяє виявляти проблеми, які можуть виникнути у реальному середовищі.

ШІ автоматично створює тестові дані, що допомагає в покращенні покриття тестування та виявленні проблем з обробкою даних.

Для покращення продуктивності тестувальників використовуються чат боти та рекомендації ШІ.

Використання тестових ботів для виконання рутинних завдань, таких як запуск тестів або аналіз результатів, дозволяє тестувальникам зосередитися на більш складних тестових сценаріях.

Системи рекомендацій можуть аналізувати результати тестів та надавати рекомендації щодо поліпшення стратегій тестування та зменшення часу на виявлення та виправлення помилок.

Ці аспекти допомагають зробити тестування більш ефективним, автоматизованим та зорієнтованим на виявлення якісних проблем у програмному забезпеченні. Такий підхід сприяє впровадженню продуктів в надійний та безпечний спосіб, забезпечуючи високу якість програмного забезпечення.

1.3. Застосування машинного навчання в сучасних тестових підходах

Машинне навчання (МО, Machine Learning, ML) - великий підрозділ штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися [4].

Історія штучного інтелекту. Артур Самуель розробив першу програму на основі алгоритмів [5], здатних самонавчатися, в 1952 році. Вона була призначена для гри в шашки. Крім того, Самуель дав перше визначення «машинного навчання» як «область досліджень розробки машин, які не є заздалегідь запрограмованими». Після цього Т. М. Мітчелл дав більш точне визначення терміну «навчання». За його словами, комп'ютерна програма навчається на основі досвіду E по відношенню до деякого класу завдань T і заходів якості P, якщо якість вирішення завдань T, виміряних на основі P, поліпшується з набуттям досвіду E.

У 1957 році була запропонована перша модель нейронної мережі, яка могла працювати з алгоритмами машинного навчання, подібними до сучасних. Наразі розробляються різні системи машинного навчання, які будуть використані в майбутніх технологіях, таких як Інтернет речей, промисловий Інтернет речей, «розумні» міста та безпілотний транспорт.

Такі факти свідчать про те, що машинне навчання зараз покладають великі надії.

1. Компанія Google сподівається, що в найближчому майбутньому її продукти «перестануть бути результатом традиційного програмування — в їх основу буде покладено машинне навчання».
2. Компанії Google, Facebook, Apple, Amazon, Microsoft і Baidu боролися за талановитих фахівців у сфері штучного інтелекту
3. .Генеральний директор Facebook Марк Цукерберг особисто бере участь у спробах переманити найкращих випускників через відеочат і телефон.
4. Найважливіші академічні конференції в цій галузі збільшилися майже в чотири рази.

5. Машинне навчання було використано для створення нових продуктів, таких як Siri від Apple, M від Facebook і Echo від Amazon.

Методи машинного навчання. Навчання по прикладах, також відоме як індуктивне навчання, і дедуктивне навчання є двома основними типами машинного навчання [5]. «Машинне навчання» і «навчання по прецедентах» можна вважати синонімами, оскільки останнє відноситься до сфери експертних систем. Як кажуть, експертні системи зараз у кризі, хоча цей підхід до навчання зараз є модним. Промислові СУБД неможливо використовувати для наповнення баз знань експертних систем, оскільки їхні основні бази знань важко узгодити з реляційною моделлю даних.

Навчання по прикладах є одним із трьох основних типів навчання. Перший включає контрольоване навчання, або навчання з учителем; другий — неконтрольоване навчання, або навчання без учителя; і третій — підкріплене навчання.

Крім відомих, розробляються різні підходи до навчання, наприклад активне, багатозадачне, різноманітне, трансферне та т. д. Останнім часом особливо успішно розвивається «глибоке навчання», яке дозволяє алгоритмам навчання успішно поєднуватися як з вчителем, так і без нього.

Контрольоване та неконтрольоване навчання. Контрольваний підхід до навчання застосовується в ситуаціях, коли є велика кількість даних, наприклад, тисячі фотографій домашніх тварин із маркерами, такими як мітки та ярлики, які вказують, що це кішка або собака. Необхідно розробити алгоритм, який дозволить машині визначити, чи зображена на фотографії собака чи кішка. В цьому випадку людина, яка заздалегідь проставила маркери, виступає як «вчитель». Машина сама визначає ознаки, за допомогою яких кішки відрізняються від собак. Таким чином, пізніше вона змогла швидко переналаштувати свій алгоритм, щоб він міг працювати з іншими завданнями, наприклад, розпізнавати качок і курей. Машина знову виконає складну і трудомістку роботу, щоб визначити ознаки, за якими вона буде розрізняти цих птахів. Крім того, можна навчити нейромережу, здатну розпізнавати кішок, швидко обробляти результати комп’ютерної томографії.

Даних без міток набагато більше, ніж маркованих, розмічених даних. Це зображення не має підпису, аудіозаписи не мають коментарів і тексти не мають анотацій. При неконтрольованому навчанні завданням машини є виявлення зв’язків між окремими даними, виявлення закономірностей, вибір шаблонів, упорядкування даних або опис їх структури, виконання класифікації даних. Неконтрольоване навчання використовується, наприклад, у рекомендаційних системах, коли клієнту в Інтернет-магазині пропонуються товари, які з більшою ймовірністю зацікавлять його, на основі аналізу його попередніх покупок. або коли користувачеві на порталі YouTube надаються десятки посилань на схожі відеокліпи після перегляду відеокліпу. або коли Google ранжує посилання в результатах пошуку для одного користувача по-різному залежно від історії пошуків.

Навчання з підкріпленням. Таке навчання є особливим видом контрольованого навчання, але вчителем є середовище. Хоча машина, яку в цій ситуації часто називають агентом, не має інформації про навколишнє середовище, вона має здатність виконувати будь-які дії. Середа реагує на ці дії, надаючи агенту дані для реагування та навчання. Фактично існує система зворотного зв’язку між агентом і середовищем.

Навчання з підкріпленням вирішує більш складні проблеми, ніж навчання з учителем або без нього. Воно використовується, наприклад, в системах навігації, які навчають роботів уникати зіткнень з перешкодами за допомогою досвіду, який вони отримують, отримуючи зворотний зв'язок про кожне зіткненні з перешкодою. Логістика, планування та складання графіків, а також навчання машини логічним іграм використовують підкріплення.

Нейронні мережі і глибоке навчання. Машинне навчання використовує різні алгоритми та технології, такі як [6]: дискримінантний аналіз, байєсовські класифікатори та багато інших математичних методів. Але в кінці XX століття все більше людей звертали увагу на штучні нейронні мережі (ANN). У 1986 році відбувся значний розвиток відомого як «Метод зворотного поширення помилки», який успішно використовували для навчання нейронної мережі, що призвело до нового зростання інтересу до них.

ANN є системою штучних нейронів, які з’єднані та працюють разом, на основі простих процесорів. Періодично кожен процесор ANN отримує сигнали від інших процесорів:або від сенсорів, або від інших джерел сигналів, а потім посилає ці сигнали іншим процесорам. У поєднанні ці прості процесори можуть виконувати досить складні завдання.

Зазвичай нейрони розташовуються в мережах за рівнями, які називають шарами. Як правило, це вхідні нейрони першого рівня. Вони отримують дані з інших джерел, наприклад, від сенсорів системи розпізнавання осіб, і обробляють їх, передаючи імпульси на наступному рівні через синапси нейронів. Нейрони на другому рівні обробляють імпульси, а потім передають їх нейронам на вихідному рівні. У процесі імітації нейронів кожен процесор вхідного рівня пов’язаний з кількома процесорами прихованого рівня, які в свою чергу пов’язані з кількома процесорами вихідного рівня. Такою є найпростіша архітектура ANN, яка здатна до навчання та знаходити прості взаємозв’язки в даних.

Глибоке навчання може бути застосоване лише до більш складних ANN з кількома прихованими рівнями. Коли це відбувається, рівні нейронів можуть чергуватися з шарами, які можуть виконувати складні логічні перетворення. Кожен наступний рівень мережі шукає зв’язки в першому рівні. Такі ANN можуть знаходити не тільки прості взаємозв’язки, але й міжзв’язки. Компанія Google вдалася значно підвищити якість свого популярного продукту «Перекладач» завдяки переходу на нейромережу з глибинним навчанням. Зокрема, якість перекладу між англійською та французькою мовами підвищилася на 7 балів, що становить більш ніж 20%. За весь час свого існування з 2006 року попередня система, яка виконувала фразовий статистичний машинний переклад, домоглася подібного поліпшення [11].

1.4. Проблематика тестування та аналіз існуючих рішень

Тестування програмного забезпечення — це процес, який допомагає виявити помилки, несправності або дефекти в програмному забезпеченні [7].

Крім того, він перевіряє відповідність клієнта вимогам замовника. Методи тестування відрізняються від рівня тестування. Перед початком процесу слід розглянути переваги та недоліки кожного методу тестування.

1. Виявлення помилок і дефектів:

• Раннє виявлення проблем: Тестування дозволяє виявляти помилки та дефекти в програмному забезпеченні на ранніх стадіях розробки, коли виправлення їх ще найменше витратно.

• Забезпечення якості: Вірне тестування сприяє поліпшенню якості програмного продукту, допомагає уникати виходу програм з недоліками та підвищує задоволення користувачів.

2. Відповідність вимогам і специфікаціям:

• Перевірка відповідності вимогам: Тестування включає в себе перевірку відповідності розробленого програмного продукту вимогам та специфікаціям, що допомагає уникнути невідповідні функціональності чи вимог замовника.

• Визначення вірності виконання задач: Тестування визначає, чи програмне забезпечення виконує свої функції та завдання з відповідною ефективністю та точністю.

3. Економія коштів та часу:

• Мінімізація витрат на виправлення помилок: Виявлення та виправлення проблем на ранніх етапах розробки обходиться дешевше, ніж у випадку, якщо помилки виявляться пізніше, наприклад, під час експлуатації продукту.

• Прискорення часу до випуску: Тестування допомагає вчасно виявляти проблеми та зменшує час, який витрачається на вирішення неполадок, що полегшує випуск продукту.

4. Забезпечення надійності та стабільності:

• Підвищення надійності: Тестування спрямоване на забезпечення стабільності та надійності програмного продукту, що є ключовими аспектами при його використанні.

• Мінімізація ризиків неполадок: Тестування допомагає мінімізувати ризики виникнення неполадок та забезпечує високу якість програмного забезпечення.

5. Забезпечення відкритості та прозорості:

• Відкритість коду: В інтересах тестування входить розкриття можливих проблем та покращення взаєморозуміння між розробниками та тестувальниками.

• Створення документації: Процес тестування включає в себе створення тестової документації, що забезпечує прозорість у відносинах між командами та в документуванні функціональностей.

6. Забезпечення безпеки:

• Тестування безпеки: Важливою складовою тестування є перевірка програмного забезпечення на вразливість до атак та забезпечення високого рівня безпеки.

• Уникнення витоку конфіденційної інформації: Тестування включає в себе перевірку наявності витоків конфіденційної інформації та забезпечення заходів безпеки.

7. Сприяння задоволенню користувачів:

• Забезпечення функціональності: Тестування впевнюється, що програмний продукт відповідає очікуванням користувачів та задовольняє їхні вимоги.

• Максимальне використання функціональностей: Тестування допомагає впевнитися, що всі функціональності програми працюють належним чином, що призводить до задоволення користувачів.

Люди проводять традиційне тестування ПЗ, а програмісти надають вхідні дані та логіку. Машина перевіряє логіку системи та бажану поведінку. Програмісти вводять дані та бажану поведінку для створення логіки програми під час тестування систем на основі машинного навчання. Систему неодноразово перевіряють на логіку, щоб переконатися, що вона залишається послідовною. Система розуміє логіку та розробляє модель для бажаної поведінки після того, як вона її засвоїла (рис. 1.1). Порівняємо тестування традиційного програмного забезпечення з тестуванням систем на основі машинного навчання.

Код є компонентом тестування традиційного програмного забезпечення, який викликає помилки та несправності. Більшість проблем з машинним навчанням виникають через погані набори даних для навчання, програму навчання самої системи та неправильне використання фреймворку. Програмний код, який був написаний за визначеними вимогами в традиційному програмному забезпеченні (ПЗ), буде завжди видавати однакові результати під час тестування. Однак результати машинного навчання можуть відрізнятися через постійне оновлення навчального набору даних. Машинне навчання використовує тестові дані для тестування програмного коду, тоді як традиційне тестування використовує різні форми даних.

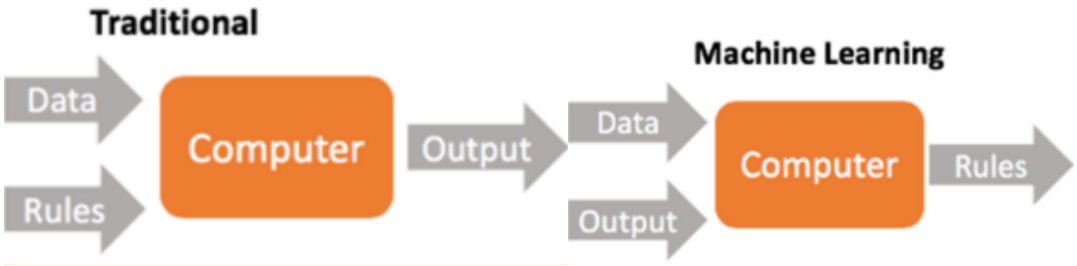


Рис. 1.1 Традиційне тестування проти тестування машинного навчання

У традиційному ПЗ [8] припускається, що завжди існує тестовий оракул, що дозволяє розробнику перевірити результат ПЗ з результатом, який був визначений наперед. Розгортання Інтернету вже використовує системи на основі машинного навчання. У більшості випадків правильність великої кількості згенерованих відповідей підтверджується вручну. У даний час виявлення тестових оракулів залишається складним завданням, оскільки багато бажаних властивостей важко формально визначити. Ідентифікація оракула, навіть якщо це стосується конкретної предметно-специфічної проблеми, все ще займає багато часу та зусиль, оскільки часто потрібні знання, пов’язані з предметною областю. У сучасному світі компанії зазвичай покладаються на ручне маркування даних, що може бути дорогим, щоб отримати їх.

Критерії адекватності тесту: критерії адекватності тесту використовуються для забезпечення ступеня цільового програмного забезпечення, яке було

протестоване. До цього часу було запропоновано багато промислових критеріїв адекватності, таких як покриття ліній, покриття гілок і покриття потоку даних. Але необхідні нові критерії адекватності тестування, щоб враховувати характеристики програмного забезпечення машинного навчання, оскільки є значні відмінності між парадигмою програмування та форматом представлення логіки для програмного забезпечення машинного навчання та традиційним програмним забезпеченням.

Помилкові спрацьовування: тестування машинного навчання має тенденцію давати більше хибних спрацьовувань у повідомленнях про помилки через труднощі з встановленням надійних стандартів тестування.

Ролі тестувальників: помилки в тестуванні машинного навчання можуть виникнути як у навчальних програмах, так і в даних або алгоритмах, тому спеціалісти з обробки даних або розробники алгоритмів також можуть виконувати функції тестувальників. В таблиці 1.1. наведено порівняння тестування та тестуванням систем на основі машинного навчання

Таблиця 1.1.

Порівняння між традиційним тестування та тестуванням систем на основі машинного навчання

Характеристика	Традиційне тестування	Тестування систем на основі машинного навчання
Компонент тестування	Код	Дані та код (фреймворк)
Поведінка під час тестування	Визначений	Змінюється з часом
Критерії адекватності	оцінка охоплення/мутації	Не визначено
Тестові дані	Дані	Дані або код
Помилкові спрацювання	Рідко	Часто

Тестувальники	Розробник	Розробник даних, алгоритмів

1.5. Загальний огляд інструментів для автоматизації тестування

Інструменти для автоматизації тестування важливі для забезпечення ефективного та швидкого тестування програмного забезпечення [9]. Ці інструменти включають в себе різні фреймворки, платформи та інструментарій, які допомагають автоматизувати виконання тестових сценаріїв, порівняння результатів та аналіз відповідності очікуваним результатам. Нижче подано загальний огляд деяких популярних інструментів для автоматизації тестування:

Selenium:

Тип: Фреймворк для автоматизації тестування веб-додатків.

Мови програмування: Підтримує Java, C#, Python, Ruby, JavaScript.

Особливості: Розширюваність, крос-браузерна сумісність, можливість інтеграції з різними інструментами.

Appium:

Тип: Фреймворк для автоматизації тестування мобільних додатків.

Мови програмування: Підтримує Java, C#, Python, Ruby, JavaScript.

Особливості: Крос-платформенність, підтримка для iOS та Android, використання стандартних API.

JUnit:

Тип: Фреймворк для тестування на рівні коду в мові Java.

Мови програмування: Java.

Особливості: Простота використання, анотації для позначення тестів, можливість групування та виконання тестів.

TestNG:

Тип: Фреймворк для тестування на рівні коду та функціонального тестування.

Мови програмування: Підтримує Java.

Особливості: Розширені можливості анотацій, паралельне виконання тестів, підтримка параметризації.

Postman:

Тип: Інструмент для автоматизації тестування API.

Мови програмування: JavaScript (для скриптів).

Особливості: Зручний інтерфейс, можливість написання та виконання автоматизованих тестів API.

Jenkins:

Тип: Інструмент для автоматизації CI/CD процесів.

Мови програмування: -

Особливості: Забезпечує неперервну інтеграцію, розгортання та виконання автоматичних тестів під час збірки.

TestComplete:

Тип: Комплексний інструмент для автоматизації тестування різних типів додатків.

Мови програмування: JavaScript, Python, VBScript.

Особливості: Підтримка різних платформ, запис та відтворення тестів, велика кількість вбудованих функцій.

Robot Framework:

Тип: Фреймворк для автоматизації тестування зі зрозумілим DSL.

Мови програмування: Python (основна), також підтримує Java, .NET.

Особливості: Зручний синтаксис, підтримка ключових слов, можливість розширення.

SikuliX:

Тип: Інструмент для автоматизації тестування GUI на основі зображень.

Мови програмування: SikuliScript (спеціальна мова).

Особливості: Відсутність прив'язки до інтерфейсу, визначення об'єктів за зразком.

Ці інструменти можуть використовуватися окремо чи в поєднанні для створення ефективної системи автоматизації тестування. Вибір конкретного інструменту залежить від типу додатка, його технічних особливостей, мов програмування, якими користуються розробники, та інших факторів.

РОЗДІЛ 2.
ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

2.1. Розгляд сучасних інструментів для автоматизації тестування та їх основних функцій**.**

Сучасні інструменти для автоматизації тестування мають різноманітні функції, які дозволяють розробникам та тестувальникам створювати, виконувати та аналізувати автоматизовані тести [10]. На рисунку 2.1. зображено бібліотеки та їх драйвер. Популярні на ринку інструменти, для керування тестими є [11]:

1) ***Selenium*** — це інструмент тестування з відкритим вихідним кодом, який призначений для автоматизації веб-додатків. Selenium надає застосунок запису/відтворення, що дозволяє створювати тести вебзастосунків без вивчення мов програмування. Основні функції програмного середовища є:

- ***Selenium IDE*** — інтегроване середовище розробки у вигляді Firefox-додатка, який дозволяє записувати та відтворювати тести в Firefox 2+.
- ***Selenium Client API*** — набір API, що дозволяє писати тести на Java, C#, Ruby, JavaScript та Python.
- ***Selenium Remote Control*** — це клієнт / серверна система, яка дозволяє керувати веббраузерами локально або на іншому комп'ютері, використовуючи практично будь-яку мову програмування та тестування системи.
- ***Selenium WebDriver*** — драйвер що дозволяє керувати веббраузером за допомогою Selenese або API.
- ***Selenium Grid*** — дозволяє одночасно запускати тести на кількох серверах та типах веббраузерів зменшуючи час на тестування.

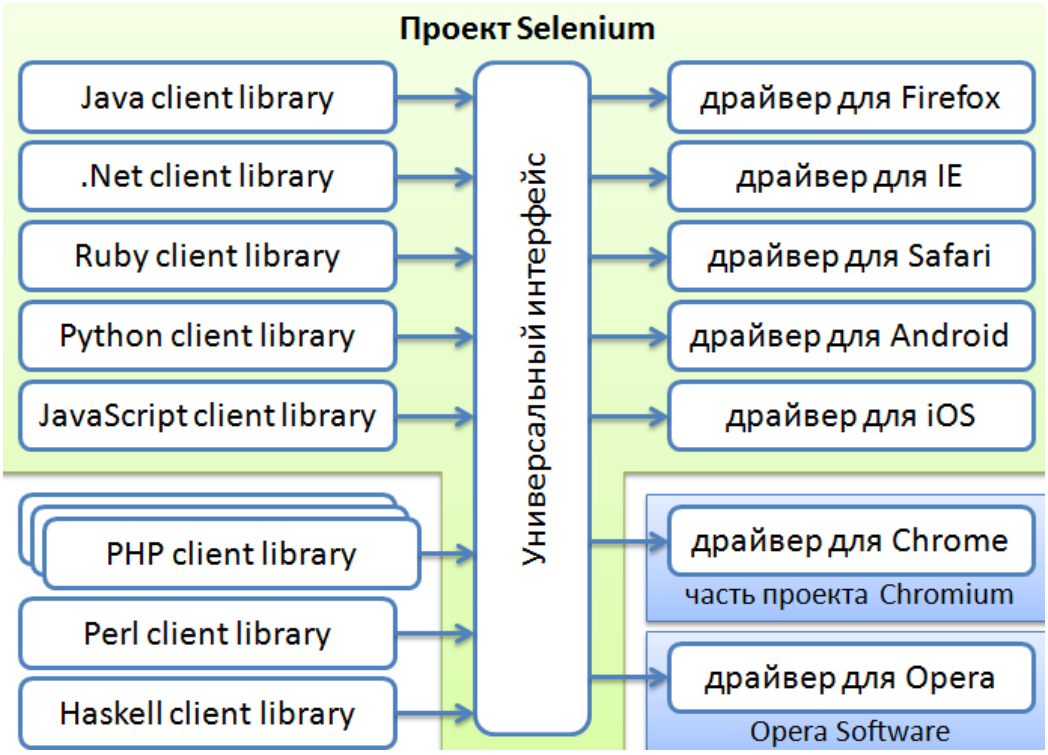


Рис. 2.1. Універсальний інтерфейс Selenium

2) ***Autify***- це безкодова платформа автоматизованого тестування програмного забезпечення, яка використовує ШІ, що самовідновлюється, для створення, керування та виконання тестів (рис. 2.2). Випадки використання:

- ***Кроссбраузерне тестування.*** Підтримує як ПК, так і мобільні браузери. Це усуває потребу в управлінні та обслуговуванні реальних пристроїв або ферм пристроїв

Рис. 2.2. Середовища виконання Autify для кроссбраузерного тестування

- ***Тестування електронної адреси.*** Може проводити тести переходу та перевіряти вміст електронної пошти, такі як електронні листи про реєстрацію нових користувачів і підтвердження покупок. Немає необхідності підготувати середовище вхідної електронної пошти, яке використовується лише для тестування. На рисунку 2.3. представлено інтерфейс вибору тестування Autify.

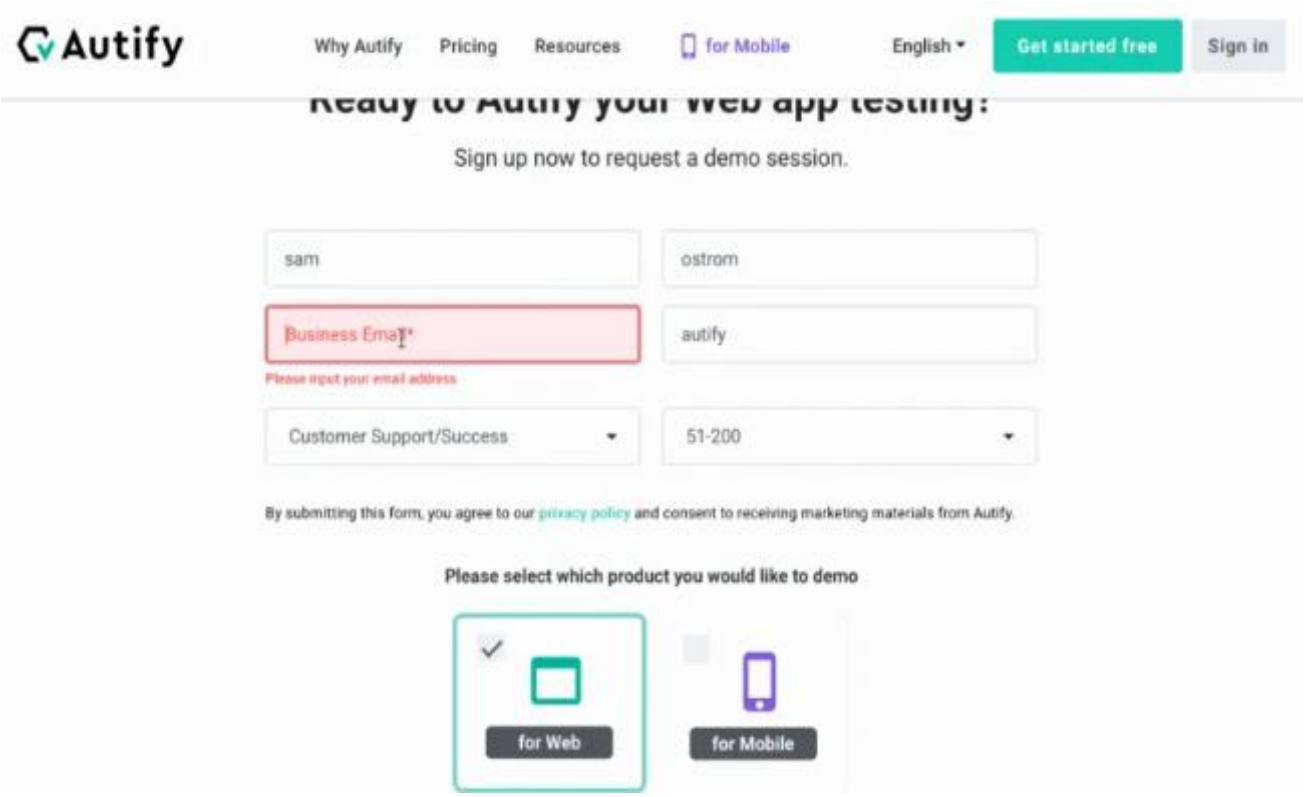


Рис. 2.3. Інтерфейс Autify для вибору тестування електронної адреси

- ***JavaScript Step*** дає можливість налаштовувати тестові приклади, комбінуючи власні коди та аргументи (рис 2.4)
- ***Паралельне тестування.*** Ви можете запускати кілька тестів одночасно. Кількість паралельних виконань можна налаштувати залежно від вашого варіанту використання
- Step ***Groups*** об’єднує набір дій у групі кроків. Ця функція економить час, при використання однакових дій в різних тестових сценаріях або коли дії повторюються в сценарії (рис 2.5).

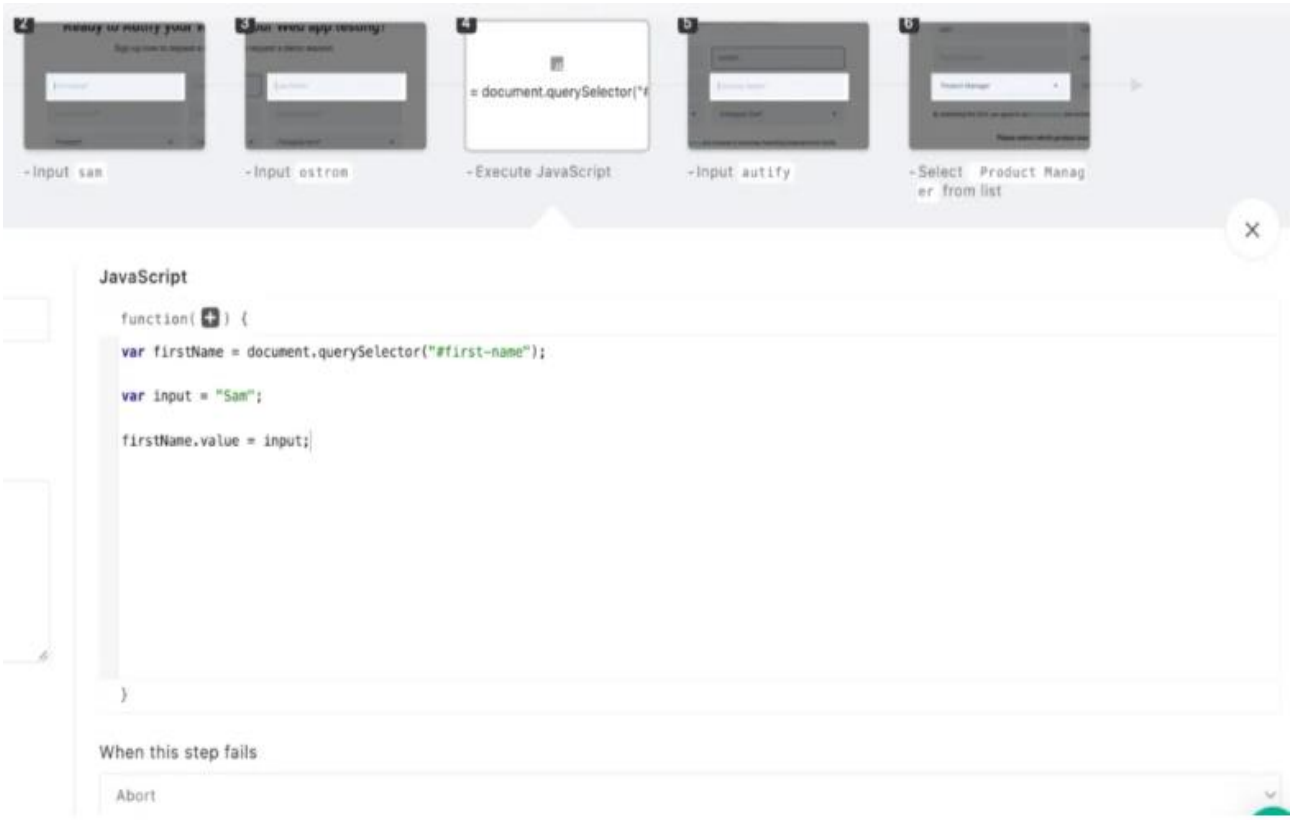


Рис. 2.4. Інтерфейс Autify програмного середовища на JavaScrip

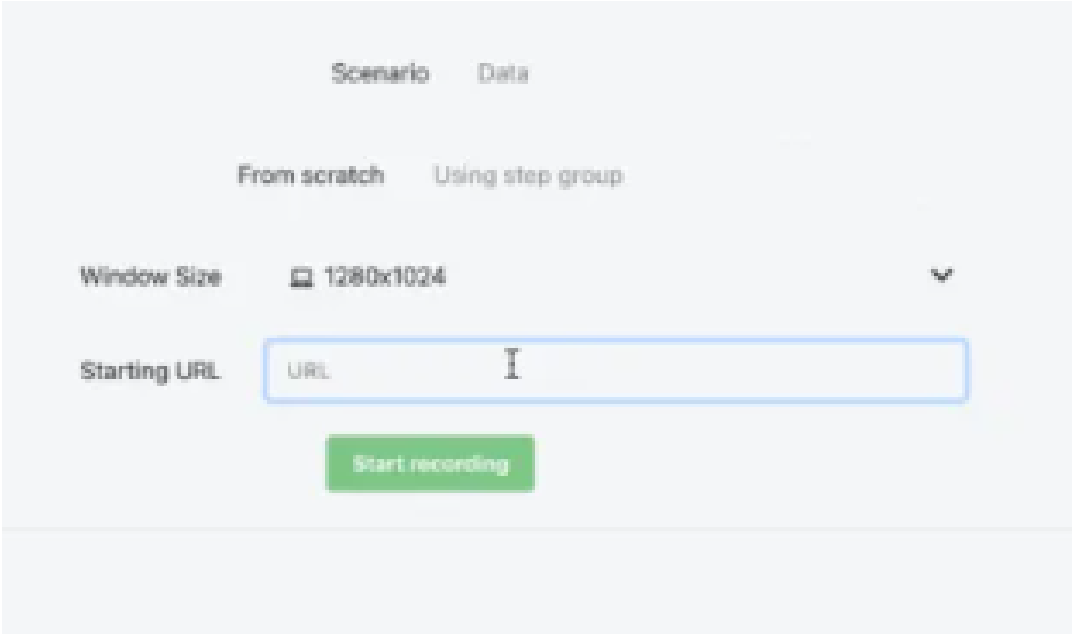


Рис. 2.5. Інтерфейс для функціоналу Steap Group

- ***Shadow DOM.*** Autify підтримує тіньові елементи DOM, які стають стандартом у Salesforce. Можливо протестувати програми, доступні на торгових майданчиках, наприклад Salesforce AppExchange.

Для забезпечення ефективного керування тестуванням програмного забезпечення має вирішальне значення для забезпечення якості та надійності програмних додатків. Відстежування тестових випадків, оптимізація процесу тестування та управління тестовими середовищами залежать від інструментів і програмного забезпечення для керування тестуванням. Популярними інструментами та програмні забезпечення для керування тестами є:

- ***Applitools*** скорочує час, необхідний для створення, виконання та підтримки автоматизованих тестів, замінюючи традиційне функціональне тестування штучним інтелектом. ***Applitools*** скорочує ручну роботу протягом усього процесу тестування, як-от створення, аналіз, виконання тестів, і дозволяє командам тестувати експоненціальну кількість своїх програм.
- ***Тестування інтерфейсу користувача.*** За допомогою тестування інтерфейсу користувача ви перевіряєте роботу веб-програми за допомогою імітації введення користувача. Наприклад, використовуючи інфраструктуру тестування Cypress.io , ви можете імітувати дії користувача під час відкриття веб-браузера, пошуку та натискання кнопки на сторінці або навіть введення тексту в поле введення. Це ідеально працює та може стати чудовою підмогою для тестування сторони інтерфейсу користувача вашої програми шляхом перевірки функціональності сторінок і того, як вони працюватимуть у виробництві.

Однак функціональне тестування інтерфейсу користувача є лише одним з аспектів тестування, і цього одного недостатньо для повного візуального тестування вашої програми. Ми повинні враховувати такі речі, як зміни у макеті, розмірі чи розташуванні компонентів на сторінці, зміни у форматуванні тексту чи вмісті та багато інших візуальних змін, які важче виявити за допомогою традиційного функціонального тестування. Особливо це стосується адаптивних веб-додатків , які змінюють макет залежно від розміру вікна перегляду.

- ***Тестування візуального інтерфейсу.*** Автоматизоване візуальне тестування інтерфейсу користувача — це форма регресійного тестування, яке

виконується за певними кроками, щоб перевірити, чи екрани чи сторінки не змінилися несподівано від одного тестового запуску до іншого. На малюнку нижче зображено типовий тестовий випадок, що моделює дії користувача для відкриття веб-сайту «Вікіпедія» та пошуку за ключовим словом «Програмне забезпечення».

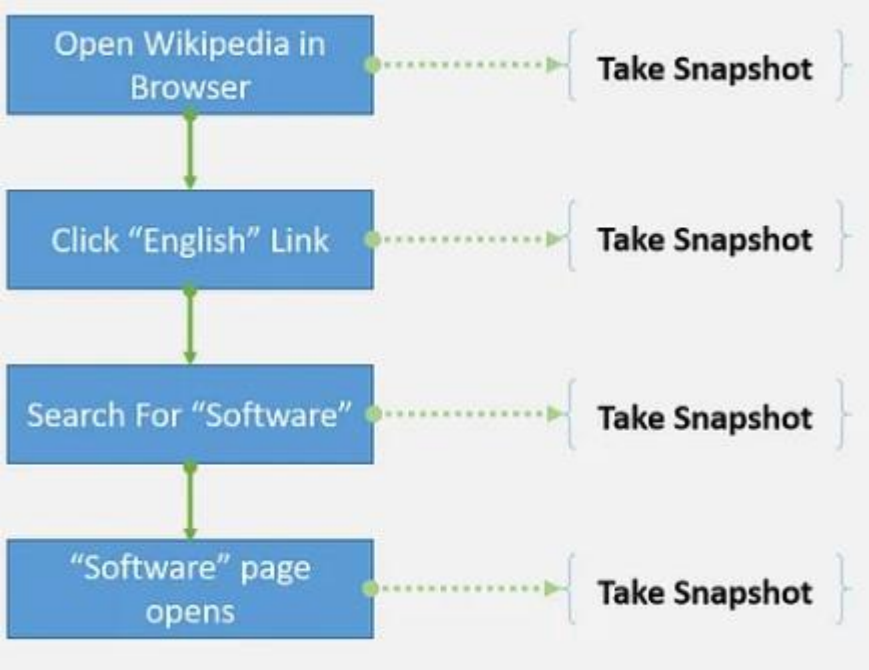


Рис. 2.6. Процес тестування візуального інтерфейсу

Процес тестування візуального інтерфейсу (рис. 2.6) користувача починається зі створення типового інтеграційного тесту з використанням тих самих інструментів і наборів тестів, які ви зазвичай використовуєте для виконання таких інтеграційних тестів. Тест інтеграції – це серія дій, які виконуються на екранах/сторінках програми, імітуючи фактичного користувача.

- ***Принцип роботи Applitools.*** Applitools використовує той самий підхід, що й вище, надаючи інструменти, які можна використовувати для створення знімків, надсилання знімків у хмару Applitools і запуску логіки штучного інтелекту для порівняння знімків із відповідними базовими лініями.

Applitools надає багатий набір комплектів розробки програмного забезпечення (SDK) для відомих наборів тестів. Наприклад:

- SDK для системи наскрізного тестування Cypress.io (нещодавно запущено)
- SDK для Storybook з React , Angular і Vue (нещодавно запущено)
- SDK для майже всіх різновидів Selenium (JavaScript , Java , C# , Ruby тощо).
- Пакети SDK майже для всіх різновидів Appium (власний C# , рідний Java , рідний PHP тощо).

Використовуючи Applitools SDK у своїх тестових випадках, ви можете попросити SDK фіксувати стан програми, коли забажаєте. Цей процес виглядає наступним чином:

- SDK робить знімок браузера
- Отриманий знімок надсилається на сервер Applitools
- Якщо попереднього базового зображення немає, сервер зберігає зображення як базове для порівняння в майбутніх тестових прогонах
- Якщо було попереднє базове зображення, Сервер запускає механізм ШІ, щоб порівняти базове зображення з новим знімком. Механізм повідомляє, чи збігаються зображення, або механізм повідомляє про будь-які розбіжності між двома зображеннями.
- Після завершення тестів створюється докладний звіт про відмінності між різними знімками та їхніми відповідними базовими лініями.

2.2. Структура збереження даних в системі тестування

Тестові запитання можуть бути класифіковані за різними критеріями, такими як тип, рівень складності, тема тощо в системах тестування зі збереженням даних реляційних СКБД [12]. Тим не менш, для більш детального розподілу в рамках однієї категорії необхідно створювати нові поля та нові таблиці. Крім того, встановлення рівнів вкладеності тем, що є складною задачею для реляційних СКБД, призводить до перевантаження даних і ускладнення структурних зв’язків. У системах, які зберігають кожен тест у окремому файлі,

неможливе повторне застосування вже існуючої ієрархії. Щоб розширити тест з опціональними завданнями, потрібно створити новий файл на основі існуючого файлу з відповідними параметрами.

Для спрощення роботи з даними необхідно обрати правильний метод збереження даних, який дозволяє зберігати ієрархічність структури даних. Для цієї роботи були розглянуті моделі даних, які представлені на рис. 2.7, і було обрано ієрархічну модель, оскільки вона дозволяє оптимально зберегти структуру даних і візуалізувати їх для зручності роботи з даними.

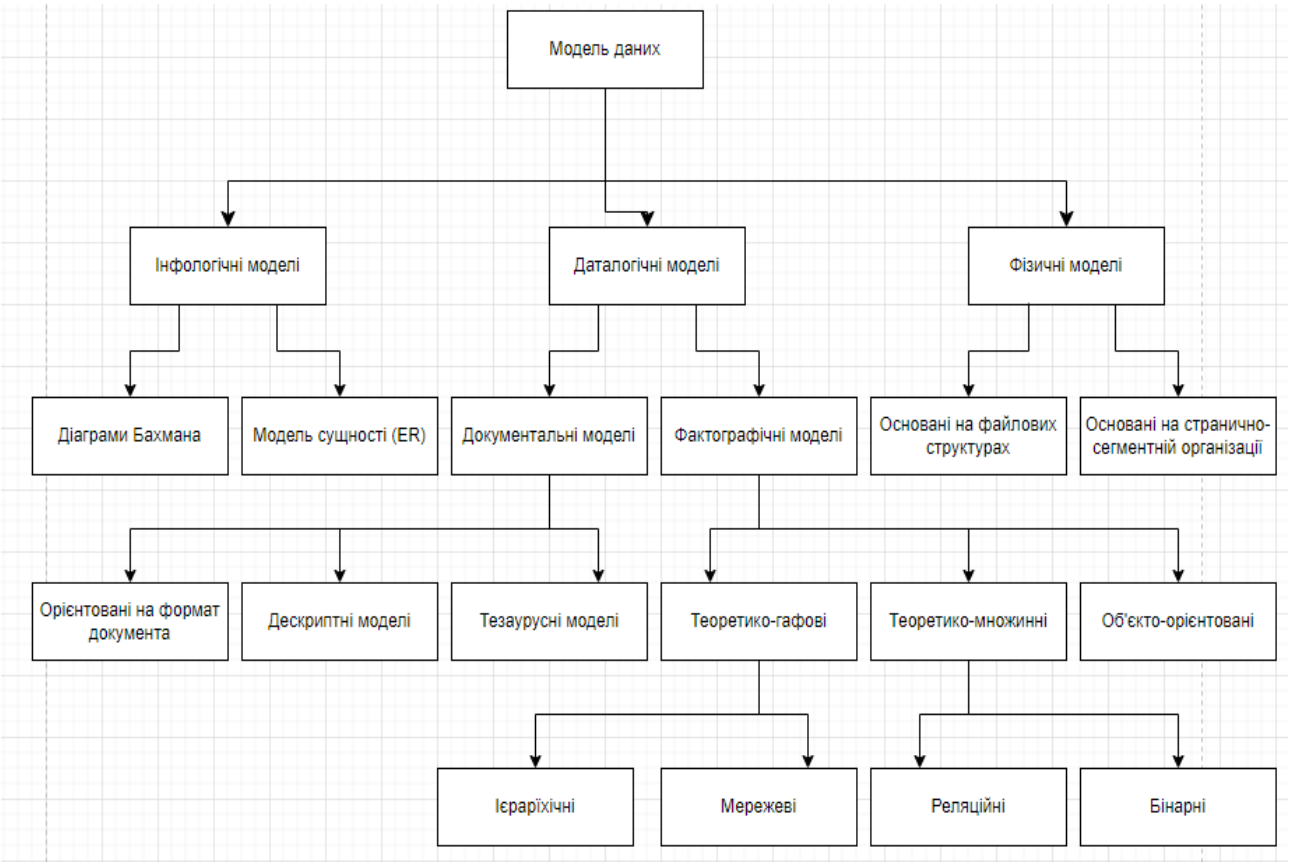


Рис. 2.7. Моделі даних

2.3. Дослідження можливостей машинного навчання для прискорення виконання тестів

Дослідження можливостей машинного навчання для прискорення виконання тестів включає в себе використання алгоритмів та моделей

машинного навчання для оптимізації тестових процесів. Основні напрями дослідження в цьому контексті включають [13]:

1) Вибіркове тестування - це стратегія тестування програмного забезпечення, при якій обираються певні компоненти чи функціональні частини програми для проведення тестів. Воно базується на ідеї, що не всі частини програми мають однаковий рівень критичності чи важливості, і, отже, важливо зосередити увагу на тих областях, де ймовірність виявлення помилок вища або де помилки можуть мати більший вплив на програму чи користувачів.

З самого початку починається збір даних, що включає в себе аналіз дефектів та помилок в минулих версіях програмного продукту, кодові заміни, розмір модулів та час, який потрібний для виправлення помилок.

Наступним кроком є визначення критеріїв критичності. Цей крок потрібен для визначення критичності частин програм, що включає в себе:

- Аналіз важливості модулів для функціональності системи
- Кількість виявлених дефекків у певних частинах коду
- Вплив помилок на користувачів.

Після визначення критеріїв критичності потрібно вибрати модель машинного навчання. Потрібно вибрати модель машинного навчання, яка підходить для завдання класифікації або регресії. Можливі моделі включають дерева рішень, метод опорних векторів (**SVM**), нейронні мережі, чи модель призначення ваг. Наприклад лоя визначення ваги області коду використовується формула:

Модель призначення ваг (Weight Assignment Model):

$$W_i = \frac{1}{1 + e^{-z_i}}$$

Де W_i – вага області коду i, z_i-сума вагованих характеристик області коду

Наступним кроком виступає підготовка данних для введення в модель, включаючи видалення непотрібних або дубльованих факторів, вибір ознак та нормалізацію даних. Приведемо приклад:

Припустимо, що у вас є датасет, який містить інформацію про різні частини програмного забезпечення та їх характеристики. Якщо деякі фактори не

мають важливого впливу на критичність або вже мають велику кореляцію з іншими факторами, їх можна вилучити (рис. 2.8).

```
# Використовуючи бібліотеку Python Pandas
df = df.drop(['непотрібний_фактор1', 'непотрібний_фактор2'], axis=1)
# df - наш датасет
```

Рис. 2.8. Вилучання елементів з датасету з використанням бібліотеки Python Pandas

Якщо в датасеті є дубльовані або ідентичні фактори, їх слід вилучити для запобігання зайвого завантаження моделі (рис. 2.9).

```
# Використовуючи бібліотеку Python Pandas
df = df.T.drop_duplicates().T
# df - наш датасет
```

Рис. 2.9. Видалення дубльованих факторів

Для вибору факторів, які найбільше впливають на цільовий показник (у випадку визначення критичності частин програми (рис. 2.10).

```
# Використовуючи бібліотеку Python Pandas
features = df[['фактор1', 'фактор2', 'фактор3']]
# df - наш датасет
```

Рис. 2.10. Вибір ознак

Нормалізація даних (рис. 2.11) важлива для забезпечення того, що всі фактори мають однаковий масштаб. Це допомагає уникнути перекосів у вагах при навчанні моделі.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
normalized_features = scaler.fit_transform(features)
```

Рис. 2.11. Нормалізація даних

Наступним кроком в вибірковому тестуванні, для визначення найбільш критичних та важливих частин програмного забезпечення (рис. 2.12) для тестування є навчання моделі, на основі підготовлених даних, де алгоритм вивчає зв'язки між вхідними факторами та критичністю частин програми.

Приклад навчання моделі на основі підготовлених даних за допомогою алгоритму класифікації. У цьому прикладі будемо використовувати бібліотеку Python Scikit-learn та алгоритм дерева рішень для визначення критичності частин програми.

```
# Імпорт бібліотек
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Підготовка даних (normalized_features - вже підготовлені та нормалізовані фактори, target - цільовий показник)
X_train, X_test, y_train, y_test = train_test_split(normalized_features, target, test_size=0.2, random_state=42)

# Ініціалізація та навчання моделі дерева рішень
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Прогнозування на тестових даних
y_pred = model.predict(X_test)

# Оцінка ефективності моделі
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:\n", report)
```

Рис. 2.12. Визначення критичності частин програми.

У цьому коді:

- Дані розділяються на тренувальний та тестовий набори.

- Ініціалізується модель дерева рішень та навчається на тренувальних даних.
- Проводиться прогноз на тестових даних.
- Обчислюється точність моделі та генерується звіт про класифікацію.

Після визнначення зв'язків між вхідними факторами та критичністю частин програми, потрібно дати оцінку моделі (рис. 2.13). Вона визначається за допомогою валідації на тестовому наборі даних, та використовується для перевірки її точності та генералізації, щоб переконатись в її точності та здатності генералізації на нових даних. Використаємо приклад для продовження коду, який ми розглядали раніше:

```
# Імпорт бібліотек
from sklearn.model_selection import cross_val_score, StratifiedKFold

# Створення моделі та визначення крос-валідації
model = DecisionTreeClassifier()
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Оцінка моделі за допомогою крос-валідації
cv_results = cross_val_score(model, normalized_features, target, cv=kfold, scoring='accuracy')

# Виведення результатів
print("Cross-Validation Accuracy Mean:", cv_results.mean())
print("Cross-Validation Accuracy Standard Deviation:", cv_results.std())
```

Рис. 2.13 Визначення оцінки ефективності моделі

В даному прикладі:

- ***StratifiedKFold*** використовується для розділення даних на стратифіковані фолди (зберігає пропорції цільового показника в кожному фолді).
- ***cross_val_score*** оцінює модель за допомогою крос-валідації та повертає масив точностей для кожного фолда.
- Середнє та стандартне відхилення точності виводяться для оцінки ефективності моделі.

Цей підхід дозволяє оцінити точність моделі на різних підвибірках даних та визначити, наскільки добре вона генералізується на нові дані. Важливо

враховувати, що ефективність моделі може залежати від конкретного датасету та обраного методу оцінки.

Наступним кроком є визначення критичних частин. Для цього використовують навчену модель для визначення критичності частин програми. Модель може виділяти та ранжувати їх за ступенем важливості.

Після навчання моделі, ви можете використовувати її для прогнозування критичності частин програми. На рисунку 2.14. наведено приклад використання навченої моделі для ранжування важливості частин програми:

```
# Припустимо, що у вас є новий датасет або дані, для яких ви хочете визначити критичність
new_data = ...

# Нормалізуємо нові дані (використовуючи той самий MinMaxScaler, який ми використовували для тренувальних даних)
normalized_new_data = scaler.transform(new_data)

# Використовуємо навчену модель для прогнозування критичності
predicted_criticality = model.predict(normalized_new_data)

# Отримання ймовірності прогнозу для кожного класу (1 - критична, 0 - не критична)
probability_of_criticality = model.predict_proba(normalized_new_data)[:, 1]

# Виведення результатів
print("Прогноз Критичності:", predicted_criticality)
print("Ймовірність Критичності:", probability_of_criticality)
```

Рис. 2.14. Приклад використання навчаної моделі

Важливо враховувати, що значення, отримані з прогнозу моделі, можуть бути використані для ранжування частин програми за ступенем їхньої важливості. Наприклад, ви можете відсортувати частини програми за зростанням ймовірності критичності (рис. 2.15).

```
# Створення DataFrame з новими даними та прогнозами
result_df = pd.DataFrame({'Частина Програми': частини_програми, 'Прогноз Критичності': predicted_criticality,
'Ймовірність Критичності': probability_of_criticality})
# Сортування за Ймовірністю Критичності
result_df_sorted = result_df.sort_values(by='Ймовірність Критичності', ascending=False)

# Виведення відсортованого DataFrame
print(result_df_sorted)
```

Рис. 2.15. Сортування за ймовірністю критичності

У цьому прикладі ми отримаємо відсортований список частин програми разом з прогнозами критичності та ймовірності.

Останнім кроком для визначення, які частини програми є найбільш критичними та важливими для тестування є інтеграція з процесом тестування.

Інтеграція результатів моделі з процесом тестування може включати автоматизоване визначення пріоритетів тестів у важливих областях програмного забезпечення. Наприклад (рис 2.16):

```
# Припустимо, що у нас є тестовий набір даних для визначення пріоритетів
test_data = ...

# Нормалізуємо тестові дані (використовуючи той самий MinMaxScaler, який ми використовували для тренувальних даних)
normalized_test_data = scaler.transform(test_data)

# Прогнозуємо критичність за допомогою навченої моделі
predicted_criticality_test = model.predict(normalized_test_data)

# Отримання ймовірності прогнозу для кожного тесту
probability_of_criticality_test = model.predict_proba(normalized_test_data)[:, 1]

# Додаємо прогнози до тестового набору даних
test_data['Прогноз Критичності'] = predicted_criticality_test
test_data['Ймовірність Критичності'] = probability_of_criticality_test

# Визначаємо пріоритети тестів на основі прогнозів
high_priority_tests = test_data[test_data['Прогноз Критичності'] == 1]
medium_priority_tests = test_data[test_data['Ймовірність Критичності'] > 0.5]
low_priority_tests = test_data[test_data['Ймовірність Критичності'] <= 0.5]

# Виведення результатів або інтеграція з системою управління тестуванням
print("Високий пріоритет тестів:")
print(high_priority_tests)

print("\nСередній пріоритет тестів:")
print(medium_priority_tests)

print("\nНизький пріоритет тестів:")
print(low_priority_tests)
```

Рис. 2.16. Інтеграція результатів моделі з процесом тестування

У цьому прикладі тестові дані розширюються із прогнозами, і вони подаються для визначення пріоритетів. Високий пріоритет може бути призначений тестам, які модель визначила як критичні, або тим, для яких ймовірність критичності більше 0.5. Це може бути інтегровано в процес

управління тестуванням для автоматизованої розстановки пріоритетів тестам у важливих областях.

2) Автоматизована генерація тестових сценаріїв - це процес використання програмних засобів для створення тестових сценаріїв без прямої участі людини. Цей підхід дозволяє ефективно та швидко генерувати тестові випадки, особливо в умовах великих та складних програмних систем.

Для автоматизованої генерації тестових сценаріїв можна використовувати різні методи та інструменти. До них відносяться:

1) Визначення критеріїв покриття [13], що в тестуванні програмного забезпечення включає в себе визначення обсягу тестового покриття, тобто того, наскільки ефективно тести покривають різні аспекти програми.

Для прикладу визначемо критерій рядків коду(Code Coverage)

Нехай:

- S-загальна кількість рядків коду (Statemants) в програмі
- E – кількість виконаних рядків коду

Тоді критерій покриття рядків коду (Statement Coverage) обчислюється за формулою:

$$Statement\ Coverage = \frac{E}{S} \cdot 100\%$$

Ця формула показує відсоток виконаних рядків коду відносно загальної кількості рядків. Чим вищий відсоток, тим більше коду було протестовано. Такий критерій дозволяє визначити, наскільки повно тестовий набір покриває програмний код.

2) Створення моделі для генерації тестових сценаріїв ***(Test Scenario Generation Model)*** [14] може бути складним завданням, і воно може ґрунтуватися на різних методах та технологіях, таких як штучний інтелект, генетичні алгоритми, машинне навчання, чи інші підходи. Для прикладу обрахуємо ймовірність включення тестового сценарію:

$$P(T_i) = \frac{1}{1 + e^{-\beta \cdot x_i}}$$

Де $P(Ti)$ ймовірність включення тестового сценарію i, X_i вектор характеристик області коду для тестового сценарію, β ваговий коефіцієнт.

3) Використання методів штучного інтелекту, таких як генетичні алгоритми чи машинне навчання, для автоматизованої генерації тестових сценаріїв [15]. Моделі можуть навчатися на основі історії тестування та інших факторів для створення більш ефективних сценаріїв.

Для прикладу використаємо бібліотеку машинного навчання “scikit-kearn”

Спочатку імпортуємо бібліотеки (рис. 2.17)

```
# Імпорт бібліотек
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

Рис. 2.17. Імпорт бібліотек

Припустимо, в нас є історія тестування у форматі DataFrame, де “features”- це фактори, а “label” – це мітка, чи тест був успішним (1), чи невдалим (0). Наш DataFrame повинен містить дані про тестові сценарії та їх результати.

Для зчитування даних або використання власних, використовуємо (рис. 2.18)

```
df = pd.read_csv("your_test_data.csv")
```

Рис. 2.18. Завантаження тестового сценарія

Далі нам потрібно розділити дані на тренувальний та тестовий набори (рис. 2.19)

```
features = df.drop('label', axis=1)
labels = df['label']
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
```

Рис. 2.19. Розділення набору данних

Для нормалізації даних використовуємо (рис. 2.20)

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Рис. 2.20. Нормалізація даних

Прогноз та оцінка точності (рис. 2.21)

```
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Точність моделі:", accuracy)
```

Рис. 2.21. Вивід точності моделі

Використовуючи генеративні можливості Random Forest для створення нових даних (рис. 2.22)

```
new_test_scenarios = model.predict(some_generated_data)
```

Рис. 2.22. Створення новиї даних

Перевірка результатів (рис. 2.23)

```
print("Нові тестові сценарії:", new_test_scenarios)
```

Рисунок 2.23 Вивід даних

В цьому прикладі використовується RandomForestClassifier для навчання моделі на історії тестування. Після навчання можна використовувати модель для прогнозування результатів для нових, згенерованих тестових сценаріїв.

4) Інтеграція з інструментами автоматизації тестування з методами штучного інтелекту може бути ефективним способом поліпшення процесу тестування програмного забезпечення [15]. Загальний огляд процесу складається з двої ітерацій:

- Вибір Інструментів
- Визначення Задач та Критеріїв

3) Оптимізація плану тестування [16] - це процес вдосконалення та ефективного використання ресурсів тестування для досягнення максимального покриття, виявлення помилок і вдосконалення якості програмного забезпечення. Це включає в себе краще визначення обсягу тестування, вибір відповідних тестових сценаріїв, оптимізацію використання автоматизації та збільшення ефективності тестового процесу.

Модель оптимізації порядку виконання тестів (Test Execution Order Optimization Model):

$$Time_i = a \cdot X_i + \beta \cdot Y_i$$

Де $Time_i$- час виконання тесту i, X_i - характеристики тесту, Y_i - попередня історія виконання тесту, a та β - параметри моделі.

4) Аналіз історії тестування [17] - це важливий етап у процесі управління якістю програмного забезпечення, який передбачає огляд та аналіз результатів попередніх тестів з метою здобуття важливих висновків та покращення стратегії тестування. Цей аналіз може включати в себе різноманітні

аспекти, такі як ефективність тестування, виявлення помилок, покриття коду, витрати та інші.

Модель виявлення аномалій (Anomaly Detection Model):

$$Score_i = \frac{|X_i - \mu|}{\sigma}$$

Де $Score_i$ - оцінка аномалій для тесту i, X_i - спостережені дані, μ - середнє значення, σ - стандартне відхилення

5) Оптимізація тестового середовища [18] - це процес покращення інфраструктури тестування для забезпечення ефективності, надійності та швидкості виконання тестів. Це включає в себе налаштування апаратних та програмних засобів, створення реалістичних тестових умов, автоматизацію та управління конфігураціями.

Для оптимізації тестового середовища можна розглядати модель, яка передбачає потребу у ресурсах для виконання тестів. Наприклад:

$$ResourcePrediction_i = \beta \cdot X_i + \gamma \cdot Y_i$$

Де:

- $ResourcePrediction_i$ - прогноз потреби у ресурсах для тесту i
- X_i - характеристики програми або тесту, які впливають на використання ресурсів
- Y_i - історія використання ресурсів для тесту i
- β та γ – параметри моделі

Ця формула допомагає визначити, які тести можуть потребувати більше ресурсів та як це може змінитися на основі історії виконання тестів.

6) Швидкісне тестування [19] - це підхід до тестування програмного забезпечення, спрямований на максимальне зменшення часу, необхідного для виконання тестів, зберігаючи при цьому ефективність та відмінність виявлення помилок. Основна ідея полягає в тому, щоб швидко отримати зворотний зв'язок про якість програми, знизити час відгуку та забезпечити швидкий цикл розробки.

Для моделі швидкісного тестування, де основна мета полягає в прискоренні виконання тестів, можна використовувати формулу, яка враховує ефективність виконання тестових сценаріїв. Основною ідеєю є призначення ваги

кожному тесту на основі його очікуваного внеску у покращення ефективності тестування.

$$Speed_i = \frac{1}{1 + e^{-\delta \cdot (X_i - \mu)}}$$

Де:

$Speed_i$ – оцінка швидкісного тестування i

X_i - характеристики тесту, які впливають на швидкість виконання

μ - середнє значення характеристик серед всіх тестів

δ - параметр, який визначає ступінь впливу характеристик на швидкість Ця формула передбачає, що тести з великими значеннями X_i (високі характеристики) матимуть вищий рейтинг швидкості, тобто їх буде вигідніше виконувати швидше. Усереднення характеристик у μ та використання параметра δ дозволяє налаштовувати вплив характеристик на швидкість виконання тестів.

2.4. Порівняння інструментів, які використовують машинне навчання для оптимізації тестування

Для прикладу візьмемо інструменти, які описували [20 – 22]: ***Applitools, Autify, Selenium***

Applitools*:*

- ***Особливості***:

1. Візуальне тестування: Applitools використовує комп'ютерне зору для розпізнавання змін в інтерфейсі.
2. Спеціалізується на перевірці візуального вигляду веб-сайтів та додатків.

- ***Переваги****:*

1. Ефективне візуальне тестування.
2. Підтримка широкого спектру технологій та платформ.

- ***Обмеження***:

1. Орієнтовано переважно на візуальне тестування, менше функцій для тестування функціональності.

Autify:

- ***Особливості***:

1. Автоматизоване тестування без написання коду: Autify дозволяє створювати тести за допомогою візуального інтерфейсу без потреби в програмуванні.
2. Застосовує машинне навчання для стабілізації тестів при змінах в інтерфейсі.

- ***Переваги***:

1. Легка автоматизація для непрограмістів.
2. Стабільність тестів при змінах.

- ***Обмеження***:

1. Може не надавати таку гнучкість, яку потребують досвідчені розробники.

Selenium:

- ***Особливості***:

1. Фреймворк для автоматизації веб-додатків.
2. Потребує написання коду тестів мовою програмування.

- ***Переваги***:

1. Гнучкість та розширюваність завдяки можливості написання коду.
2. Підтримка багатьох мов програмування.

- ***Обмеження***:

1. Вимагає від розробників написання коду для автоматизації тестів.

***Порівняння*:**

Якщо потрібно акцентувати візуальне тестування та розпізнавання змін в інтерфейсі, ***Applitools*** може бути відмінним вибором.

Autify підходить, якщо потрібна легка автоматизація для непрограмістів та стабільність тестів при змінах.

Selenium залишається основним фреймворком для автоматизації, що надає гнучкість та можливості для досвідчених розробників.

2.4. Типи машинного навчання.

Машинне навчання поділяється на кілька основних типів в залежності від способу навчання та завдань, які вони вирішують. Основні типи машинного навчання включають [23]:

Навчання з учителем (Supervised Learning) (рис. 2.24) — це тип МН, де алгоритм отримує вхідні дані разом із правильними відповідями, тобто належно позначеними даними навчання. Задача полягає в тому, щоб створити модель, яка може використовувати знання, отримані на основі вхідних даних і правильних відповідей, щоб передбачати відповіді на нові, раніше невідомі дані.

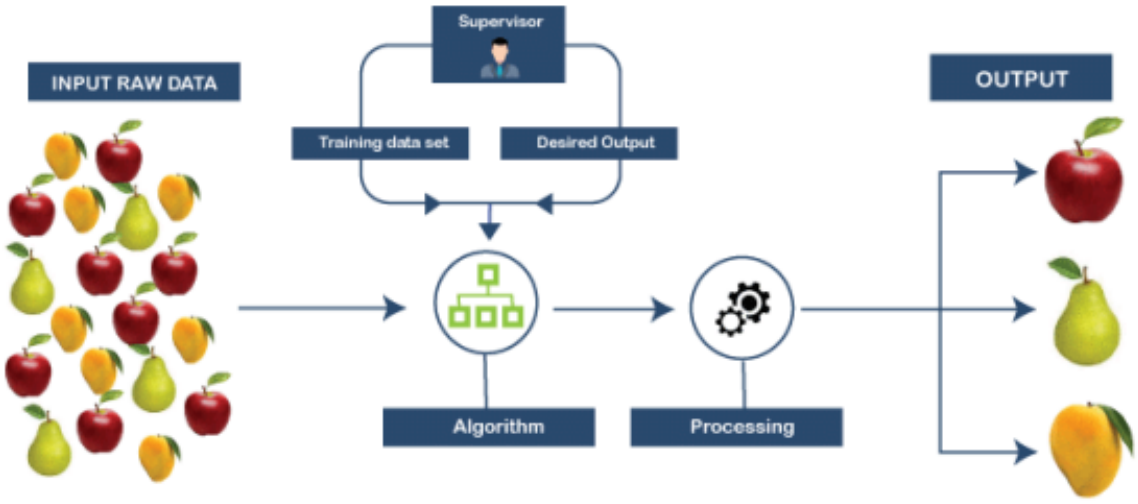


Рис. 2.24. Навчання з учителем

Навчання з учителем створює модель, яка може передбачати відповіді на вхідні дані. Пошук таких параметрів моделі, які мінімізують функцію втрат на вхідних даних навчання, є частиною процесу навчання. Після завершення навчання модель можна використовувати для передбачення відповідей на нові дані. Цей тип поділяється на два підтипа:

- ***Класифікація***: алгоритм повинен розрізняти заданий набір класів як мітку;
- ***Регресія***: метою алгоритму є прогнозування числових значень, відсотків або ймовірностей в якості мітки.

Навчання без учителя (Unsupervised Learning) є одним із типів МН, коли модель вивчає закономірності в наборі даних без попереднього навчання на мітках. В навчанні без нагляду модель отримує тільки вхідні дані, не знаючи, які очікувані результати. Це відрізняється від навчання під наглядом, коли модель отримує набір даних із позначками, пов'язаними з очікуваними вихідними результатами (рис. 2.25).

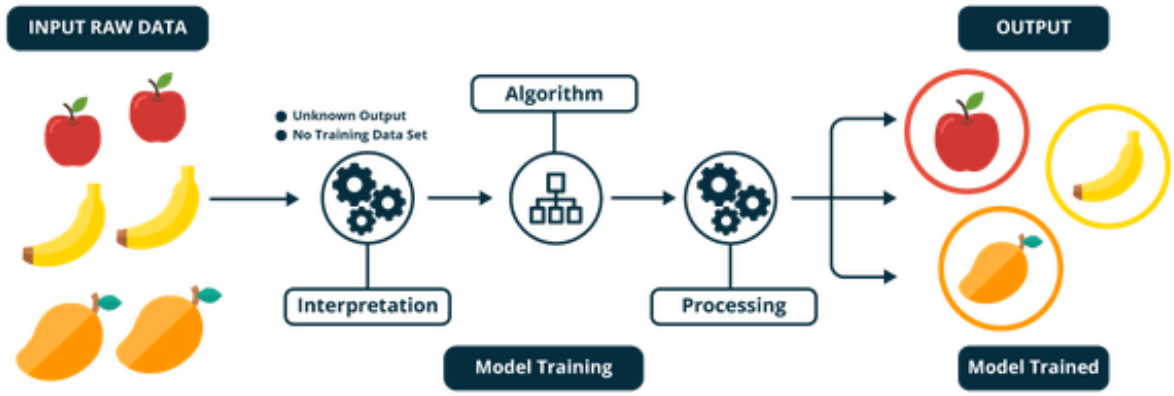


Рис. 2.25 Навчання без учителя

Моделі навчання без учителя використовуються для виявлення складних структур у даних, які можуть бути важко знайти вручну. Наприклад, алгоритми кластеризації можуть розділити дані на групи на основі їх подібності, алгоритми зменшення розмірності можуть зменшити розмірність даних, а асоціативні правила можуть знайти зв'язки між різними елементами даних.

Обробка природних мов, комп'ютерний зір і аналіз даних включають безнаглядне навчання. Цей тип розділяється на наступні категорії:

- Метод кластеризації МН групує схожі об’єкти в один клас або кластер на основі їх схожості та відмінностей від інших кластерів. Аналіз даних, комп’ютерний зір, маркетинг, біологія, інформаційна безпека та інші галузі — це лише деякі з багатьох областей, де кластеризація може бути використана. У кластеризації об’єкти групуються за спільними ознаками або характеристиками, що полегшує спрощення та аналіз великих обсягів даних. Багато методів кластеризації доступні, наприклад, метод к-середніх, ієрархічна кластеризація, DBSCAN та агломеративна кластеризація;
- ***Візуалізація*** — це процес збору даних і отримання нових знань за допомогою візуальних засобів, таких як графіки, діаграми та графіки. Цей тип зберігає оригінальну структуру даних незмінною, призначає окремі кластери для розрізнення перекривлених даних, що допомагає знайти невиявлені закономірності. Як правило, результат представлений у вигляді двовимірних або трьохвимірних графіків.
- Визначення статистично значимих зв’язків між різними елементами в наборі даних називається виявленням асоціативного правила. Асоціативні правила використовуються для визначення залежностей між різними елементами або подіями в даних, такими як покупки в магазині, відвідування веб-сайту або використання програми. Вони можуть бути використані для прийняття рішень щодо маркетингових стратегій, оптимізації процесів або покращення користувацького досвіду, оскільки вони допомагають зрозуміти, які елементи зазвичай супроводжуються іншими.

Apriori є одним із багатьох прикладів алгоритмів виявлення асоціативних правил, які використовуються в різних областях штучного інтелекту та машинного навчання.

Півнавчання або напівконтрольоване навчання (Semi-Supervised Learning) - це тип МН, в якому модель навчається на даних; приклади містять як позначені, так і непозначені мітки класів. Напівконтрольоване навчання використовує значно менше позначених даних, ніж повністю контрольоване навчання, коли кожен навчальний набір має маркування класу.

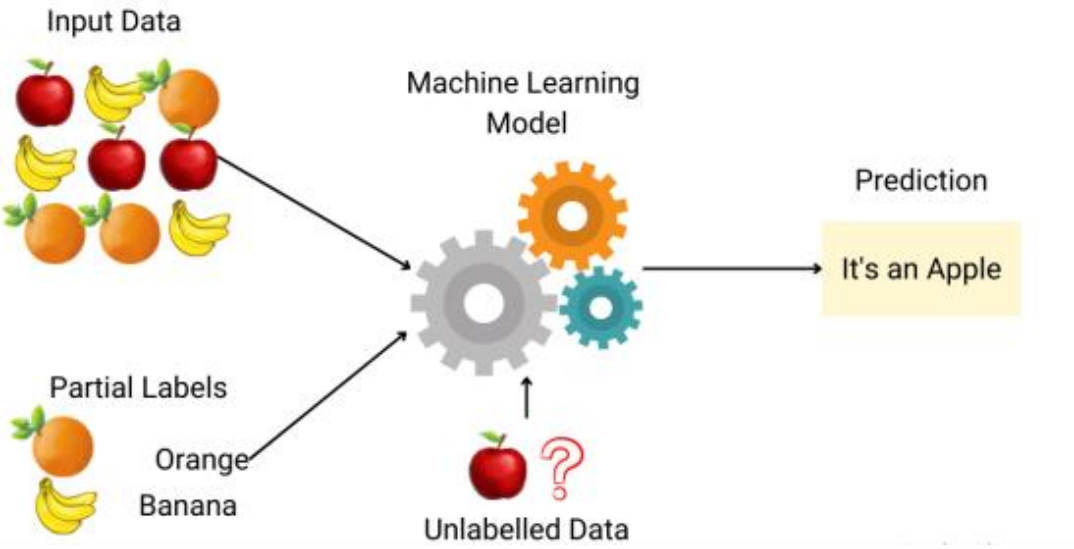


Рис. 2.26. Напівконтрольоване навчання

Напівконтрольоване навчання (рис. 2.26) може бути корисним, коли маємо обмежену кількість позначених даних; однак нам все ще потрібно створити модель, яка може робити передбачення на нових, непозначених даних. Це можливо завдяки використанню непозначених даних для побудови загальної структури даних і зменшення ризику перенавчання на занадто малому наборі позначених даних.

Півнавчання навчання використовується в багатьох сферах, зокрема в комп'ютерному зорі, розпізнаванні голосу та обробці природних мов.

Навчання з підкріпленням (Reinforcement Learning) - це один із підходів до МН, в якому агент навчається приймати рішення на основі результатів своїх попередніх дій у певному середовищі. Агент отримує інформацію про стан навколишнього середовища та можливі дії, які він може зробити. Його мета полягає в тому, щоб максимізувати нагороду (винагороду) за послідовність своїх дій, а також уникнути штрафу. Нагороди зазвичай визначаються заздалегідь і залежать від успішності дій агента (рис. 2.27). Наприклад, коли агент грає в гру, йому нараховується певна кількість балів за правильні рухи, а за неправильні рухи знімається певна кількість балів.

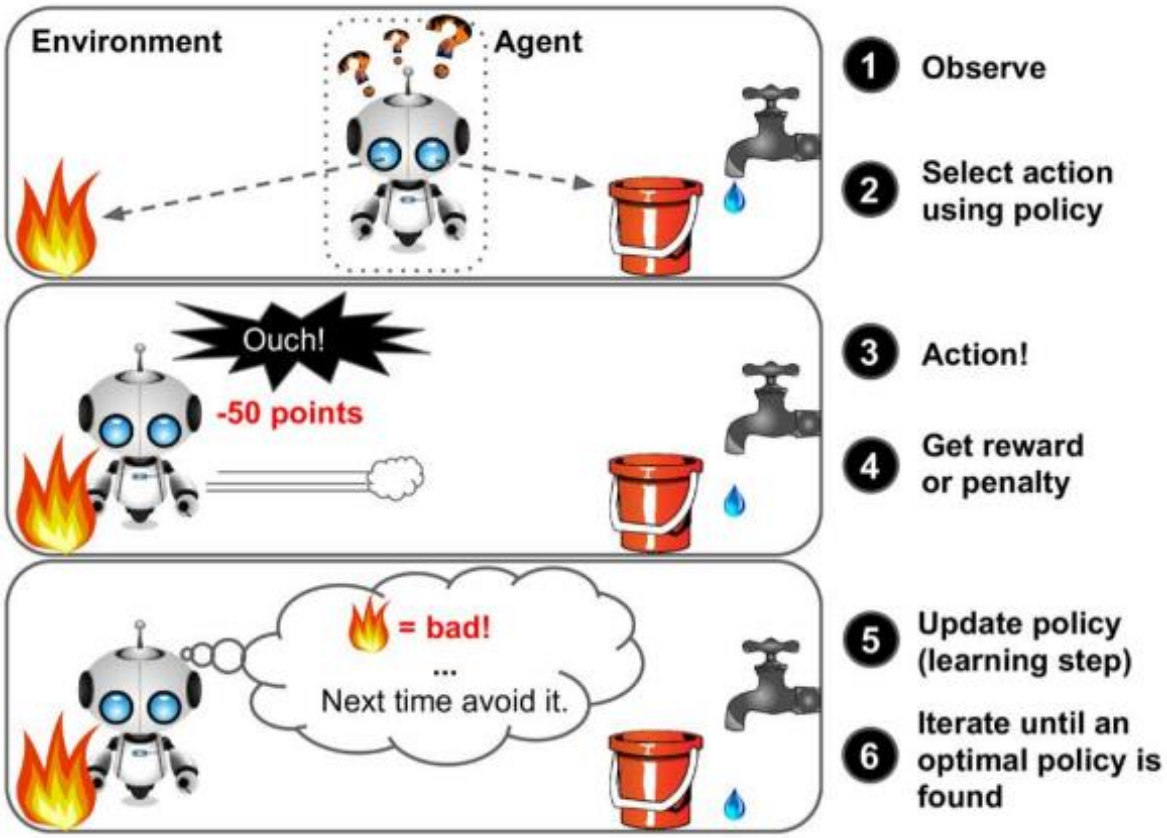


Рис. 2.27 Навчання з підкріпленням

Алгоритм навчання з підкріпленням повинен знайти таку послідовність дій, яка максимізує очікувану майбутню нагороду. Для досягнення цього агент повинен проводити експерименти, взаємодіяти з середовищем і оцінювати результати своїх дій на основі нагород.

Агент, середовище та функція нагороди є ключовими компонентами навчання з підкріпленням. Агент отримує нагороду за кожну взаємодію з середовищем, і на основі свого досвіду він намагається максимізувати нагороду. Функція винагороди визначає, які дії агента є корисними для досягнення мети навчання, а які є некорисними.

2.5. Використання алгоритмів машинного навчання для аналізу та оптимізації тестових сценаріїв.

Використання алгоритмів машинного навчання для аналізу та оптимізації тестових сценаріїв може значно полегшити та покращити процес тестування програмного забезпечення.

Деякі можливі способи, які ви можете впроваджувати:

1. Класифікація тестових сценаріїв:

- Використання алгоритмів класифікації для автоматичного визначення того, який тип тестування (наприклад, функціональне тестування, навантажувальне тестування, інтеграційне тестування) найбільш підходить для конкретного сценарію. Це допоможе оптимізувати використання ресурсів та часу.

2. Призначення пріоритетів тестових сценаріїв:

- Використання алгоритмів для визначення важливості та пріоритетів тестових сценаріїв на основі ризиків, історії дефектів та інших критеріїв. Це може допомогти зосередити увагу на критичних аспектах програми.

3. Автоматизована генерація тестових сценаріїв:

- Застосування алгоритмів машинного навчання для автоматичної генерації тестових сценаріїв на основі вхідних даних. Це може полегшити процес створення та оновлення тестових наборів.

4. Адаптація до змін в програмному коді:

- Використання алгоритмів для виявлення змін в програмному коді та автоматичної адаптації тестових сценаріїв до цих змін. Це дозволяє швидше реагувати на розвиток програмного забезпечення.

5. Оптимізація послідовності виконання тестів:

- Використання алгоритмів для оптимізації порядку виконання тестових сценаріїв з метою максимізації покриття та виявлення дефектів на ранніх етапах тестування.

6. Врахування динаміки додатку:

- Застосування алгоритмів, які можуть адаптувати тестові сценарії до динамічних змін у додатку, таких як нові функції, виправлення помилок, або зміни в інтерфейсі.

Ці підходи дозволяють створити більш гнучкі, ефективні та інтелектуальні тестові сценарії, що допомагає прискорити та поліпшити процес тестування програмного забезпечення.

2.6. Алгоритм машинного навчання для створення ефективних тестових сценаріїв.

Застосування машинного навчання для створення ефективних тестових сценаріїв може виявитися корисним в ряді аспектів тестування програмного забезпечення.

Алгоритм машинного навчання для створення тестових сценаріїв:

1. Збір та підготовка даних:

- Зібрати дані про взаємодію користувачів з додатком під час тестування або реального використання.
- Очистити та обробити ці дані для подальшого використання в алгоритмі.

2. Визначення мети:

- Визначити мету створення тестових сценаріїв, наприклад, покриття ключових функціональностей, оптимізація часу виконання тестів тощо.

3. Вибір моделі машинного навчання:

- Вибрати відповідну модель для завдання. Наприклад, може бути використано класифікаційні або кластерні алгоритми, залежно від характеру завдання.

4. Навчання моделі:

- Використовувати зібрані та підготовлені дані для тренування моделі. Модель повинна навчитися визначати, які шляхи та дії є важливими для досягнення мети.

5. Тестування та валідація моделі:

- Перевірити ефективність моделі на тестовому наборі даних.
- Провести валідацію для переконання, що модель добре узагальнює результати на нових даних.

6. Створення тестових сценаріїв:

- Використовувати навчену модель для автоматичного створення тестових сценаріїв. Модель може визначати послідовність дій, їхню важливість та оптимальний порядок виконання.

7. Адаптація до змін:

- Розробити механізми для автоматичної адаптації тестових сценаріїв до змін в програмному коді чи інтерфейсі додатку.

8. Оптимізація тестових сценаріїв:

- Використовувати модель для оптимізації тестових сценаріїв відповідно до змін в програмному коді та вимог тестування.

2.7. Застосування алгоритму Байєса в аналізі даних

Апріорний розподіл ймовірностей — це розподіл, який використовується для опису попередніх знань про параметри моделі до отримання даних [24].. Воно використовується в Байєсівській статистиці та дозволяє оцінити невідомі параметри моделі на основі даних та попередніх знань.

Завдання апріорного розподілу ймовірностей залежить від типу моделі та параметрів, які потрібно оцінити.

Наприклад, якщо ми оцінюємо середнє значення деякого параметра, можна використовувати нормальний розподіл як апріорний розподіл, де середнє значення та дисперсія вибираються на основі попередніх знань про параметр. Якщо потрібно оцінити частку успіхів у деякому експерименті, можна використовувати бета-розподіл.

Завдання апріорного розподілу вимагає деяких попередніх знань про модель та параметри. Часто такі знання ґрунтуються на експертній думці або на

попередніх дослідженнях. Якщо знань недостатньо, можна використовувати неінформативні розподіли, які мало впливають оцінки параметрів.

Крім того, апріорний розподіл може змінюватися в процесі аналізу даних, наприклад після отримання перших спостережень. У таких випадках можна використовувати так звані оновлені апріорні розподіли, які враховують нові дані.

Байєсівський підхід є методом виявлення та модифікації оцінок ймовірностей на основі даних, причому ці оцінки можуть бути виражені за допомогою розподілу ймовірностей [25]. Суть підходу полягає у оновленні апріорних знань (апріорних ймовірностей) на основі нових даних, які називаються також спостереженнями.

Байєсівській підхід заснований на формулі Байєса, яка дозволяє нам обчислювати апостеріорні ймовірності на основі апріорних ймовірностей та спостережень. Формула Байєса виглядає так:

$$P(A|B) = P(B|A) * \frac{P(A)}{P(B)P(A|B)} = P(B|A) * \frac{P(A)}{P(B)}$$

де $P(A|B)$— апостеріорна ймовірність гіпотези A і $P(A|B)$— ймовірність спостерігати дані B, якщо гіпотеза A вірна. $P(A)$ – апріорна ймовірність гіпотези A та $P(B)$– ймовірність спостерігати дані B у будь-якому випадку.

Таким чином, ми можемо використовувати формулу Байєса, щоб оновити апріорні знання гіпотези на основі нових даних. Якщо ми маємо апріорні знання про гіпотезу, ми можемо використовувати ці дані, щоб оцінити апостеріорну ймовірність гіпотези, коли ми отримаємо додаткові дані.

Імовірність правдоподібності – це міра того, наскільки ймовірно, що дані, які ми спостерігаємо, були згенеровані гіпотезою [26].

Щоб використати ймовірність правдоподібності в алгоритмі Байєса, потрібно визначити ймовірність гіпотез, які ми хочемо перевірити на основі наявних даних. Потім слід визначити ймовірність того, що дані були згенеровані кожною з цих гіпотез.

Для визначення ймовірності правдоподібності використовується функція правдоподібності, яка визначається як ймовірність отримання певного набору

даних за умови, що гіпотеза вірна. Іншими словами, вона вимірює, наскільки ймовірно, що ці дані були згенеровані з використанням цієї гіпотези.

Коли ми визначили ймовірність правдоподібності для кожної гіпотези, ми можемо використовувати теорему Байєса, щоб перерахувати їх ймовірності на основі нових даних, які ми отримуємо. Це дозволяє нам точніше визначати ймовірність того, що кожна гіпотеза вірна.

Мультиноміальна модель Байєса – це статистичний алгоритм класифікації, що ґрунтується на теорії ймовірності [27]. Вона використовується визначення того, якого класу належить новий об'єкт, грунтуючись на ймовірнісної оцінці його характеристик.

Застосовується мультиноміальна модель наївного Байєса, наприклад, завдання фільтрації спам-листів в електронній пошті, визначення тональності текстів або класифікації товарів за категоріями.

Для застосування цієї моделі необхідно:

- Підготувати навчальну вибірку з описом об'єктів та їх ознак. Кожен об'єкт має бути віднесений до одного з класів, на які ми їх класифікуватимемо.
- Оцінити ймовірність появи кожної ознаки для кожного класу на основі навчальної вибірки.
- Використовуючи отримані можливості, класифікувати новий об'єкт.

Наївний клас класифікатора Байєса вважається наївним через припущення про незалежність ознак об'єкта. Незважаючи на те, що це припущення може не виправдовуватися на практиці, метод все одно може дати хороші результати і широко застосовується в аналізі даних.

Переваги і недоліки

Переваги:

- Гнучкість. Алгоритм Байєса підходить для різних класифікаційних завдань, таких як класифікація текстів, аналіз тональності або класифікація зображень.
- Висока точність. Алгоритм Байєса може досягти високої точності класифікації, особливо у завданнях із великою кількістю параметрів.

- Гарна обробка шуму. Алгоритм Байєса може видаляти шуми даних, тим самим покращуючи продуктивність.

Недоліки:

- Необхідність припущення незалежності. Алгоритм Байєса працює краще, коли всі ознаки незалежні одна від одної.
- Витратність обчислень. Алгоритм Байєса може бути витратним у обчислювальному плані, особливо коли є велика кількість параметрів.
- Обмеження у застосуванні. Алгоритм Байєса може неправильно працювати, якщо даних дуже мало або ознаки сильно корелюванні.

РОЗДІЛ 3.
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДУ

3.1. Практичне забезпечення засобів збирання та оброблення даних з використанням машинного навчання

Практичне забезпечення засобів збирання та оброблення даних з використанням машинного навчання включає в себе використання різноманітних практични методів і алгоритмів для ефективної обробки та аналізу великих обсягів інформації.

Практичний метод використання математичних методів у машинному навчанні на прикладі класифікації електронних листів на спам та неспам (рис. 3.1).

```python
# Імпортуємо необхідні бібліотеки
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Завантажуємо дані
data = pd.read_csv('spam_dataset.csv')  # Припустимо, у вас є CSV-файл з даними

# Розділяємо дані на тренувальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.2, random_state=42)

# Використовуємо CountVectorizer для векторизації тексту
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Тренуємо модель на основі наївного баєсівського класифікатора
classifier = MultinomialNB()
classifier.fit(X_train_vectorized, y_train)

# Прогнозуємо класи для тестового набору
predictions = classifier.predict(X_test_vectorized)

# Оцінюємо точність та інші метрики
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

# Виводимо результати
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(report)
```

Рис. 3.1. Математичні методи у машинному навчанні

Спочатку іде імпорт необхідних бібліотек, які є необхідним етапом для використання функцій та класів (рис. 3.2)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

Рис. 3.2. Імпорт бібліотек на мові програмування Python

В даному випадку, ми використовуємо бібліотеку **“*pandas*”** для роботи з даними у форматі DataFrame.

- ***train_test_split***: Дозволяє розділити дані на тренувальний та тестовий набори.
- ***CountVectorizer***: Векторизація тексту, перетворення тексту в матрицю числових ознак.
- ***MultinomialNB***: Модель наївного Баєсівського класифікатора для категоріальних ознак.
- ***accuracy_score ma classification_report***: Метрики для оцінки ефективності моделі.

Далі завантажуємо дані з CSV-файлу в DataFrame (рис. 3.3)

```
data = pd.read_csv('spam_dataset.csv')
```

Рису. 3.3. Завантаження даних

Наступним кроком розділяємо дані на тренувальний та тестовий набори (рис. 3.4)

```
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['label'], test_size=0.2, random_state=42)
```

Рис. 3.4. Розділення даних на тренувальний та тестовий набори

В даному випадку ***“train_test_split”*** розділяє дані у випадковому порядку, вибираючи 80% для тренування і 20% для тестування.

“random_state” гарантує, що розділення буде випадковим, але при цьому при кожному запуску програми буде вивід однакового розділу.

Далі при використанні **“*CountVectorizer*”** провести векторизацію тексту (рис. 3.5) для перетворення текстового корпусу в матрицю числових ознак, яка може бути використана для навчання моделей машинного навчання.

```
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
```

Рис. 3.5. Векторизація текстових даних

На (рис 3.5.) при використанні **“*CountVectorizer*”** виконується векторизація тексту та тренування моделі наївного Байєсівського класифікатора для розпізнавання спаму та неспаму.

“fit_transform” використовується для тренування векторизатора на тренувальних даних і одночасної трансформації тренувальних даних у вектори.

“*transform*” застосовується до тестових даних для їхньої трансформації у вектори.

Для тренування моделі використовуємо класифікатор наївного Байєсівського класифікатора (рис. 3.6)

```
classifier = MultinomialNB()
classifier.fit(X_train_vectorized, y_train)
```

Рис. 3.6. Тренування моделі наївним Байєсівським класифікатором

Використовується ***“MultinomialNB”*** для тренування наївного Бйаєсівського класифікатора на векторизованих тренувальних даних.

Модель використовується для прогнозування класів на тестовому наборі, а потім оцінюються точність (рис. 3.7) та інші метрики за допомогою ***“accuracy_score”*** та ***“classification_report”.***

```
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)
```

Рис. 3.7. Оцінка точності та інші метрики

Де:

- ***“accuracy_score”*** Обчислює точність моделі.
- ***“classification_report”*** генерує звіт про класифікацію, який містить основні метрики.

Та вивід результату за допомогою “print” (рис. 2.8)

```
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(report)
```

Рис. 3.8. Вивід результату

3.1. Математичне забезпечення засобів збирання та оброблення даних з використанням машинного навчання

Розглянемо простий математичний приклад з використанням машинного навчання. Нехай ми маємо набір даних (рис. 3.9), що представляє собою залежність між кількістю годин вивчення (x) та оцінкою студента (y).

Години Вивчення (x)	Оцінка (y)
2	60
3	70
4	80
5	85
6	90

Рис. 3.9. Набір даних

Побудуємо модель машинного навчання, яка буде передбачати оцінку студента на основі кількості годин вивчення. Для цього використаємо простий лінійний регресійний підхід.

Модель лінійної регресії має вигляд:

$$y = mx + b$$

Де:

y – оцінка студента,

x – кількість годин вивчення,

m – коефіцієнт нахилу(вага),

b – відсув (зсув або вільний член).

Мета полягає в знаходженні значень m та b, які найкраще відображають залежність між годинами вивчення та оцінкою.

Використовуючи методи машинного навчання для тренування моделі на цьому наборі даних та знаходження оптимальних значень параметрів. Наприклад, можемо використовувати бібліотеку ***“Scikit-learn”*** в ***“Python”***:

```python
from sklearn.linear_model import LinearRegression
import numpy as np
```

Рис. 3.10. Імпорт бібліотеки **“*Scikit-learn*”**

```
hours_studied = np.array([2, 3, 4, 5, 6]).reshape(-1, 1)
grades = np.array([60, 70, 80, 85, 90])
```

Рис. 3.11. Ввід вхідних даних

```
model = LinearRegression()
model.fit(hours_studied, grades)
```

Рис. 3.12. Створення та тренування моделі

```
new_hours = np.array([7, 8]).reshape(-1, 1)
predicted_grades = model.predict(new_hours)
```

Рис. 3.13. Передбачення оцінок для нових годин вивчення

```
print("Коефіцієнт нахилу (вага):", model.coef_[0])
print("Вільний член (відсув):", model.intercept_)
print("Предбачені оцінки для нових годин вивчення:", predicted_grades)
```

Рис. 3.14. Вивід результатів

Цей приклад (рис. 3.10 – рис. 3.14) показує базовий підхід до використання машинного навчання для побудови моделі, яка може передбачати оцінки на основі вхідних годин вивчення.

Розглянемо другий приклад машинного навчання на основі розпізнавання зображення. Маємо завдання класифікації, де ми хочемо навчити модель розпізнавати, чи є зображення цифри "5" на основі пікселів цього зображення.

Нехай у нас є простий набір даних (рис. 3.15), що представляє собою чорно-білі зображення розміром 3x3 пікселі:

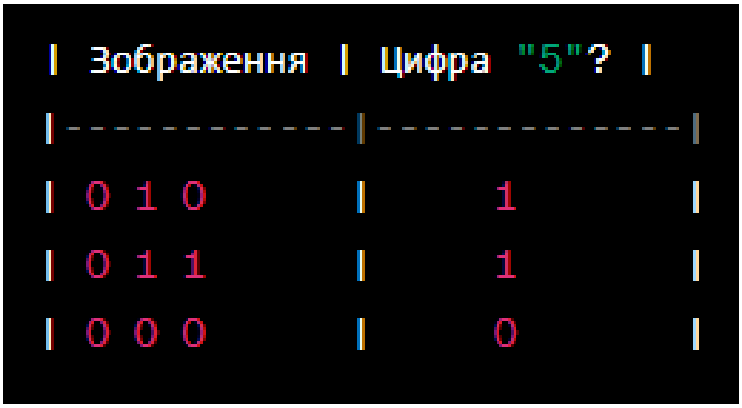


Рис. 3.15. Набір даних

У цьому випадку:

- Кожен рядок представляє рядок пікселів на зображенні.
- Кожен стовпчик представляє значення пікселя для відповідного рядка.
- "Цифра '5'?" - це мітка класу, де 1 вказує на те, що на зображенні зображена цифра "5", а 0 - що ні.

Наша задача - навчити модель класифікації на основі цих даних. Давайте використаємо простий лінійний класифікатор, наприклад, логістичну регресію.

Модель логістичної регресії може бути визначена як:

$$y = \sigma(wx + b)$$

де:

y – вихід моделі,

σ – функція активації сигмоїда

w – коефіцієнт моделі

x – значення пікселів

b- вільний член

Функція активації сигмоїда:

$$\sigma(\mathcal{Z}) = \frac{1}{1+e^{-\mathcal{Z}}},$$

де

$$\mathcal{Z} = wx + b.$$

Ми можемо тренувати цю модель за допомогою методу градієнтного спуску, мінімізуючи витрати (наприклад, перехресна ентропія) між прогнозованими та справжніми мітками.

Тренування моделі за допомогою градієнтного спуску включає кілька кроків. Розглянемо цей процес для задачі логістичної регресії та мінімізації перехресної ентропії.

Позначимо:

y – справжній клас (0 або 1),

$\hat{y}$ – прогнозований клас (вихід моделі),

X – матриця ознак,

w – ваги моделі,

b – вільний член.

1. Ініціалізація параметрів

- Ініціалізуємо ваги w та вільний член b.

2. Обчислення виходу моделі

- Визначаємо ваговану суму: $z = wx + b$
- Застосовуємо функцію активації сигмоїди: $\hat{y} = \sigma(\mathcal{Z}) = \frac{1}{1+e^{-\mathcal{Z}}}$

3. Обчислення функції витрат

- Використовуємо функцію витрат, таку як перехресна ентропія, для оцінки різниці між прогнозованими та справжніми мітками

4. Обчислення градієнту

Обчислюємо градієнт функції витрат щодо параметрів w та b

5. Оновлення параметрів

- Застосовуємо метод градієнтного спуску для оновлення параметрів:

$$w = w - a\frac{\varrho J}{\varrho w},$$

$$b = b - a\frac{\varrho J}{\varrho b},$$

де a – крок навчання

6. Повторення

- Повторюємо кроки 2-5 для заданої кількості ітерацій або до збіжності.

Цей процес триває, поки функція витрат зменшується, і модель здатна давати точні прогнози для невідомих даних. Метод градієнтного спуску дозволяє оптимізувати параметри моделі, змінюючи їх у напрямку, що призводить до мінімізації витрат.

3.2. Архітектура розробленого програмного забезпечення

Програмна система була реалізована у формі мікросервісів. Мікросервіс має більше переваг, ніж монолітна архітектура.

Мікросервіси – це форма сервіс-орієнтованої архітектури (SOA), яка використовується для створення розподілених програмних систем [28]. Мікросервісна архітектура дозволяє модулям працювати разом у мережі з однією метою. Мікросервіси поступово перевершують монолітні програми та стають стандартом розвитку програмних систем.

Важливо пам’ятати, що «сервіс» — це сукупність послуг і функціональності для користувача. Мікросервіси – це поділ функцій, які можна використовувати в інших частинах системи.

Кожен мікросервіс виконує небагато функцій через дрібний поділ функціональності. Мікросервісна архітектура складається з кількох функціональних модулів, які працюють разом через мережу.

Мікросервіси відрізняються від моноліту тим, що вони використовують спеціалізовані простіші програми (модулі) для виконання сценарію програми.

Щоб зробити процес більш простим, модулі можуть розташовуватися на різних серверах і взаємодіяти один з одним через мережу за допомогою протоколонезалежної технології.

Микросервісна архітектура має багато переваг:

- мікросервіси мають симетричну архітектуру (в монолітних додатках вона є ієрархічною);
- взаємозамінність мікросервісів;
- незалежність один від одного;

- організація модулів навколо окремих функцій;
- написання мікросервісів за допомогою будь-яких програмних засобів, оптимізованих для кожного модуля, при цьому вони добре «розуміються» один з одним завдяки інтерфейсу;
- Мікросервіси викликаються тільки користувачем, але не один
- одним

В результаті була обрана мікросервісна архітектура, яка представлена на (рис. 3.16).

Вся система складається з двох частин: клієнтської та серверної. До клієнтської частини входять такі сервіси, як реєстратор, агент і клієнт.

Це клієнтська версія API. HTTP-клієнти, які обробляють запити HTTP.

Агент є інтеграцією фреймфорку. Спеціальні репортери/слухачі, які спостерігають за тестовими подіями та викликають їх надсилання через клієнта

Логер — це інтеграція логів, яка допомагає збирати журнали, пов’язує його з тестовим кодом через агент і надсилає його через клієнта на сервер [29].

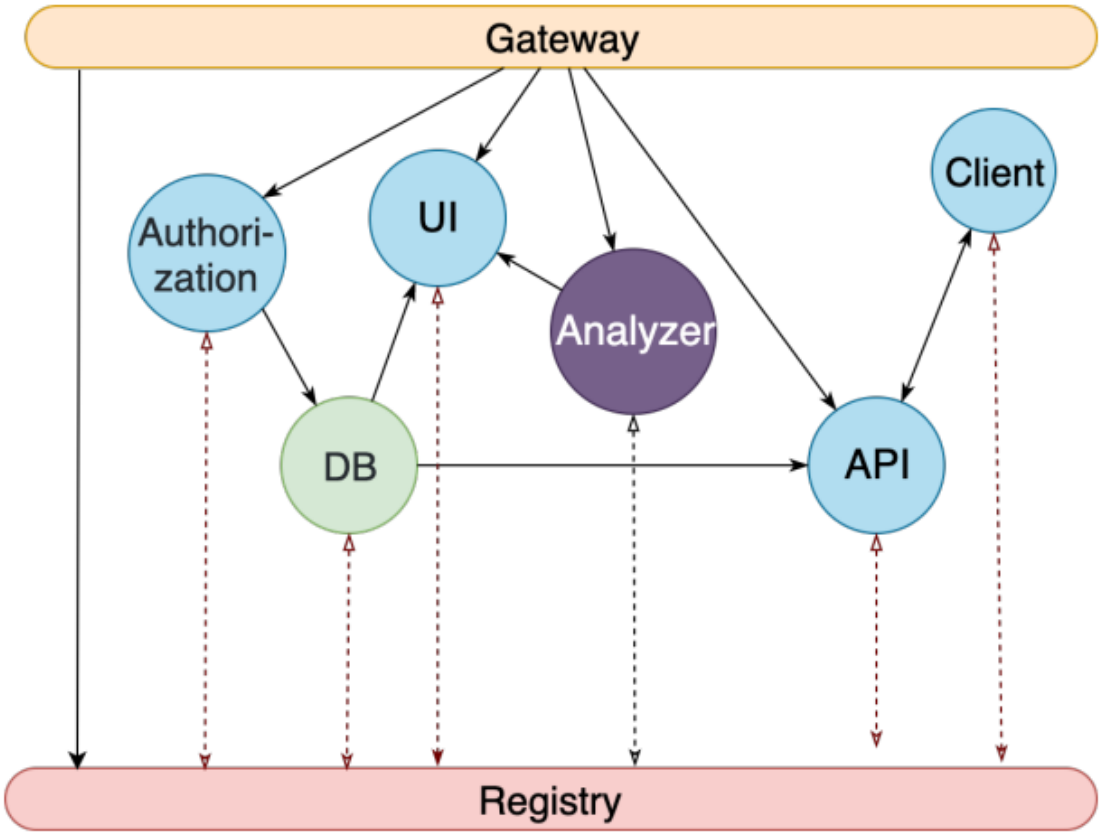


Рисунок 3.16. Архітектура програмного забезпечення

Gateway служить основним входом до служб прикладних програм.

Цей сервіс відповідає за маршрутизацію запитів для отримання належного обслуговування та балансування навантаження. Шлюз зв'язується з реєстром послуг, щоб дізнатися, які служби дійсно мають дозвіл маршрутизувати трафік.

Registry — це інструмент, який зберігає реальний список служб, які були запущені, разом із доданою метаінформацією. Він перевіряє доступність кожної запущеної служби.

API є основою системи. Він відповідає як за обробку запитів агентів, так і за інтерфейс користувача.

Модуль авторизації дозволяє користувачам створювати та відкликати токени. Він підтримує кілька систем аутентифікації, такі як:

- ***Basic Auth;***
- ***GitHub Auth (OAuth2)***
- ***LDAP Auth***. вказує на аутентифікацію за допомогою протоколу LDAP (Lightweight Directory Access Protocol), який є протоколом доступу до каталогів, що використовується для управління та доступу до розподілених каталогів інформації, таких як іменники або каталоги користувачів.
- UI (токен, що закінчується)
- API – токен, що не закінчується, призначений для використання на стороні агента

Analyzer — це інструмент, який зберігає індекс журналів користувачів проекту та надає можливість проводити пошук відповідно до цього індексу. використовується функція автоматизації.

Універсальний інтерфейс (UI) — це сервіс, який відповідає за клієнтську частину системи.

База даних(DB) — це сервіс, який відповідає за підтримку бази даних системи.

3.3. Аналіз баз даних, що відповідають визначеним вимогам

Cassandra — це сховище, засноване на структурі ключ-значення, високомасштабоване, узгоджене (автоматично відновлює узгодженість даних) і розподілене. Модель даних (рис. 3.17) Google BigTable і технології розподілених систем Dynamo доступні в Cassandra [30].

Обробка великої кількості неструктурованих даних є однією з найсильніших характеристик цієї бази даних. Коли база даних потребує швидкого масштабування з мінімальним адміністративним навантаженням, Cassandra може бути хорошим варіантом.

Cassandra може обробляти завантаження програм, таких як Instagram, які щодня завантажують приблизно 80 мільйонів фотографій у базу даних.

Використовуючи рядки та стовпці, Cassandra дозволяє змінювати їхні назви та формат. Вона використовує ключові та табличні значення. Таблиці можна створювати, змінювати та скидати під час обробки запитів бази даних. Це відрізняє стандартну систему керування реляційною базою даних (РСУБД).

Сімейство стовпців, як і таблиця в РСУБД, складається з рядків і стовпців, але кожен рядок має свій власний ключ. На відміну від стандартних СУБД, у кожному рядку таблиці не повинні бути однакові стовпці. Крім того, ці стовпці можна додати «на льоту». Доступ до них можна отримати за допомогою мови запиту Cassandra (CQL). Хоча CQL має багато спільного з синтаксисом SQL, Cassandra не є реляційною базою даних, тому вона має різні методи отримання та зберігання даних.

Cassandra дозволяє копіювати дані з «коробки». Вам потрібно лише вказати кількість вузлів, у які вона повинна скопіювати дані, а потім вона завершить решту процесу.

***Переваги СУБД Cassandra*:**

- Висока масштабованість і надійність, а також відсутність елементів, відмова від яких може призвести до виходу з ладу всієї системи.
- Реалізація сімейства колонок NoSQL.

- Висока швидкість запису та зчитування.
- мова запитів, схожа на SQL (починаючи з версії 0.8), і підтримка пошуку за допомогою вторинних індексів.
- Підтримка копій.
- Гнучка структура

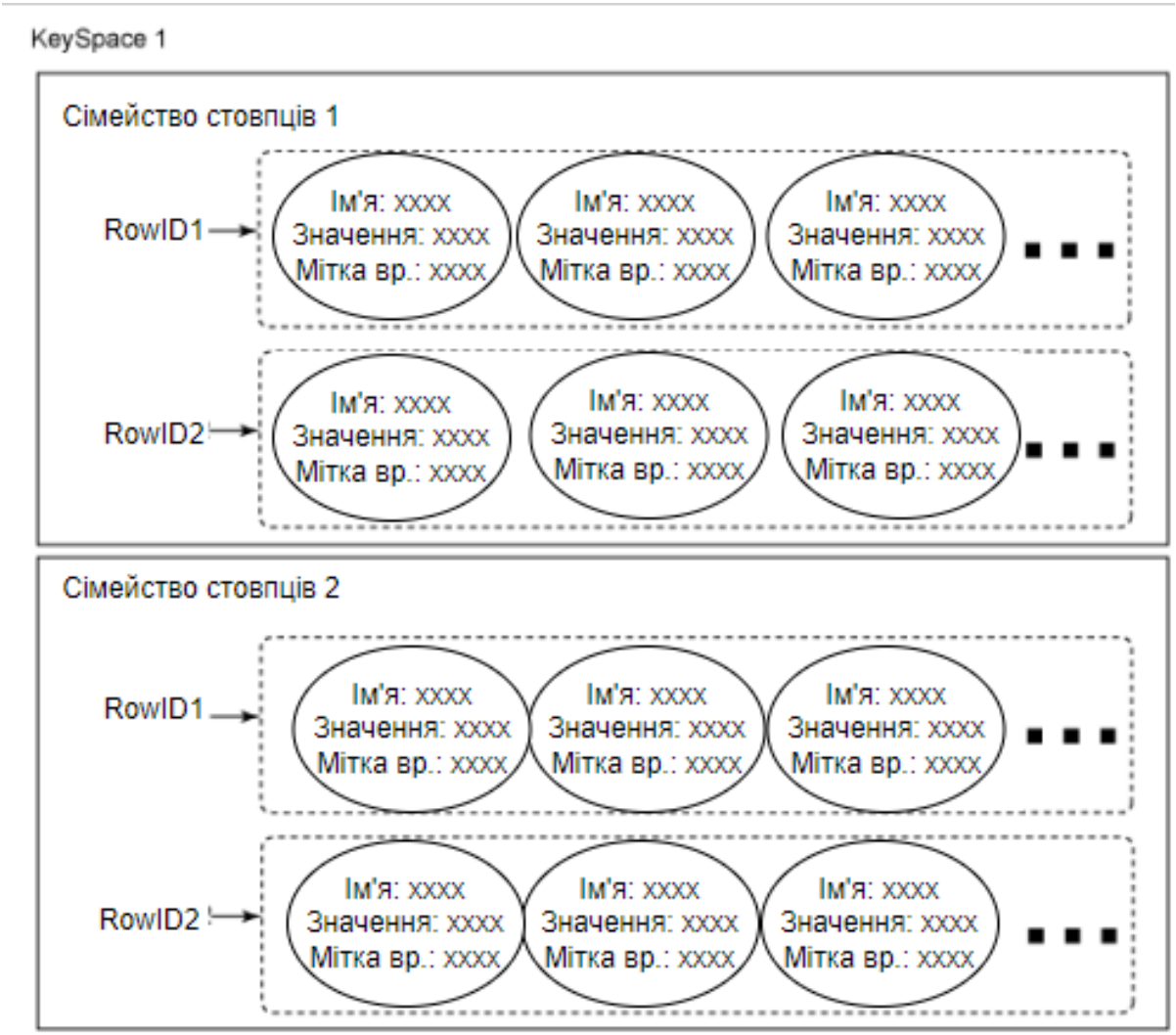


Рис. 3.17. Модель даних СУБД Cassandra

***Недоліки СУБД Cassandra*:**

- Погано реалізовано пошук по діапазону.
- Дуже багато налаштувань винесені на рівень кластеру.

MongoDB має документо-орієнтовану модель даних (рис. 3.18), що робить його швидшим, більш масштабованим і простішим у використанні, на відміну від реляційних баз даних.

Рис. 3.18. Модель даних MongoDB

JSON (JavaScript Object Notation) є одним із найпоширеніших стандартів для обміну та зберігання даними. JSON добре підходить для опису складних даних. Хоча цей метод зберігання даних не використовує JSON, він досить схожий на метод зберігання в MongoDB. Формат зберігання даних MongoDB називається BSON, що є скороченням від binary JSON.

Мова запитів MongoDB дозволяє отримати доступ до збережених даних. Оскільки він не має схем, документи можна створювати, не задаваючи їм початкової структури, як це потрібно для реляційних баз даних.

Переваги MongoDB:

- Гнучкість схеми. MongoDB використовує гнучку схему даних, що означає, що документи в колекції не обов'язково повинні мати однакову структуру. Це дозволяє легко змінювати структуру даних без необхідності модифікації всіх записів.

- Швидкодія: використовує формат даних BSON (бінарний JSON), що сприяє ефективному зберіганню та обробці даних. Відсутність складних JOIN-операцій також сприяє високій продуктивності.
- Масштабованість: легко масштабується горизонтально, дозволяючи додавати нові сервери для обробки більше обсягів даних та забезпечення високої доступності.
- Вбудовані можливості шардування: підтримує вбудовані можливості шардування, що дозволяє розподілити дані між різними серверами для оптимізації продуктивності та забезпечення масштабованості.
- Вбудовані можливості реплікації: надає можливості реплікації для забезпечення високої доступності та відновлення в разі відмови.
- Можливості текстового пошуку: має вбудовану підтримку повнотекстового пошуку, що дозволяє виконувати потужні операції пошуку в текстових полях

Недоліки MongoDB:

- Велика кількість пам'яті: може вимагати значної кількості оперативної пам'яті для ефективної роботи, особливо при великих обсягах даних.
- Високі вимоги до дискового простору: надає можливості стиснення даних, вона може вимагати значного дискового простору, особливо при використанні резервного копіювання та реплікації.
- Відсутність транзакцій у стилі реляційних баз даних: не підтримує транзакцій у стилі реляційних баз даних, що може створити складнощі при роботі з великими обсягами даних та потребою в атомарності операцій.
- Менш ефективна для складних JOIN-операцій: призначена для простих операцій зчитування та запису, і вона не є найкращим вибором для складних JOIN-операцій, які характерні для реляційних баз даних.
- Велика кількість дискового вводу-виводу: може генерувати значну кількість дискового вводу-виводу, що може впливати на продуктивність в тих випадках, коли дискові системи менш продуктивні.

CouchDB використовує розроблене сховище даних у форматі JSON. Ви можете читати, додавати, редагувати та видаляти документи з бази даних за допомогою RESTful HTTP API. Кожен документ містить кілька полів і вкладень. Логічні значення, списки, текст, цифри тощо можуть складати поля.

Перегляди CouchDB схожі на індекси SQL. Їх можна використовувати для сортування документів, збору даних у певному порядку та створення корисних індексів для пошуку документів у їх межах. Індекси дозволяють відображати зв'язки між документами. Цей перегляд результатів зберігається в структурі індексу B-tree.

MapReduce — це двоетапний процес, який використовується CouchDB, щоб переглядати всі документи та створити масив значень, який складається з упорядкованого списку пар ключів/значення. Відображення відбувається лише після того, як документ створений. Після цього він не змінюється до моменту оновлення документа.

CouchDB доступний за допомогою Erlang і працює з Android, BSD, iOS, Linux, OS X, Solaris і Windows.

Багато мов програмування підтримуються, включаючи Haskell, Java, C, C#, ColdFusion, Erlang, Objective-C, OCaml, Perl, PHP, PL/SQL, Python, JavaScript, Lisp, Lua, Ruby і Smalltalk.

OuchDB підтримує реплікацію master-master і master-slave. Це гарантує низькі затримки доступу до даних незалежно від їх місцезнаходження. Реплікація CouchDB настільки ж проста, як надсилання запитів HTTP до бази даних.

CouchDB передає в цільову базу даних будь-які зміни в джерелі. Це однонаправлений процес. Якщо потрібен процес, який працює в двох напрямках, реплікацію потрібно почати на цільовому сервері, коли він є джерелом, а віддалений сервер є призначенням.

Обрання бази даних. На попередньому етапі був проведений аналіз можливостей кожної з баз даних, що найбільш відповідають зазначеним до них вимогам. Отримані результати аналізу наведено в табл. 3.1.

У якості СУБД для зберігання даних було прийняте рішення обрати

MongoDB, так як вона більше підходить для часткової зміни даних, має вбудований повнотекстовій пошук

Таблиця 3.1.

Порівняння баз даних

СУБД	Переваги СУБД	Недоліки СУБД
Cassandra	Висока масштабованість, SQL-подібна мова запитів	Погано реалізовано пошук по діапазону, дуже багато налаштувань винесені на рівень кластеру.
MongoDB	Балансування навантаження, можливість використовувати в якості файлового сховища, підтримка асинхронної реплікації	Проблеми з узгодженістю, блокування на рівні документу
CouchDB	REST запити з "коробки"	Не призначений для часто оновлюваних даних, не має вбудованого повнотекстового пошуку

3.4. Вибір мови програмування та необхідних бібліотек

Python [31] було обрано як мову програмування, яка забезпечує доступ до бази даних і необхідний математичний апарат. Python має наступні переваги, що стосуються вирішення поставленої задачі:

- Для роботи з графовими та реляційними СКБД дана мова програмування пропонує прості та легко зрозумілі драйвери, що дозволяє легко

інтегрувати розроблений модуль до вже існуючих ПП, поєднуючи використання СКБД різних типів за необхідності

- наявна велика кількість бібліотек, які вже готові реалізувати велику кількість алгоритмів, нейронних мереж та інших методів машинного навчання, а також бібліотека ***TensorFlow***
- наявна велика кількість бібліотек для роботи з даними, їх представлення та візуалізації.
- наявна велика кількість бібліотек з вже готовою реалізацією великої кількості алгоритмів, нейронних мереж та інших методів машинного навчання;

Основним недоліком python є низька швидкодія. Однак у поставленій задачі не потрібно обробляти надвеликі обсяги даних, тому швидкодії достатньо.

Pi2neo було обрано в якості драйвера для роботи з Neo4j, оскільки він простий у використанні, має детальну документацію, яка регулярно оновлюється зі змінами версії та поширений у використанні, що дозволяє легко знайти відповіді на запитання, що виникають під час використання. для роботи з даними та розробки алгоритмів:

- ***Scikit-Learn*** — це зручна бібліотека, яка містить велику кількість методів машинного навчання, які вже використовуються. Бібліотека містить вже готову реалізацію модифікованого ***k-means++***
- NumPy містить необхідний математичний апарат для роботи з даними, складних обчислень і роботи з багатовимірними масивами
- Pandas забезпечує зручне представлення даних і спрощення їх оброблення та аналізу
- Matplotlib забезпечує зручне представлення даних

РОЗДІЛ 4
ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1. Опис програмного забезпечення

Створення програмного забезпечення на основі машинного навчання для тестування програмного забезпечення – це достатньо об’ємна робота, яка вміщує в себе велику кількість складових:

- Генерація тестових сценаріїв
- Виявлення автоматичних тестових наборів
- Автоматизована ідентифікація тестових випадків
- Покращення стійкості тестових сценаріїв
- Виявлення та прогнозування дефектів
- Оптимізація тестових даних
- Моніторинг результатів тестування
- Автоматичне виявлення та оновлення локаторів об'єктів (для UI-тестування)

Для кожного з цих напрямів потрібно визначитись з програмним забезпеченням. Для створення даного програмного продукту буде використовуватись різне програмне забезпечення до кожної складової, та за допомогою помп’ютерних технологій.

4.2. Програмне забезпечення для створення тестових сценаріїв

Автоматизація тестування програмного забезпечення на основі машинного навчання ***(Machine Learning-Based Test Automation)*** - це використання методів та технологій машинного навчання для покращення та оптимізації процесу автоматизованого тестування. Це включає в себе використання алгоритмів та моделей машинного навчання для автоматизації різних аспектів тестування, таких як генерація тестових сценаріїв, вибір тестових наборів, виявлення дефектів, оптимізація тестових даних та інше.

Для створення тестових сценаріїв використаємо фреймворк “Cucumber”, та його застосування для створення UI-автотестів. Ця програма популярна не тільки серед тестувальників високого рівня, а й тих, хто тільки починає своє знайомство зі сферою автоматизованого тестування програмного забезпечення.

Gherkin являє собою структуровану манеру написання документації для PO, бізнес-аналітиків та тестувальників. «Академічне» визначення Gherkin говорить, що це людино-читана мова для опису поведінки системи, кожен рядок починається з одного з ключових слів ***(Given-When-Then-And)*** і описує одну з передумов/кроків/результатів.

Cucumber - це інструмент, який використовується для виконання behavior driven (BDD) сценаріїв, написаних мовою Gherkin.

BDD (behavior driven development) - методологія розробки ПЗ, основною ідеєю якої є складання вимог до продукту у форматі понятих не-фахівцю поведінкових сценаріїв. Приклад BDD сценарію(рис. 4.1):

```
Scenario: Login with PIN
Given the app is running
And I'am registered user
And I see Login screen
When I enter 4-digits PIN
Then I am logged in
```

Рис. 4.1. Приклад використання behavior driven development

Поєднавши ці три поняття, ми отримуємо методологію, в якій всі вимоги до нової фічі описуються простими, зрозумілими для будь-якої людини сценаріями, що покривають усі можливі шляхи поведінки користувача. Ці сценарії одночасно є тест кейсами.

Як встановити Cucumber.

Через те, що дана програма є semi-official, то процесс встановлення та налаштування даного фреймворка відбувається вручну.

Cucumber - це бібліотека CocoaPod, тому починаємо з неї. Відкриваємо термінал та встановлюємо CocoaPod (рис. 4.2)

```
sudo gem install cocoa pods
```

Рис. 4.2. Встановлення бібліотеки CocoaPod

Переходимо у свій проект і створюємо підфайл ***“pod init”***

Заходимо у створений підфайл. Перевіряємо, що в ньому є таке

```
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '11.0'

use_frameworks!
inhibit_all_warnings!

def test_pods
  pod 'Cucumberish'
end

target 'НАЗВАНИЕ_ВАШЕГО_ПРОЕКТАCucumberTests' do
  test_pods
end
```

Рис. 4.3. Дані про встановлення CocoaPod]

Для того, щоб створити свій перший тест

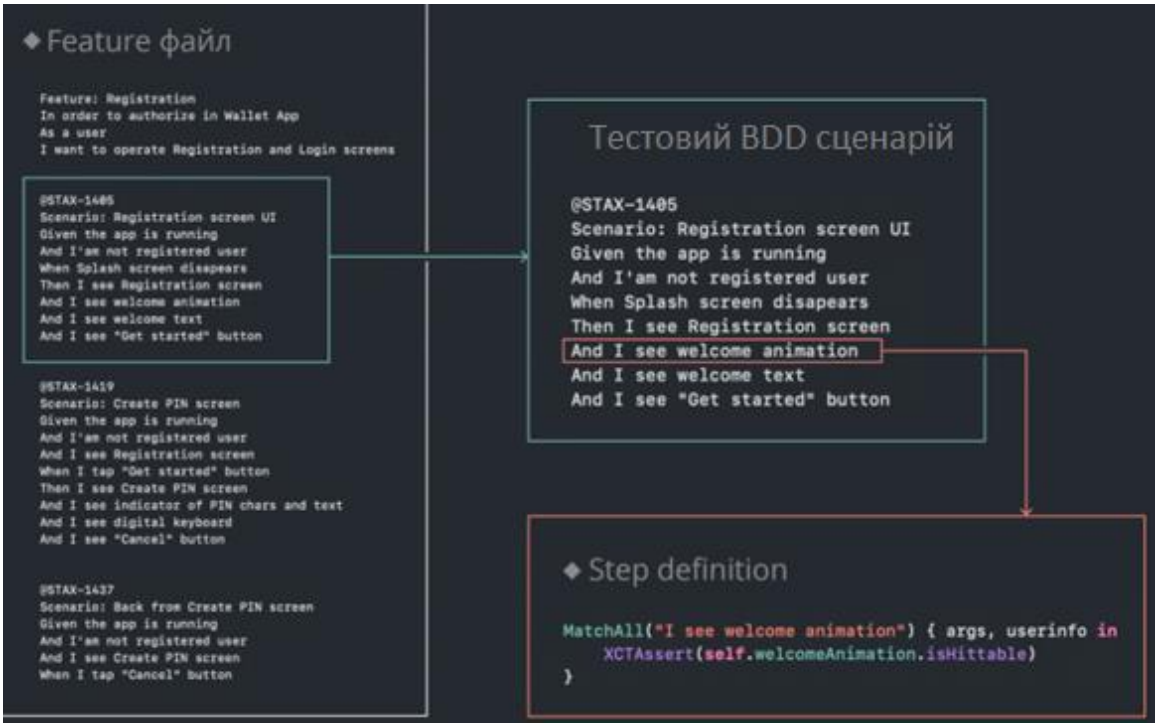


Рис. 4.4. Створення проекту в програмного забезпечення Cucumber

На цьому етапі(4.4):

- фреймворк збере проект на вибраному девайсі/емуляторі
- фреймворк построчно розпарить перший сценарій з першого фіча-файлу
- всі перші Given та And рядки будуть розглядатися як передумови тесту. Вони можуть бути як зовсім простими і складатися з єдиної Given умови, так і комплексними Given-And-And-... наборами рядків

4.3. Виявлення автоматичних тестових наборів

За допомогою рішення Preflight для тестування з низьким вмістом коду будь-хто в організації, незалежно від своїх технічних можливостей, може створювати, запускати та підтримувати автоматизовані тести з винятковою швидкістю. Оскільки немає потреби писати код, команди можуть створювати складні сценарії наскрізного тестування без необхідності вкладати дорогі інженерні ресурси. Це означає, що більшу частину програми можна належним

чином протестувати без уповільнення інвестицій або швидкості випуску нових функцій і продуктів[6].

Applitools пропонує галузеву платформу автоматизації тестування наступного покоління, впроваджуючи штучний інтелект на кожному етапі життєвого циклу тестування. Applitools Eyes — це єдине ринкове рішення для копіювання 'людського ока' і автоматично виявляти функціональні та візуальні помилки з кожним випуском. Тепер із Applitools' Ultrafast Test Cloud, Ultrafast Grid і Execution Cloud – майже всі в організації бездоганно інтегровані в процес тестування – від команд проектування, розробки, контролю якості, операцій, маркетингу та продуктів.

Applitools' Флагманський продукт, Eyes, працює на основі технології Visual AI, яка копіює людське око та мозок, щоб фіксувати функціональні та візуальні помилки перед тим, як їх передадуть у виробництво. До інтеграції з Preflight потужність Applitools Eyes і Visual AI була доступна в основному групам інженерів, які мали навички писати та підтримувати тестовий код. Однак Applitools визнали зростаючий попит у галузі зробити Visual AI доступним для ширшої аудиторії. Протягом багатьох років Applitools Eyes інтегрувала Visual AI із фреймворками тестування з відкритим кодом, такими як Selenium і Cypress, щоб допомогти значно підвищити стабільність тестування, охоплення та підтримку. Тепер, завдяки Preflight, який спеціалізується на легкому автоматизованому створенні тестів, це означає, що не лише розробники та інженери, а й команди будь-якого рівня кваліфікації можуть створювати тести з використанням Visual AI.

"За останнє десятиліття Applitools' Технологія візуального штучного інтелекту допомогла командам інженерів із тестування та контролю якості у провідних глобальних організаціях ефективно виявляти функціональні та візуальні помилки у своїх продуктах і прискорювати їх тестування," сказав Моше Мілман, співзасновник і головний операційний директор Applitools. "Однак ми усвідомили необхідність зробити технологію Visual AI доступною для всіх в організації, незалежно від рівня кваліфікації. Тепер, незалежно від того, чи є хтось досвідченим інженером з контролю якості, ручним тестувальником або

зовсім новачком у світі тестування, усі вони можуть скористатися перевагами Visual AI. Інтеграція Preflight і Eyes є свідченням нашої відданості майбутньому ефективного, інклюзивного та комплексного тестування."

Завдяки інтеграції Preflight і Eyes Applitools прагне спростити й покращити процес автоматизації тестування завдяки таким перевагам:

- Кожен може автоматизувати: Завдяки можливостям Preflight без програмування навіть ті, хто не має глибоких знань програмування, тепер можуть використовувати потужність Visual AI through Eyes. Це демократизує процес тестування, дозволяючи більшій кількості членів команди ефективно сприяти автоматизації тестування.
- `Розширене покриття: Поєднання можливостей безкодового тестування Preflight із точністю Eyes' Візуальний штучний інтелект гарантує, що додатки та веб-сайти не лише функціонально правильні, але й візуально ідеальні в більшій частині інтерфейсу. Це має вирішальне значення на сучасному конкурентному ринку, де досвід користувача став великою відмінністю.
- Оптимізовані робочі процеси: тепер користувачі можуть розробляти свої тести в Preflight і миттєво використовувати потужність Eyes Visual AI, і все це в межах єдиної платформи. Це усуває потребу жонглювати між кількома інструментами, тим самим економлячи час і зменшуючи можливі помилки.

4.4. Автоматизована ідентифікація тестових випадків

Testim.io – це хмарна платформа, яка використовує ШІ для швидкого створення, виконання та обслуговування автоматизованих тестів.

Метод створення: Testim використовує машинне навчання для реєстрації та створення тестових випадків під час ручного тестування, після чого генерує автоматизовані тести на основі цих дій.

Отже, що Testim.io дозволяє вам робити:

- Створювати групи тестів і повторно використовувати їх у різних областях тестування. Таким чином, групи зберігають свої властивості від тесту до тесту і можуть бути вкладеними для кращої архітектури тесту.
- Використовувати умови та цикли в логіці тестів. Наприклад, оцінка стану присутності/відсутності певного елемента на сторінці допоможе вам визначити, чи виконуватиметься тестовий крок.
- Додати на свій крок наведення курсора та навігацію.
- Додати різні типи перевірок. Таким чином, можливо перевірити, чи результат програми відповідає очікуванням, включаючи значення або текст. Це означає, що можливо підтверджувати видимий елемент, невидимий елемент, текст елемента, пікселі елемента, пікселі вікна перегляду, пікселі документа…
- Додавати очікування та сон. Те саме, що в Selenium WebDriver!
- Додавати контрольні точки та відтворюйте тести з будь-якого кроку тестування.
- Виконувати тести локально, в режимі анонімного перегляду або в хмарі.
- Упорядковувати набори тестів, копіюючи тести, установлюючи мітки та запускаючи тести з мітками спеціально.
- Інтеграція з CI (TeamCity, Jenkins.io).
- Інтеграція з інструментами керування завданнями, такими як Jira та Trello, TestRail.

4.5. Покращення стійкості тестових сценаріїв

Appvance IQ — це інструмент автоматизації тестування на базі штучного інтелекту, який забезпечує тестування власних мобільних, мобільних веб-додатків і гібридних програм для iOS і Android [32]. Він задовольняє всі вимоги до тестування, включаючи функціональні тести, тести API, продуктивність і навантаження, забезпечуючи оптимальну роботу.

Для подальшого спрощення створення тестів Appvance IQ використовує AI Blueprinting для мобільних пристроїв. Ця функція автоматично визначає локатори, які можна використовувати в сценаріях Mobile Designer, що робить процес створення тесту ще простішим і ефективнішим.

Короткий опис переваг Appvance IQ у тестуванні на основі штучного інтелекту:

- AI Scripting може швидко генерувати тисячі сценаріїв, запускати їх і звітувати про результати.
- Тестування за допомогою штучного інтелекту в 1000 разів швидше, ніж тестування вручну або сценарії.
- ШІ-скрипти можуть покращити покриття коду порівняно з будь-яким іншим ручним або автоматизованим тестуванням.
- Тестування, кероване штучним інтелектом, може швидко знаходити реальні помилки користувача, повторне тестування та перевірку для швидкого реагування в розробниках.
- AI Scripting генерує тести, які представляють реальні дії користувача.
- Appvance IQ може покращувати тести для створення UX-тестів із тестів рівня API.
- Appvance IQ може використовувати ті самі сценарії для перевірки функціональності, продуктивності та безпеки.
- Appvance IQ може створювати тести для Інтернету, веб-сервісів, мобільних додатків і програм IOT.
- Тести Appvance IQ можна запускати за допомогою популярних інструментів CI і повністю підтримують методологію DevOps.

- Сценарії штучного інтелекту можуть змінювати та підтримувати випадки використання автоматично, коли змінюється програма.
- Сценарії AI можуть перевіряти за будь-якими критеріями, включаючи відомі коди відповідей, тайм-аути або очікувані відповіді.
- Appvance IQ доповнює або замінює більшість ручних завдань.

AIQ, AIOps, DevSecOps – спільна робота. Платформа *GAVS AIOps Zero Incident FrameworkTM (ZIFTM)* — це рішення зі штучним інтелектом, яке забезпечує 360° AIOps для зсуву вліво

Від традиційних IT-операцій. Використовуючи алгоритми машинного навчання (ML), ZIFTM передбачає потенційні збої та продуктивність деградації та потреби в ємності завчасно, з точністю 95%+. Це також виходить за рамки надання такої інформації прописуючи можливі рішення. ZIFTM має понад 250 ботів для автоматизації, які можна розгорнути для прискорення процесу вирішення проблем.

Прогнози на основі штучного інтелекту, що сприяють проактивному вирішенню гострих проблем, перш ніж вони переростуть в інциденти, ZIFTM значно збільшує надійність бізнес-додатків.

Послуги GAVS з розробки та розробки програмних продуктів очолюються AI/ML, Automation, Cloud, Big Data, Analytics та

DevSecOps. Ці послуги дають організаціям змогу прискорити інновації та розробку якісного програмного забезпечення за допомогою Agile методології розробки, наскрізний повністю автоматизований DevSecOps, автоматизація тестування рівня 5 AIQ і проактивна підтримка ZIFTM вирішення проблем, інтегроване в конвеєр DevSecOps. Можливості AIQ у поєднанні з автоматизованими функціями DevSecOps GAVS і функціями прогнозування, приписів і автоматичного виправлення на основі штучного інтелекту

ЗІФТМ складають потужну команду. Таким чином, потужність штучного інтелекту та машинного навчання зробить ваш SQA глибшим, швидшим і більшим, чим колись можливим раніше, забезпечуючи швидку доставку якісного програмного забезпечення клієнтам.

4.6. Виявлення та прогнозування дефектів

Jira — це система управління проєктами, яка дозволяє виконувати майже всі завдання PM — від планування до контролю процесів і результатів — за допомогою одного інструмента [33]. Збірка IT-рішень Atlassian, які об'єднані в Jira Family Products, включає Jira WM для роботи з бізнес-процесами, Jira SM для побудови сервіс-диску та Jira Software для програмних проектів.

Основні переваги Jira:

- Jira можна адаптувати до роботи з проектами будь-якої складності.
- великий потенціал для інтеграції. Jira може працювати з Github, Salesforce, Outlook, Slack, Gmail і Teams.
- можливість використовувати різні плагіни для розширення функціоналу. Найпопулярнішими програмами є Tempo, Script Runner, EazyBI, Big Picture і Structure.
- можливість використання методів Scrum або Kanban «з коробки»

Одним із недоліків Jira є складний інтерфейс і тривалий процес налаштування для конкретних робочих процесів. Через широку функціональність він має деяку правду, але ці недоліки можна вирішити, навчаючись інструменту та практикуючи його постійно. Тоді просто налаштувати Jira і оптимізувати робочі процеси за потреби.

Що PM має вивчити про Jira? Jira — це багатофункціональний помічник, схожий на конструктор Lego. Без інструкції ви швидше за все зможете щось зробити, але 70% деталей, або корисних функцій, не будуть використані. У результаті весь сенс використання інструменту буде втрачено.

Знаючи тонкощі роботи Jira, ви завжди зможете запропонувати цікаві ідеї щодо оптимізації процесів, які допоможуть підвищити рейтинг внутрішньої компанії. Крім того, є офіційна сертифікація Jira, що підвищує конкурентоспроможність експерта на ринку.

Отримання джерела ідей для розробки програмного забезпечення є ще однією причиною серйозно заглибитися у функціональність сервісу. Продумана архітектура, рішення щодо безпеки, реалізація окремих модулів і загальна

структура комплексу — це те, що ви можете використовувати для своїх власних проєктів.

Основні функціональні можливості Jira.

На ринку представлено три редакції сервісу:

- Jira Data Center — для великих компаній, які розгортають Jira у своїй інфраструктурі і для яких важливо мати масштабоване та стійке до відмов рішення.
- Jira Server — редакція офіційно підтримується до 2024 року, але вже не продається.
- Jira Cloud — найбільш актуальна редакція з хмарним сховищем, яку юзає багато проєктних та продуктових менеджерів.

Саме з прикладу «хмари» розглянемо основні функції інструмента.

Створення нового проєкту в адмінці — це перша точка роботи в Jira, як показано на рис. 4.5. Система надає широкий вибір шаблонів і можливість вибрати метод роботи відповідно до особливостей кожної людини. Щоб визначити тип проєкту, Jira запропонує визначити, керований компанією чи командою. Дуже важливо приймати правильні рішення.

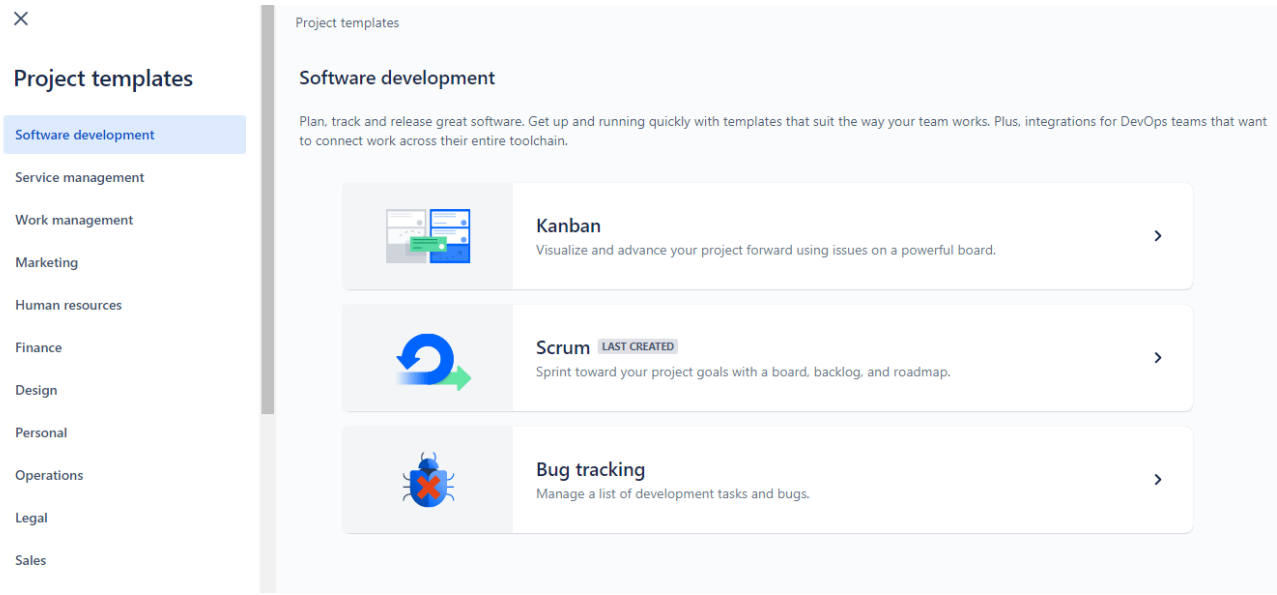


Рис. 4.5. Типи проєктів у Jira

У таблиці 4.1 описані особливості кожного з них.

Класифікація завдань, ієрархія та поля. Для створення завдань Jira можна використовувати готові шаблони або змінити їх відповідно до конкретного випадку, як і для створення ролей.

Таблиця 4.1.

Особливості типів проєктів

Проєкт під керуванням команди	Проєкт під управлінням компанії
Команда залежить від адміну. Будь-який учасник проєкту може налаштовувати Workflow.	Налаштовується та підтримується адміністраторами Jira.
Налаштування не впливають на інші проєкти.	Налаштування можна розшарити між проєктами.
Спрощене налаштування типів завдань та полів. Лише один тип на рівні підзавдань.	Повний контроль над типами завдань і полями, що налаштовуються.
Спрощені можливості автоматизації.	Повний спектр можливостей автоматизації.
Спрощена модель доступу. Учасник може мати лише одну роль.	Гнучке налаштування доступів та дозволів. Учасник може мати декілька ролей.
Одна дошка у проєкті.	Можливість мати кілька дощок у проєкті.

Ієрархія Jira складається з трьох рівнів (рис. 4.6). Перший містить епічні твори, другий містить історії, завдання, помилки та кастомізовані типи, а третій містить підзавдання та кастомізовані підтипи.

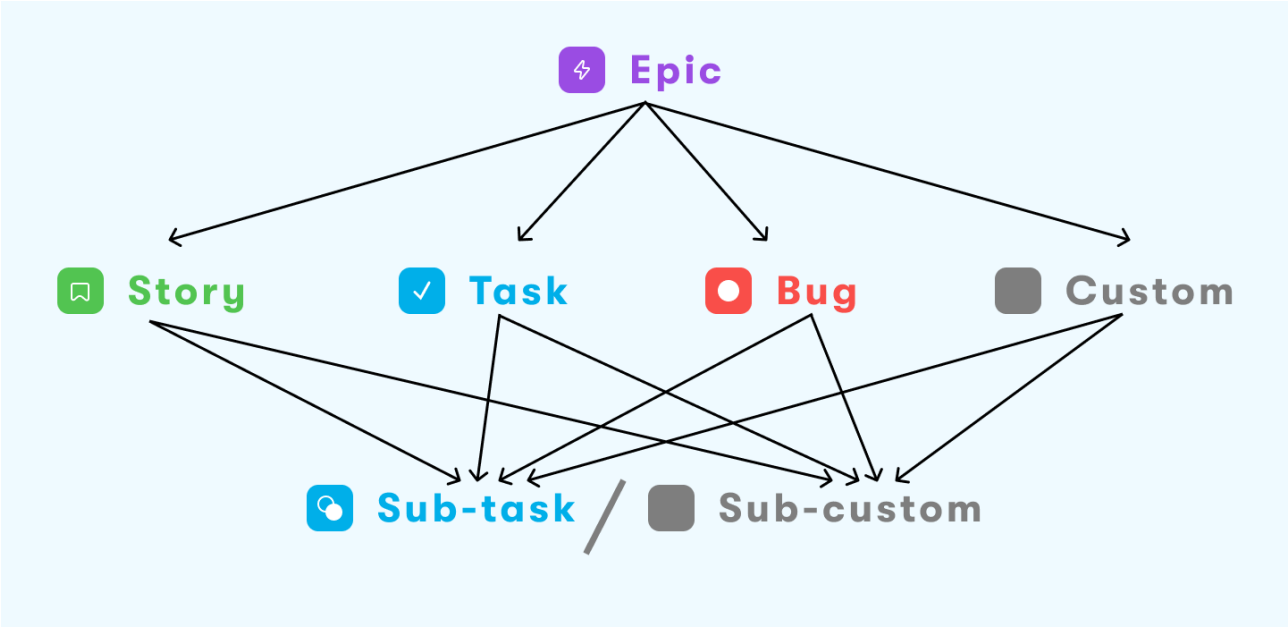


Рис. 4.6 Рівні ієрархії Jira

4.7. Оптимізація тестових даних

TOSCA Testsuite - це програмний інструмент для автоматизованого виконання функціонального та регресійного тестування програмного забезпечення [34].

Окрім функцій автоматизації тестування, TOSCA включає

- Інтегроване управління тестами
- Графічний інтерфейс користувача (GUI)
- Інтерфейс командного рядка (CLI)
- Інтерфейс прикладного програмування (API)

Набір тестів підтримує весь життєвий цикл тестового проекту. Вона починається з передачі та синхронізації специфікацій із системи керування вимогами.

TOSCA підтримує своїх користувачів у створенні ефективних тестів на методологічно обґрунтованій основі, виконує функції виконавчого помічника та підсумовує результати тестів у різних звітах.

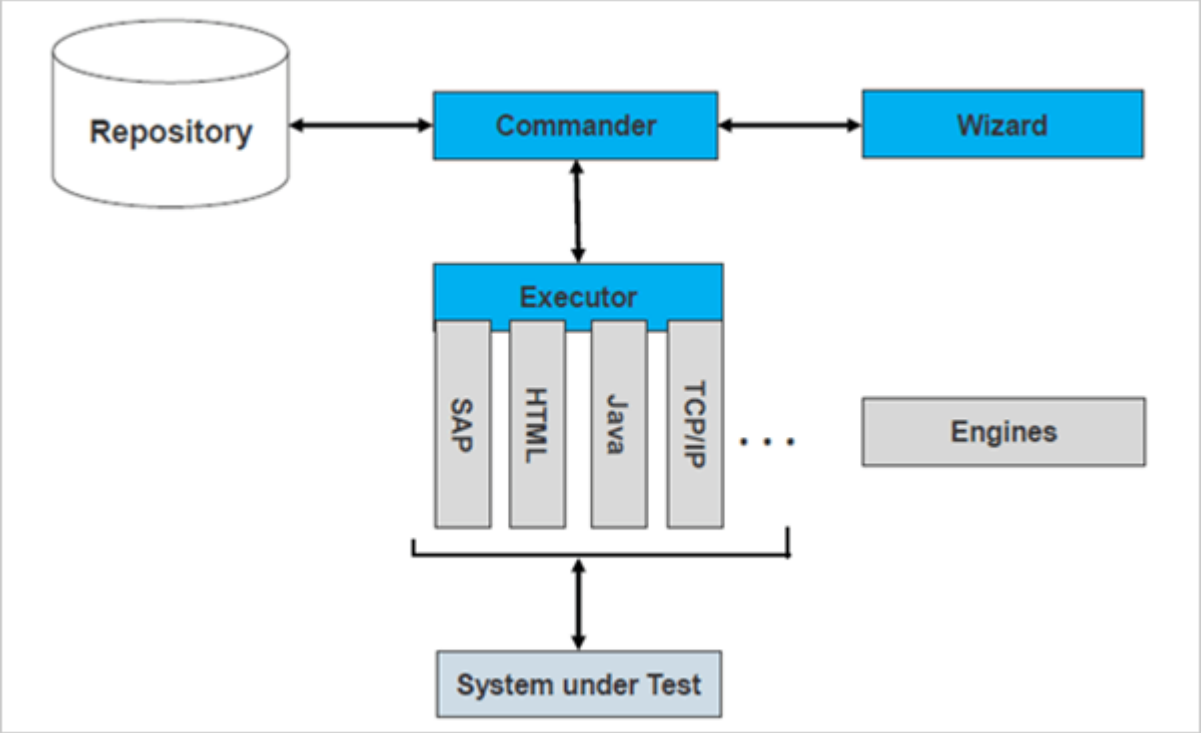


Рис. 4.7. Різні компоненти TOSCA Testsuite

Як показано на рисунку 4.7, різні компоненти набору тестів є:

- Командир TOSCA
- Майстер TOSCA
- Виконавець TOSCA

Усі ці три знаходяться на стороні клієнта, він також включає репозиторій (також званий «тестовим репозиторієм»), який знаходиться на стороні сервера.

TOSCA Commande. Це графічний інтерфейс користувача TOSCA Testsuite. Він вважається ядром набору тестів. Командир використовує «Робочу область» для адміністрування тестів. Це означає, що він дозволяє легко створювати, керувати, виконувати та аналізувати тестові випадки.

Оскільки це система проміжного програмного забезпечення між тестовим репозиторієм і TOSCA Executor, вона отримує тестові приклади зі сховища та пересилає їх до тестового виконавця, який пізніше запускає їх у тестовій системі (SUT).

Усі елементи відображаються у вигляді дерева (приклад знімка екрана вище). Ліва частина вікна використовується для навігації, тоді як права частина є робочою областю.

Наведений вище знімок екрана є зразком вікна «Тестовий приклад», макет інших вікон (Вимоги, Список виконання тощо) виглядає так само. Усі елементи в TOSCA Commande структуровані один під одним у суворо дотриманій ієрархічній послідовності. Кожну операцію можна виконати, лише дотримуючись цієї ієрархії об’єктів.

Він забезпечує функцію перетягування , яка використовується для переміщення елементів у програмі. Він також має функцію закріплення , яка дозволяє користувачеві налаштовувати макет вікна відповідно до потреби.

Тож TOSCA Commande надає користувачам такі функції та функції для їхньої зручності. Він працює так само, як Windows Explorer. Під час створення структури папок можна використовувати такі команди, як створення, копіювання, вставлення, перейменування, видалення тощо.

Робочий простір TOSCA. Це ваша особиста робоча зона, де ви можете створювати, адмініструвати, виконувати й аналізувати тестові випадки. Він містить різні об’єкти, які називаються об’єктами TOSCA Commander і це,

- Модулі
- Списки виконання
- TestCases
- Вимоги
- Дизайн тестового випадку

Ви можете побудувати зв’язок між цими об’єктами, відображаючи/зв’язуючи їх. У TOSCA це називається відображенням об’єктів. Під час виконання керуюча інформація цих об’єктів (модулі, списки виконання, тестові випадки та вимоги тощо) об’єднується.

4.8. Моніторинг результатів тестування

Splunk — це платформа для збірки, зберігання, обробки й аналізу машинних даних, а також є логів [35]. Сьогоднішній день є надзвичайно популярним у США та Європі та поступово виходить на інші ринки, включаючи Росію. Одна з головних особливостей платформи полягає в тому, що вона може працювати з даними практично з будь-яких джерел, тому список можливих застосувань системи дуже широкий.

Splunk у більшості випадків (автоматично або за допомогою аддонів) розбирає вхідні дані на поля та значення, а в подальшому обробляє їх. Обробка відбувається за допомогою запитів SPL (спеціальна мова від Splunk), за допомогою яких можна будувати різні вибори та таблиці, сортувати, фільтрувати, агрегувати за допомогою, будувати звіти, створювати вичислювані поля, звертатися як до внутрішніх, так і зовнішніх довідників, створювати інформаційні панелі, з широким спектром візуалізації та робити алерти (наприклад, за результатами виконання запиту відправити тикети в Service Desk). Все це можна упакувати у свій персональний додаток.

Основні відмінності або сильні сторони Splunk

- Real Time Architecture: Splunk здійснює збір, пошук, моніторинг і аналіз за різними і досить великими (сотні ТБ даних в день) обсягами даних в режимі реального часу і все це — одна система.

Чому це важливо? Тому Splunk може забезпечити збір даних у реальному часі з тисяч різноманітних джерел — і це може бути як фізичний, так і віртуальний хост, так і хмарно. Також Splunk підтримує пошук не тільки в реальному часі, але й у всьому тимчасовому проміжку, дані для якого були зібрані. Це ми можемо здійснювати пошук, моніторинг, оповіщення, звітність та аналіз за будь-який час (історичні дані та дані в реальному часі в одному рішенні). І нарешті, Splunk забезпечує швидкий результат і високу інтерактивність пошукових запитів на надзвичайно великих обсягах даних.

- Універсальна платформа машинних даних: Splunk є універсальною платформою для машинних даних, яка забезпечує комплексний збір даних, їх обробку та аналіз. Таким чином ми можемо індексувати будь-які машинні дані з відміткою про час незалежності від структури та формату. Splunk здатний

об'єднати в себе машинні дані + бізнес-дані + дані користувача, що робить його надзвичайно універсальним.

- Schema on the Fly: Splunk здійснює пошук за часом, тож вам не потрібно заздалегідь знати структуру даних, щоб сформувати запит. Ви можете вибрати проміжок часу, ввести пару ключових слів і швидко познакомитися з даними. Немає ніяких жорстких обмежень на столбці, таблиці і прочее. Це сильно підвищує гнучкість системи. Також будь-який запит можна встановити, поставити на паузу або показати проміжні результати.
- Agile Reporting & Analytics: Splunk надає широкі можливості для побудови аналітики, звітів та їх візуалізації. Окрім цільових даних, система також може звертатися до зовнішніх справочників, наприклад у SQL BD. Також хотілося б сказати, що Splunk достатньо відкрита система, і ви завжди можете додати свій модуль, хоча можливості візуалізації дуже різноманітні.
- Масштабується від Desktop до Enterprise: Splunk використовує технологію MapReduce, яка забезпечує розподіл навантажень і горизонтальну масштабованість системи, тож ми можемо завантажити з одного сервера для Splunk, а при збільшенні даних — швидко додати пару нових серверів і розподілити навантаження. Також завдяки технології MapReduce Splunk можна швидко переробляти реально більші об’єми даних, які не вимагають виданого заліза.
- Швидкий час отримання вартості: Splunk дозволяє швидко отримати результат від використання. Внедрение занимает часы или дни, а не недели и месяцы. Тоже самое з масштабуванням і експлуатацією.
- Пристрасна та яскрава спільнота: у Splunk є дуже якісний і головний безкоштовний ком’юніти, який включає:
- Splunk Base — портал, що містить всеможливі додатки та аддони, 99% з яких безкоштовно
- Splunk Answers — форум з великим числом питань/відповідей і живих учасників
- Splunk Dev — портал для розробників

- Splunk Dock — повна база знань продуктів

4.9. Реалізація алгоритму пошуку кращого сценарію з використанням методу Монте-Карло

Розглянемо класи, які буде використано під час програмування програмного продукту

MonteCarloTreeSearchNode (Абстрактний клас) - цей клас визначає загальну структуру вузла дерева MCTS. Кожний вузол представляє можливий стан [36].

Для ініціалізації вузла з вказоного стану та батьківсього вузла використовується **"*_init_".**

"q", "n": Абстрактны властивосты, які повертають значення q(загальна кількість) та n(кількість ітеризацій).

Для абстрактних методів, які відповідають за розширення вузла, визначення того, чи він є термінальним, проведення симуляції та оновлення статистики використовуємо **"*expand*"**, **"*is_terminal_node*"**, **"*rollout*"**, **"*backpropagate*"**.

T*woScenaryMonteCarloTreeSearchNode* (Спадкоємець класу MonteCarloTreeSearchNode для мультизадачності)

- Додатково визначає статистику для обчислення q (якості) на основі випадкових симуляцій.
- untried_actions: Перевизначає властивість, щоб повертати список можливих дій на основі поточного стану симуляції.
- q: Обчислює різницю між якістю і кількістю ітерацій.
- n: Кількість ітерацій до даного вузла.
- expand, is_terminal_node, rollout, backpropagate: Реалізує абстрактні методи.

***MonteCarloTreeSearch*:**

- Основний клас, який управляє пошуком у дереві MCTS.
- **_init_:** Ініціалізація з кореневим вузлом.
- **best_action**: Знаходження найкращого результату за допомогою вказаної кількості симуляцій.
- **_tree_policy**: Вибір вузла для проведення симуляції.

State:

- Клас, що представляє стан.
- __init__: Ініціалізація з початковими параметрами
- next_state: Генерація наступного стану.
- terminal: Перевірка, чи симуляція завершилась.
- reward: Обчислення вихідного результату.
- Інші методи для отримання кешу, порівняння, та представлення стану.

Node:

- Клас, що представляє вузол дерева.
- __init__: Ініціалізація зі станом та батьківським вузлом.
- add_child: Додавання дочірнього вузла.
- update: Оновлення вузла на основі отриманого результату.
- fully_expanded: Перевірка, чи вузол повністю розгорнутий.

ВИСНОВКИ

В цьому дослідженні розглядається значення тестування в навчальному процесі, а тестування процесу автоматизації важливість також Досліджено способи створення окремих тестових варіантів для систем, які вже існують, а також їхні структурні характеристики та особливості їх реалізації. Досліджено методи створення окремих тестових варіантів, які описані у відповідних джерелах, але не можуть бути використані в системах, які є вільним доступом.

На основі проведених досліджень було розроблено та визначено завдання, які повинні бути вирішені в процесі створення ПП. Розроблено новий спосіб створення індивідуальних тестових варіантів, використовуючи переваги методів, які були перевірені в існуючих системах тестування, технології інтелектуального аналізу даних і методи, які були лише теоретично описані. Модель даних була розроблена та обрана найзручніша технологія для її впровадження для виконання завдань.

ПП були розроблені на основі завдань, які були виконані, а також технологій для їх вирішення. Виявлено кілька можливих напрямків подальшого дослідження та вдосконалення ПП. Одним із них було б створення можливості динамічного обрання кількості рівнів складності запитань відповідно до вимог користувача. Іншим варіантом було б динамічне обрання кількості запитань у межах варіанту тесту з урахуванням обмежень і результатів тестів, що дозволило б проводити не лише тестування з метою перевірки рівня знань, а й навчальне тестування.

Вдосконалення впровадження даних дозволяють лише розширювати існуючі функції, не вимагаючи повної переробки ПП. Крім того, БД, яка була обрана для зберігання тестових завдань, не потребує структурних змін.

Список використаних джерел

1. Parsa S. Fault Localization Tools: Diagnosis Matrix and Slicing. Software Testing Automation. Cham, 2023. С. 333–364. URL: https://doi.org/10.1007/978-3-031-22057-9_8

2. Winkler D., Hametner R., Biffl S. Automation component aspects for efficient unit testing. Factory Automation (ETFA 2009), м. Palma de Mallorca, Spain, 22–25 верес. 2009 р. 2009. URL: https://doi.org/10.1109/etfa.2009.5347022

3. Пчелянський Д. П., Воінова С. А. ШТУЧНИЙ ІНТЕЛЕКТ: ПЕРСПЕКТИВИ ТА ТЕНДЕНЦІЇ РОЗВИТКУ. Automation of technological and business processes. 2019. Т. 11, № 3. С. 59–64. URL: https://doi.org/10.15673/atbp.v11i3.1500

4. Jordan M. I., Mitchell T. M. Machine learning: Trends, perspectives, and prospects. Science. 2015. Vol. 349, no. 6245. P. 255–260. URL: https://doi.org/10.1126/science.aaa8415

5. Марченко А. В. Комп'ютерна автоматизована система визначення тональності тексту : магістерська робота. 2020. URL: https://dspace.znu.edu.ua/jspui/handle/12345/4999

6. Aggarwal C. C. Training Deep Neural Networks. Neural Networks and Deep Learning. Cham, 2018. P. 105–167. URL: https://doi.org/10.1007/978-3-319-94463-0_3

7. Bishop C. M., Bishop H. Graph Neural Networks. Deep Learning. Cham, 2023. P. 407–427. URL: https://doi.org/10.1007/978-3-031-45468-4_13

8. Software Testing from an Agile and Traditional view / S. Najihi et al. Procedia Computer Science. 2022. Vol. 203. P. 775–782. URL: https://doi.org/10.1016/j.procs.2022.07.116

9. Green B. S. Software test automation. ACM SIGSOFT Software Engineering Notes. 2000. Vol. 25, no. 3. P. 66. URL: https://doi.org/10.1145/505863.505892

10. Analytical Evaluation of Web Performance Testing Tools: Apache JMeter and SoapUI / V. Tiwari et al. 2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT), Bhopal, India, 8–9 April 2023. 2023. URL: https://doi.org/10.1109/csnt57126.2023.10134699

11. SEMA G., Konté M. A., Diongue A. K. The spread of the coronavirus is putting a strain on financial markets and the resulting stock market volatility is causing huge problems for investors. Volatility in the U.S. market has returned to levels not seen since the 2011 sovereign debt crisis. It is already clear that this volatility has had a negative effect on the economy. In this study, we introduce a regime-switching GJR-GARCH modelwi. African Journal of Applied Statistics. 2021. T. 8, № 1. C. 1049–1071. URL: https://doi.org/10.16929/ajas/2021.1049.257

12. Maniar K. B., Khatri C. B. Data Science: Bigtable, MapReduce and Google File System. International Journal of Computer Trends and Technology. 2014. Vol. 16, no. 3. P. 115–118. URL: https://doi.org/10.14445/22312803/ijctt-v16p128

13. Exploring the use of machine learning for interpreting electrochemical impedance spectroscopy data: evaluation of the training dataset size / V. Bongiorno et al. Corrosion Science. 2022. Vol. 198. P. 110119. URL: https://doi.org/10.1016/j.corsci.2022.110119

14. Generation of a Scenario Library for Testing driver-automation Cooperation Safety under Cut-in Working Conditions / H. Guo et al. Green Energy and Intelligent Transportation. 2022. P. 100004. URL: https://doi.org/10.1016/j.geits.2022.100004

15. Salehi H., Burgueño R. Emerging artificial intelligence methods in structural engineering. Engineering Structures. 2018. Vol. 171. P. 170–189. URL: https://doi.org/10.1016/j.engstruct.2018.05.084

16. Walker E., Lange T., Cramer W. VERAView Software Requirements, Test Plan, and Test Report. Office of Scientific and Technical Information (OSTI), 2022. URL: https://doi.org/10.2172/1881108

17. Artistas e produção musical da Vindouro | Wine and History | Festa Pombalina 2023 : Anúncio de procedimento no. 4489/2023. Diário da República II Série. 2023. 22 March. URL: https://files.dre.pt/cp_hora/2023/03/058/416298273.pdf

18. Effects of liraglutide on depressive behavior in a mouse depression model and cognition in the probe trial of Morris water maze test / M. K. Seo et al. Journal of Affective Disorders. 2023. Vol. 324. P. 8–15. URL: https://doi.org/10.1016/j.jad.2022.12.089

19. Validity and reliability of speed tests used in soccer: A systematic review / S. Altmann et al. PLOS ONE. 2019. Vol. 14, no. 8. P. e0220982. URL: https://doi.org/10.1371/journal.pone.0220982

20. Concurrent validity and reliability of global positioning systems for measuring sprint and peak speed performance: A systematic review / H. Nobari et al. INPLASY - International Platform of Registered Systematic Review and Meta-analysis Protocols, 2021. URL: https://doi.org/10.37766/inplasy2021.6.0007

21. A systematic review of the criterion validity and reliability of technical and tactical field-based tests in soccer / F. M. Clemente et al. International Journal of Sports Science & Coaching. 2022. P. 174795412210852. URL: https://doi.org/10.1177/17479541221085236

22. Stewart P. F., Turner A. N., Miller S. C. Reliability, factorial validity, and interrelationships of five commonly used change of direction speed tests. Scandinavian Journal of Medicine & Science in Sports. 2012. Vol. 24, no. 3. P. 500–506. URL: https://doi.org/10.1111/sms.12019

23. Performance Analysis of Different Types of Machine Learning Classifiers for Non-Technical Loss Detection / K. M. Ghori et al. IEEE Access. 2020. Vol. 8. P. 16033–16048. URL: https://doi.org/10.1109/access.2019.2962510

24. Ivanov M. M., Kaurov A. A., Sibiryakov S. Non-perturbative probability distribution function for cosmological counts in cells. Journal of Cosmology and Astroparticle Physics. 2019. Vol. 2019, no. 03. P. 009. URL: https://doi.org/10.1088/1475-7516/2019/03/009

25. Cosmological constraints from the convergence 1-point probability distribution / K. Patton et al. Monthly Notices of the Royal Astronomical Society. 2017. Vol. 472, no. 1. P. 439–446. URL: https://doi.org/10.1093/mnras/stx1626

26. Chudaykin A., Ivanov M. M., Sibiryakov S. Renormalizing one-point probability distribution function for cosmological counts in cells. Journal of Cosmology and Astroparticle Physics. 2023. Vol. 2023, no. 08. P. 079. URL: https://doi.org/10.1088/1475-7516/2023/08/079

27. Multinomial Naive Bayes Based Machine Learning Analysis of Twitter Sentiment / U. K. B. Saravanan et al. 2023 2nd International Conference on Edge Computing and Applications (ICECAA), Namakkal, India, 19–21 July 2023. 2023. URL: https://doi.org/10.1109/icecaa58104.2023.10212150

28. Burgic V., Keco D. SMS CLASSIFICATION: CONJOINT ANALYSIS OF MULTINOMIAL NAIVE BAYES APPLICATION. International Journal of Advanced Research. 2021. Vol. 9, no. 08. P. 1165–1173. URL: https://doi.org/10.21474/ijar01/13366

29. Multinomial Naive Bayes Classifier for Sentiment Analysis of Internet Movie Database / C. Dewi et al. Vietnam Journal of Computer Science. 2023. URL: https://doi.org/10.1142/s2196888823500100

30. A Decentralized, Flat-Structured Control System for Chiller Plants / D. He et al. Applied Sciences. 2019. Vol. 9, no. 22. P. 4811. URL: https://doi.org/10.3390/app9224811

31. Decentralized Storage Rental System / D. M. Pradhan et al. International Journal for Research in Applied Science and Engineering Technology. 2023. Vol. 11, no. 11. P. 1475–1479. URL: https://doi.org/10.22214/ijraset.2023.56790

32. Ricca F., Marchetto A., Stocco A. A Retrospective Analysis of Grey Literature for AI-Supported Test Automation. Communications in Computer and Information Science. Cham, 2023. P. 90–105. URL: https://doi.org/10.1007/978-3-031-43703-8_7

33. Ricca F., Marchetto A., Stocco A. AI-based Test Automation: A Grey Literature Analysis. 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Porto de Galinhas, Brazil, 12–16 April 2021. 2021. URL: https://doi.org/10.1109/icstw52544.2021.00051

34. Parsa S. Search-Based Testing. Software Testing Automation. Cham, 2023. P. 427–460. URL: https://doi.org/10.1007/978-3-031-22057-9_11

35. Parsa S. Fault Localization Tools: Diagnosis Matrix and Slicing. Software Testing Automation. Cham, 2023. P. 333–364. URL: https://doi.org/10.1007/978-3-031-22057-9_8

36. Xu L., Dockhorn A., Perez-Liebana D. Elastic Monte Carlo Tree Search. IEEE Transactions on Games. 2023. P. 1–11. URL: https://doi.org/10.1109/tg.2023.3282351

ДОДАТКИ

Додаток А

```
import numpy as np
from collections import defaultdict
from abc import ABC, abstractmethod


class MonteCarloTreeSearchNode(ABC):
    def __init__(self, state, parent=None):

        self.state = state
        self.parent = parent
        self.children = []

    @property
    @abstractmethod
    def untried_actions(self):
        """
        Returns
        -------
        """
        pass

    @property
    @abstractmethod
    def q(self):
        pass

    @property
    @abstractmethod
    def n(self):
        pass

    @abstractmethod
    def expand(self):
        pass

    @abstractmethod
    def is_terminal_node(self):
        pass

    @abstractmethod
    def rollout(self):
        pass

    @abstractmethod
    def backpropagate(self, reward):
        pass

    def is_fully_expanded(self):
        return len(self.untried_actions) == 0

    def best_child(self, c_param=1.4):
        choices_weights = [
            (c.q / c.n) + c_param * np.sqrt((2 * np.log(self.n) / c.n))
            for c in self.children
        ]
        return self.children[np.argmax(choices_weights)]

    def rollout_policy(self, possible_moves):
        return possible_moves[np.random.randint(len(possible_moves))]
```

```
class TwoPlayersGameMonteCarloTreeSearchNode(MonteCarloTreeSearchNode):
    def __init__(self, state, parent=None):
        super().__init__(state, parent)
        self._number_of_visits = 0.
        self._results = defaultdict(int)
        self._untried_actions = None

    @property
    def untried_actions(self):
        if self._untried_actions is None:
            self._untried_actions = self.state.get_legal_actions()
        return self._untried_actions

    @property
    def q(self):
        wins = self._results[self.parent.state.next_to_move]
        loses = self._results[-1 * self.parent.state.next_to_move]
        return wins - loses

    @property
    def n(self):
        return self._number_of_visits

    def expand(self):
        action = self.untried_actions.pop()
        next_state = self.state.move(action)
        child_node = TwoPlayersGameMonteCarloTreeSearchNode(
            next_state, parent=self
        )
        self.children.append(child_node)
        return child_node

    def is_terminal_node(self):
        return self.state.is_game_over()

    def rollout(self):
        current_rollout_state = self.state
        while not current_rollout_state.is_game_over():
            possible_moves = current_rollout_state.get_legal_actions()
            action = self.rollout_policy(possible_moves)
            current_rollout_state = current_rollout_state.move(action)
        return current_rollout_state.game_result

    def backpropagate(self, result):
        self._number_of_visits += 1.
        self._results[result] += 1.
        if self.parent:
            self.parent.backpropagate(result)


class MonteCarloTreeSearch(object):
    def __init__(self, node):
        """
        MonteCarloTreeSearchNode
        Parameters
        ----------
        """
        self.root = node

    def best_action(self, simulations_number):
        """
        Parameters
        ----------
        simulations_number : int
            number of simulations performed to get the best action
        Returns
```

```
        -------
        """
        for _ in range(0, simulations_number):
            v = self._tree_policy()
            reward = v.rollout()
            v.backpropagate(reward)
        # to select best child go for exploitation only
        return self.root.best_child(c_param=0.)

    def _tree_policy(self):
        """
        selects node to run for
        Returns
        -------
        """
        current_node = self.root
        while not current_node.is_terminal_node():
            if not current_node.is_fully_expanded():
                return current_node.expand()
            else:
                current_node = current_node.best_child()
        return current_node


import random
import math
import hashlib
import argparse

SCALAR = 1 / math.sqrt(2.0)


class State():
    NUM_TURNS = 10
    GOAL = 0
    MOVES = [2, -2, 3, -3]
    MAX_VALUE = (5.0 * (NUM_TURNS - 1) * NUM_TURNS) / 2
    num_moves = len(MOVES)

    def __init__(self, value=0, moves=[], turn=NUM_TURNS):
        self.value = value
        self.turn = turn
        self.moves = moves

    def next_state(self):
        nextmove = random.choice([x * self.turn for x in self.MOVES])
        next = State(self.value + nextmove, self.moves + [nextmove], self.turn -
1)
        return next

    def terminal(self):
        if self.turn == 0:
            return True
        return False

    def reward(self):
        r = 1.0 - (abs(self.value - self.GOAL) / self.MAX_VALUE)
        return r

    def __hash__(self):
        return int(hashlib.md5(str(self.moves).encode('utf-8')).hexdigest(), 16)

    def __eq__(self, other):
        if hash(self) == hash(other):
            return True
        return False
```

```
    def __repr__(self):
        s = "Value: %d; Moves: %s" % (self.value, self.moves)
        return s


class Node():
    def __init__(self, state, parent=None):
        self.visits = 1
        self.reward = 0.0
        self.state = state
        self.children = []
        self.parent = parent

    def add_child(self, child_state):
        child = Node(child_state, self)
        self.children.append(child)

    def update(self, reward):
        self.reward += reward
        self.visits += 1

    def fully_expanded(self):
        if len(self.children) == self.state.num_moves:
            return True
        return False

    def __repr__(self):
        s = "Node; children: %d; visits: %d; reward: %f" % (len(self.children),
self.visits, self.reward)
        return s


def UCTSEARCH(budget, root):
    for iter in range(int(budget)):
        if iter % 10000 == 9999:
            logger.info("simulation: %d" % iter)
            logger.info(root)
        front = TREEPOLICY(root)
        reward = DEFAULTPOLICY(front.state)
        BACKUP(front, reward)
    return BESTCHILD(root, 0)


def TREEPOLICY(node):
    # a hack to force 'exploitation' in a game where there are many options, and
you may never/not want to fully expand first
    while node.state.terminal() == False:
        if len(node.children) == 0:
            return EXPAND(node)
        elif random.uniform(0, 1) < .5:
            node = BESTCHILD(node, SCALAR)
        else:
            if node.fully_expanded() == False:
                return EXPAND(node)
            else:
                node = BESTCHILD(node, SCALAR)
    return node


def EXPAND(node):
    tried_children = [c.state for c in node.children]
    new_state = node.state.next_state()
    while new_state in tried_children:
        new_state = node.state.next_state()
    node.add_child(new_state)
```

```
    return node.children[-1]


# current this uses the most vanilla MCTS formula it is worth experimenting with
THRESHOLD ASCENT (TAGS)
def BESTCHILD(node, scalar):
    bestscore = 0.0
    bestchildren = []

    for c in node.children:
        exploit = c.reward / c.visits
        explore = math.sqrt(2.0 * math.log(node.visits) / float(c.visits))
        score = exploit + scalar * explore
        if score == bestscore:
            bestchildren.append(c)
        if score > bestscore:
            bestchildren = [c]
            bestscore = score
    if len(bestchildren) == 0:
        logger.warn("OOPS: no best child found, probably fatal")
    return random.choice(bestchildren)


def DEFAULTPOLICY(state):
    while state.terminal() == False:
        state = state.next_state()
    return state.reward()


def BACKUP(node, reward):
    while node != None:
        node.visits += 1
        node.reward += reward
        node = node.parent
    return


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='MCTS research code')
    parser.add_argument('--num_sims', action="store", required=True, type=int)
    parser.add_argument('--levels', action="store", required=True, type=int,
choices=range(State.NUM_TURNS))
    args = parser.parse_args()

    current_node = Node(State())
    for l in range(args.levels):
        current_node = UCTSEARCH(args.num_sims / (l + 1), current_node)
        print("level %d" % l)
        print("Num Children: %d" % len(current_node.children))
        for i, c in enumerate(current_node.children):
            print(i, c)
        print("Best Child: %s" % current_node.state)
        import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

kmeans = KMeans(n_clusters=3, init='k-means++', n_init=20, max_iter=300,
tol=0.00001, precompute_distances='auto',
                verbose=0, random_state=None, copy_x=True, n_jobs=None,
algorithm='auto')
kmeans = kmeans.fit(X)
abels = kmeans.predict(X)
C = kmeans.cluster_centers_
```

```
fig = plt.figure()

import click

from py2neo.meta import NEO4J_URI, NEO4J_AUTH


@click.group(help="""\
Multipurpose Neo4j toolkit.
""")
def py2neo():
    pass


@py2neo.command("version", help="""\
Display the current library version
""")
def py2neo_version():
    from py2neo.meta import __version__
    click.echo(__version__)


@py2neo.command("console", help="""\
Interactive Cypher console
""")
@click.option("-u", "--uri", default=NEO4J_URI, help="Set the connection URI.")
@click.option("-a", "--auth", default=NEO4J_AUTH, help="Set the user and
password.")
@click.option("-s", "--secure", is_flag=True, default=False, help="Use encrypted
communication (TLS).")
@click.option("-v", "--verbose", is_flag=True, default=False, help="Show low
level communication detail.")
def py2neo_console(uri, auth=None, secure=None, verbose=None):
    from py2neo.console import Console, ConsoleError
    try:
        con = Console(uri, auth=auth, secure=secure, verbose=verbose)
    except ConsoleError as error:
        click.echo(error)
        raise SystemExit(1)
    else:
        raise SystemExit(con.loop())


def py2neo_auth():
    pass


def py2neo_auth_list(auth_file):
    from py2neo.admin.install import AuthFile
    for user in AuthFile(auth_file):
        click.echo(user.name)


@click.argument("auth_file")
@click.argument("user_name")
def py2neo_auth_remove(auth_file, user_name):
    from py2neo.admin.install import AuthFile
    AuthFile(auth_file).remove(user_name)


@click.argument("auth_file")
@click.argument("user_name")
@click.password_option()
def py2neo_auth_update(auth_file, user_name, password):
    from py2neo.admin.install import AuthFile
    AuthFile(auth_file).update(user_name, password)
```

```
def py2neo_get():
    from py2neo.admin.dist import Distribution
    Distribution().download()


@click.option("-u", "--uri", default=NEO4J_URI, help="Set the connection URI.")
@click.option("-a", "--auth", default=NEO4J_AUTH, help="Set the user and
password.")
@click.option("-s", "--secure", is_flag=True, default=False, help="Use encrypted
communication (TLS).")
@click.option("-v", "--verbose", is_flag=True, default=False, help="Show low
level communication detail.")
@click.argument("cypher", nargs=-1)
def py2neo_run(cypher, uri, auth=None, secure=None, verbose=None):
    from py2neo.console import Console, ConsoleError
    try:
        con = Console(uri, auth=auth, secure=secure, verbose=verbose)
    except ConsoleError as error:
        click.echo(error)
        raise SystemExit(1)
    else:
        raise SystemExit(con.run_all(cypher))


def main():
    try:
        py2neo(obj={})
    except Exception as error:
        click.secho(error.args[0], err=True)
        exit(1)
    else:
        exit(0)


if __name__ == "__main__":
    main()
import re

from py2neo.cypher import cypher_escape, cypher_repr
from py2neo.data import Node
from py2neo.internal.collections import is_collection
from py2neo.internal.compat import Sequence, Set

_operators = {
    "exact": "=",
    "not": "<>",
    "regex": "=~",
    "gt": ">", "gte": ">=",
    "lt": "<", "lte": "<=",
    "startswith": "STARTS WITH",
    "endswith": "ENDS WITH",
    "contains": "CONTAINS",
}

_operators_search = "^(.+)__(%s)$" % "|".join(_operators.keys())


def _property_conditions(properties, offset=1):
    for i, (key, value) in enumerate(properties.items(), start=offset):
        if key == "__id__":
            condition = "id(_)"
        else:
            condition = "_.%s" % cypher_escape(key)
        if value is None:
```

```
            condition += " IS NULL"
            parameters = {}
        elif isinstance(value, (tuple, set, frozenset)):
            condition += " IN {%d}" % i
            parameters = {"%d" % i: list(value)}
        elif re.match(_operators_search, key):
            parts = re.search(_operators_search, key)
            prop = parts.group(1)
            operator = parts.group(2)
            condition = "_.%s %s {%d}" % (prop, _operators[operator], i)
            parameters = {"%d" % i: value}
        else:
            condition += " = {%d}" % i
            parameters = {"%d" % i: value}
        yield condition, parameters


class NodeMatch(object):
    """ Immutable set of node selection criteria.
    """

    def __init__(self, graph, labels=frozenset(), conditions=tuple(),
order_by=tuple(), skip=None, limit=None):
        self.graph = graph
        self._labels = frozenset(labels)
        self._conditions = tuple(conditions)
        self._order_by = tuple(order_by)
        self._skip = skip
        self._limit = limit

    def __len__(self):
        """ Return the number of nodes matched.
        """
        return self.graph.evaluate(*self._query_and_parameters(count=True))

    def __iter__(self):
        """ Iterate through all matching nodes.
        """
        for record in self.graph.run(*self._query_and_parameters()):
            yield record[0]

    def first(self):
        """ Evaluate the match and return the first :class:`.Node`
        matched or :const:`None` if no matching nodes are found.
        :return: a single matching :class:`.Node` or :const:`None`
        """
        return self.graph.evaluate(*self._query_and_parameters())

    def _query_and_parameters(self, count=False):
        """ A tuple of the Cypher query and parameters used to select
        the nodes that match the criteria for this selection.
        :return: Cypher query string
        """
        clauses = ["MATCH (_%s)" % "".join(":%s" % cypher_escape(label) for
label in self._labels)]
        parameters = {}
        if self._conditions:
            conditions = []
            for condition in self._conditions:
                if isinstance(condition, tuple):
                    condition, param = condition
                    parameters.update(param)
                conditions.append(condition)
            clauses.append("WHERE %s" % " AND ".join(conditions))
        if count:
            clauses.append("RETURN count(_)")
```

```
        else:
            clauses.append("RETURN _")
            if self._order_by:
                clauses.append("ORDER BY %s" % (", ".join(self._order_by)))
            if self._skip:
                clauses.append("SKIP %d" % self._skip)
            if self._limit is not None:
                clauses.append("LIMIT %d" % self._limit)
        return " ".join(clauses), parameters

    def where(self, *conditions, **properties):
        """ Refine this match to create a new match. The criteria specified
        for refining the match consist of conditions and properties.
        Conditions are individual Cypher expressions that would be found
        in a `WHERE` clause; properties are used as exact matches for
        property values.
        To refer to the current node within a condition expression, use
        the underscore character ``_``. For example::
            match.where("_.name =~ 'J.*'")
        Simple property equalities can also be specified::
            match.where(born=1976)
        :param conditions: Cypher expressions to add to the `WHERE` clause
        :param properties: exact property match keys and values
        :return: refined :class:`.NodeMatch` object
        """
        return self.__class__(self.graph, self._labels,
                              self._conditions + conditions +
tuple(_property_conditions(properties)),
                              self._order_by, self._skip, self._limit)

    def order_by(self, *fields):
        """ Order by the fields or field expressions specified.
        To refer to the current node within a field or field expression,
        use the underscore character ``_``. For example::
            match.order_by("_.name", "max(_.a, _.b)")
        :param fields: fields or field expressions to order by
        :return: refined :class:`.NodeMatch` object
        """
        return self.__class__(self.graph, self._labels, self._conditions,
                              fields, self._skip, self._limit)

    def skip(self, amount):
        """ Skip the first `amount` nodes in the result.
        :param amount: number of nodes to skip
        :return: refined :class:`.NodeMatch` object
        """
        return self.__class__(self.graph, self._labels, self._conditions,
                              self._order_by, amount, self._limit)

    def limit(self, amount):
        """ Limit to at most `amount` nodes.
        :param amount: maximum number of nodes to return
        :return: refined :class:`.NodeMatch` object
        """
        return self.__class__(self.graph, self._labels, self._conditions,
                              self._order_by, self._skip, amount)


class NodeMatcher(object):
    """ Base matcher for selecting nodes that fulfil a specific set of
    criteria.
    :param graph: :class:`.Graph` object on which to perform matches
    """

    _match_class = NodeMatch
```

```
    def __init__(self, graph):
        self.graph = graph

    def __len__(self):
        """ Return the number of nodes matched.
        """
        return len(self.match())

    def __getitem__(self, identity):
        """ Return a node by identity.
        """
        entity = self.get(identity)
        if entity is None:
            raise KeyError("Node %d not found" % identity)
        return entity

    def get(self, identity):

        t = type(identity)
        if issubclass(t, (list, tuple, set, frozenset)):
            missing = [i for i in identity if i not in self.graph.node_cache]
            if missing:
                list(self.match().where("id(_) in %s" % cypher_repr(missing)))
            return t(self.graph.node_cache.get(i) for i in identity)
        else:
            try:
                return self.graph.node_cache[identity]
            except KeyError:
                return self.match().where("id(_) = %d" % identity).first()

    def match(self, *labels, **properties):
        """ Describe a basic node match using labels and property equality.
        :param labels: node labels to match
        :param properties: set of property keys and values to match
        :return: :class:`.NodeMatch` instance
        """
        criteria = {}
        if labels:
            criteria["labels"] = frozenset(labels)
        if properties:
            criteria["conditions"] = tuple(_property_conditions(properties))
        return self._match_class(self.graph, **criteria)


class RelationshipMatch(object):
    """ Immutable set of relationship selection criteria.
    """

    def __init__(self, graph, nodes=None, r_type=None,
                 conditions=tuple(), order_by=tuple(), skip=None, limit=None):
        if nodes is not None and not isinstance(nodes, (Sequence, Set)):
            raise ValueError("Nodes must be supplied as a Sequence or a Set")
        self.graph = graph
        self._nodes = nodes
        self._r_type = r_type
        self._conditions = tuple(conditions)
        self._order_by = tuple(order_by)
        self._skip = skip
        self._limit = limit

    def __len__(self):
        """ Return the number of relationships matched.
        """
        return self.graph.evaluate(*self._query_and_parameters(count=True))

    def __iter__(self):
```

```
        """ Iterate through all matching relationships.
        """
        query, parameters = self._query_and_parameters()
        for record in self.graph.run(query, parameters):
            yield record[0]

    def first(self):

        return self.graph.evaluate(*self._query_and_parameters())

    def _query_and_parameters(self, count=False):


        def verify_node(n):
            if n.graph != self.graph:
                raise ValueError("Node %r does not belong to this graph" % n)
            if n.identity is None:
                raise ValueError("Node %r is not bound to a graph" % n)

        def r_type_name(r):
            try:
                return r.__name__
            except AttributeError:
                return r

        clauses = []
        parameters = {}
        if self._r_type is None:
            relationship_detail = ""
        elif is_collection(self._r_type):
            relationship_detail = ":" + "|:".join(cypher_escape(r_type_name(t))
for t in self._r_type)
        else:
            relationship_detail = ":%s" %
cypher_escape(r_type_name(self._r_type))
        if not self._nodes:
            clauses.append("MATCH (a)-[_" + relationship_detail + "]->(b)")
        elif isinstance(self._nodes, Sequence):
            if len(self._nodes) >= 1 and self._nodes[0] is not None:
                start_node = Node.cast(self._nodes[0])
                verify_node(start_node)
                clauses.append("MATCH (a) WHERE id(a) = {x}")
                parameters["x"] = start_node.identity
            if len(self._nodes) >= 2 and self._nodes[1] is not None:
                end_node = Node.cast(self._nodes[1])
                verify_node(end_node)
                clauses.append("MATCH (b) WHERE id(b) = {y}")
                parameters["y"] = end_node.identity
            if len(self._nodes) >= 3:
                raise ValueError("Node sequence cannot be longer than two")
            clauses.append("MATCH (a)-[_" + relationship_detail + "]->(b)")
        elif isinstance(self._nodes, Set):
            nodes = {node for node in self._nodes if node is not None}
            if len(nodes) >= 1:
                start_node = Node.cast(nodes.pop())
                verify_node(start_node)
                clauses.append("MATCH (a) WHERE id(a) = {x}")
                parameters["x"] = start_node.identity
            if len(nodes) >= 1:
                end_node = Node.cast(nodes.pop())
                verify_node(end_node)
                clauses.append("MATCH (b) WHERE id(b) = {y}")
                parameters["y"] = end_node.identity
            if len(nodes) >= 1:
                raise ValueError("Node set cannot be larger than two")
            clauses.append("MATCH (a)-[_" + relationship_detail + "]-(b)")
```

```
        else:
            raise ValueError("Nodes must be passed as a Sequence or a Set")
        if self._conditions:
            conditions = []
            for condition in self._conditions:
                if isinstance(condition, tuple):
                    condition, param = condition
                    parameters.update(param)
                conditions.append(condition)
            clauses.append("WHERE %s" % " AND ".join(conditions))
        if count:
            clauses.append("RETURN count(_)")
        else:
            clauses.append("RETURN _")
            if self._order_by:
                clauses.append("ORDER BY %s" % (", ".join(self._order_by)))
            if self._skip:
                clauses.append("SKIP %d" % self._skip)
            if self._limit is not None:
                clauses.append("LIMIT %d" % self._limit)
        return " ".join(clauses), parameters

    def where(self, *conditions, **properties):
        return self.__class__(self.graph,
                              nodes=self._nodes,
                              r_type=self._r_type,
                              conditions=self._conditions + conditions +
tuple(_property_conditions(properties)),
                              order_by=self._order_by,
                              skip=self._skip,
                              limit=self._limit)

    def order_by(self, *fields):
        return self.__class__(self.graph,
                              nodes=self._nodes,
                              r_type=self._r_type,
                              conditions=self._conditions,
                              order_by=fields,
                              skip=self._skip,
                              limit=self._limit)

    def skip(self, amount):
        """ Skip the first `amount` relationships in the result.
        :param amount: number of relationships to skip
        :return: refined :class:`.RelationshipMatch` object
        """
        return self.__class__(self.graph,
                              nodes=self._nodes,
                              r_type=self._r_type,
                              conditions=self._conditions,
                              order_by=self._order_by,
                              skip=amount,
                              limit=self._limit)

    def limit(self, amount):
        """ Limit to at most `amount` relationships.
        :param amount: maximum number of relationships to return
        :return: refined :class:`.RelationshipMatch` object
        """
        return self.__class__(self.graph,
                              nodes=self._nodes,
                              r_type=self._r_type,
                              conditions=self._conditions,
                              order_by=self._order_by,
                              skip=self._skip,
                              limit=amount)
```

```
class RelationshipMatcher(object):
    """ Base matcher for selecting relationships that fulfil a specific
    set of criteria.
    :param graph: :class:`.Graph` object on which to perform matches
    """

    _match_class = RelationshipMatch

    def __init__(self, graph):
        self.graph = graph
        self._all = self._match_class(self.graph)

    def __len__(self):
        """ Return the number of relationships matched.
        """
        return len(self.match())

    def __getitem__(self, identity):
        """ Return a relationship by identity.
        """
        entity = self.get(identity)
        if entity is None:
            raise KeyError("Relationship %d not found" % identity)
        return entity

    def get(self, identity):
        """ Create a new :class:`.RelationshipMatch` that filters by identity
and
        if no entity is found.
        """
        t = type(identity)
        if issubclass(t, (list, tuple, set, frozenset)):
            missing = [i for i in identity if i not in
self.graph.relationship_cache]
            if missing:
                list(self.match().where("id(_) in %s" % cypher_repr(missing)))
            return t(self.graph.relationship_cache.get(i) for i in identity)
        else:
            try:
                return self.graph.relationship_cache[identity]
            except KeyError:
                return self.match().where("id(_) = %d" % identity).first()

    def match(self, nodes=None, r_type=None, **properties):
        """ Describe a basic relationship match...
        :param nodes: Sequence or Set of start and end nodes (:const:`None`
means any node);
                a Set implies a match in any direction
        :param r_type:
        :param properties: set of property keys and values
        :return: :class:`.RelationshipMatch` instance
        """
        criteria = {}
        if nodes is not None:
            criteria["nodes"] = nodes
        if r_type is not None:
            criteria["r_type"] = r_type
        if properties:
            criteria["conditions"] = tuple(_property_conditions(properties))
        return self._match_class(self.graph, **criteria)
```

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ
ФАКУЛЬТЕТ АВТОМАТИЗАЦІЇ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Автоматизація тестування програмного забезпечення на основі машинного навчання

Виконав: Мацієвський О.О.
КНм-22
Науковий керівник: к.т.н. доц. Горда О.В.

Київ- 2023

Актуальність дослідження

Автоматизація тестування програмного забезпечення на основі машинного навчання є важливим напрямком досліджень, оскільки вона вирішує ряд сучасних викликів у сфері розробки програмного забезпечення, та визначається кількома ключовими факторами:

- Зростання складності програмного забезпечення :
- Підвищення частоти випуску нових версій:
- Потреба в ефективних засобах виявлення дефектів:
- Зростання обсягів даних для тестування:
- Зростання ролі штучного інтелекту в розробці ПЗ:
- Потреба в автоматизації тестування для великих та складних проектів:

НАУКОВА НОВИЗНА

Вперше запропоновано метод, який, на відміну від існуючих рішень, дозволяє автоматично аналізувати результати тестування програмного забезпечення, що значно зменшує час, необхідний для аналізу помилок та дозволяє переглядати стан виконання тестів у реальному часі.

Наукове завдання

Розробити та вдосконалити методи тестування програмного забезпечення за допомогою машинного навчання

Мета дослідження

Зменшити час, необхідний для аналізу результатів тестування програмного забезпечення

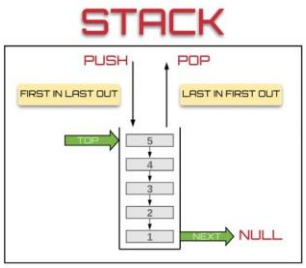
ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕНЬ

- Об’єктом дослідження є процес автоматизації тестування програмного забезпечення за допомогою методів машинного навчання.

- Предметом дослідження є алгоритми кластеризації для аналізу результатів тестування програмного забезпечення

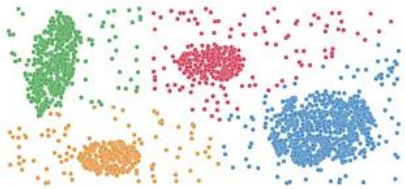
ТЕРМІНОЛОГІЯ

STACK TRACE

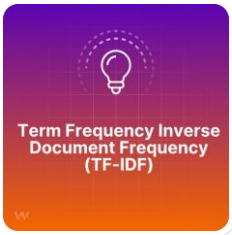


TF (TERM FREQUENCY – ЧАСТОТА СЛОВА)

КЛАСТЕРИЗАЦІЯ



IDF (INVERSE DOCUMENT FREQUENCY – ОБЕРНЕНА ЧАСТОТА ДОКУМЕНТА)

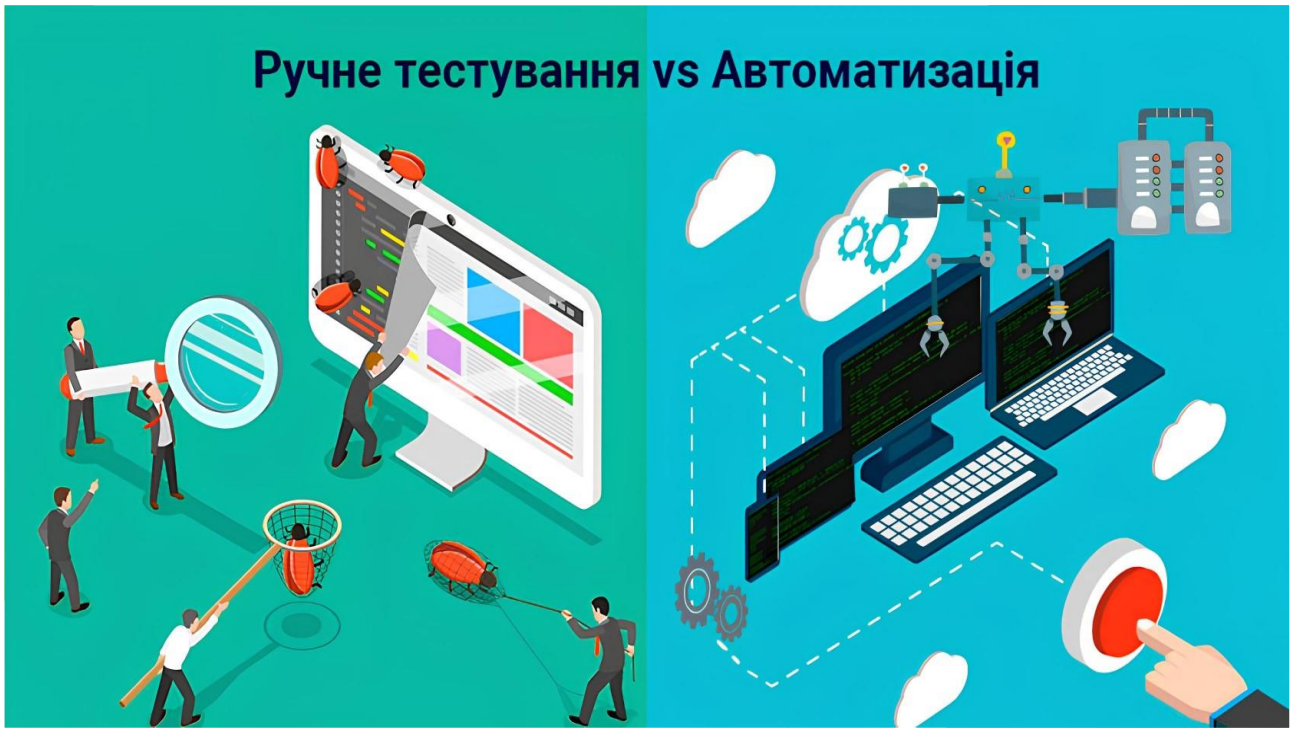


ПРОБЛЕМАТИКА

• Значні часові затрати на аналіз результатів виконання тестів

• Велика кількість людских ресурсів, необхідних для аналізу результатів тестування

• Складність оцінювання покриття тестами програмного забезпечення



Методи тестування

- **Автоматизоване тестування з використанням алгоритмів класифікації**
- **Тестування на основі глибокого навчання (Deep Learning Testing)**
- **Генерація тестових даних**
- **Виявлення помилок з використанням аналізу даних**
- **Тестування рекомендацій**
- **Тестування з використанням багатоагентних систем**
- **Тестування з використанням візуального розпізнавання**

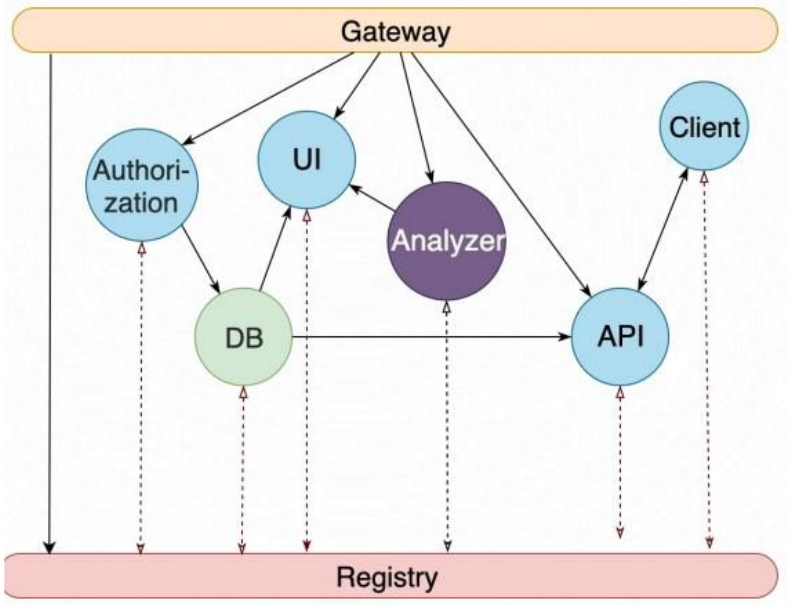
МЕТОДИ КЛАСТЕРИЗАЦІЇ

Назва методу	Простота реалізації	Точність кластеризації	Швидкість виконання
Метод Байєса	-	+/-	+
Метод опорних векторів	-	+	+
Метод k-найближчих сусідів	+	+	-

НЕДОЛІКИ ІСНУЮЧИХ АНАЛОГІВ

- Залежність від початкових умов
- Чутливість до шуму та викидів
- Проблема обрання кількості кластерів
- Низька масштабованість
- Неспроможність враховувати форму та розмір кластерів
- Неспроможність робити зміну в реальному часі
- Неможливість працювати з нелінійно роздільними даними

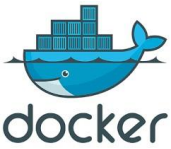
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



ВИКОРИСТАНІ ТЕХНОЛОГІЇ

- Python
- MongoDB
- Angular
- Node.Js
- Jenkins
- Docker
- Git






Висновки

Застосування машинного навчання в автоматизованому тестуванні дозволяє значно покращити ефективність та точність тестів, що призводить до швидкісного виявлення та виправлення помилок. Адаптивність до змін в коді, виявлення складних дефектів та автоматизація валідації великих обсягів даних є ключовими перевагами, які вносить машинне навчання.

Зменшення ручних трудовитрат і вартості тестування, а також можливість автоматичного виявлення та пристосування до змін роблять цей підхід ефективним і економічно доцільним. У цілому, автоматизація тестування на основі машинного навчання є потужним інструментом для забезпечення високої якості програмного забезпечення та прискорення процесу розробки.


posydenky.com.ua
Дякую за увагу!