Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

«Інтелектуальні методи автоматизації тестування вебзастосунків»
(тема)

Виконав:
студент 2 курсу, групи СШІм-18-1
Трет'якова М.С.
(прізвище, ініціали)
Спеціальність 122 – Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо -наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва освітньої програми)
Керівник доц. Вітько О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо -наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Трет'яковій Марині Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальні методи автоматизації тестування вебзастосунків
затверджена наказом по університету від 04 листопада 2019 р. № 1623Ст
2. Термін подання студентом роботи до екзаменаційної комісії 18 грудня 2019 р.
3. Вихідні дані до роботи досліджені алгоритми класифікації тексту, розроблена програма для інтелектуального тестування вебзастосунків, постановка задачі та технічне завдання, навчальна література, алгоритми машинного навчання, технології тестування програмного забезпечення
4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної області та постановка задачі; тестування програмного забезпечення; машинне навчання як метод оптимізації тестування програмного забезпечення; аналіз вибору мови програмування, середовища розробки та інструментів автоматизації; дослідження методів автоматичної класифікації тексту для застосування їх у тестуванні вебзастосунків; програмна реалізація алгоритмів; реалізація тестових класів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _Рисунок 1 – Приклад максимального запасу, що_ розділяє два класи точок даних; Рисунок 2 – Навчання класифікатора для вебсторінок;__ Рисунок 3 – Середній показник мікро-вимірювання з показаним інтервалом 95%;_____ Рисунок 4 – Середній макро-показник F-вимірювання; Рисунок 5 – Середня точність____ класифікації; Рисунок 6 – Сформований клас RegistrationPageObjects; Рисунок 7 – _____ Результати роботи програми, сгенеровані розробленою системою тести; результати_____ проходження тестів. _____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	доц. Вітько О.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Опис об'єкту автоматизації	04.11.2019	
2	Огляд сучасного стану задачі, аналіз предметної	04.11.2019 – 06.11.2019	
3	Формування вимог до задачі в цілому, вимог до функціональної структури, вимог до практичної частини	04.11.2019 – 07.11.2019	
4	Опис постановки задачі	05.11.2019 – 06.11.2019	
5	Розробка та аналіз інтелектуальних методів	06.11.2019 – 15.11.2019	
6	Розробка алгоритму вирішення задачі	12.11.2019 – 17.11.2019	
7	Розробка елементів програмного забезпечення	17.11.2019 – 20.11.2019	
8	Вибір та обґрунтування комплексу технічних засобів	21.11.2019 – 23.11.2019	
9	Розробка рекомендацій щодо використання системи	24.11.2019 – 30.11.2019	
10	Опис рішень із захисту інформації	01.12.2019 – 11.12.2019	
11	Оформлення пояснювальної записки	05.12.2019 – 13.12.2019	
12	Захист дипломної роботи	18.12.2019	

Дата видачі завдання ____4 листопада_____2019 р.

Студент _____
(підпис)

Керівник роботи _____ ____доц. Вітько О.В._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Записка пояснювальна: 111 с., 7 рис., 5 табл., 1 дод., 38 джерел.

АВТОМАТИЗАЦІЯ, ВЕБЗАСТОСУНОК, ГАРАНТІЯ ЯКОСТІ, ІНТЕЛЕКТУАЛЬНІ МЕТОДИ, КЛАСИФІКАЦІЯ ТЕКСТУ, МАШИННЕ НАВЧАННЯ, ТЕСТУВАННЯ

Об’єктом дослідження є інтелектуальні методи та алгоритми класифікації текстів для застосування їх у автоматизації програмного забезпечення. Останнім часом штучний інтелект все більше зустрічається у нашому житті. Відділ забезпечення якості не є винятком. Тестувальники прагнуть максимально автоматизувати свій робочий процес. Створюються автоматизовані тести, які майже покривають все те, що робиться при ручному тестуванні. Але з розширенням системи чи програмного продукту кількість необхідних тестів зростає, як і час на їх створення і підтримку.

Предметом дослідження стали такі важливі аспекти, як оптимізація робочого процесу, зменшення часу, витраченого на рефакторинг коду, та, звичано, зменшення використанних ресурсів. Використовуючи розроблену програму, тестувальник зможе суттєво економити час на написання нових автоматизованих тестів, час на підтримку вже існуючих тестів та аналіз результатів тестування.

Мета роботи – дослідити методи класифікації текстів для застосуваня їх у інтелектуальній автоматизації тестування вебзастосунків.

Методи дослідження – алгоритми машинного навчання для класифікації текстів, а саме класифікатор Naive Bayes, поліноміальний Naive Bayes, класифікатор TWCNB, k-найближчих сусідів (kNN) , метод опорних векторів (SVM), класифікація на основі N-грам.

РЕФЕРАТ

Пояснительная записка: 111 с., 7 рис., 5 табл., 1 прил., 38 источников.

АВТОМАТИЗАЦИЯ, ВЕБ-ПРИЛОЖЕНИЕ, ГАРАНТИЯ КАЧЕСТВА, ИНТЕЛЛЕКТУАЛЬНЫЕ МЕТОДЫ, КЛАССИФИКАЦИЯ ТЕКСТА, МАШИННОЕ ОБУЧЕНИЕ, ТЕСТИРОВАНИЕ

Объектом исследования являются интеллектуальные методы и алгоритмы классификации текстов для применения их в автоматизации программного обеспечения. В последнее время искусственный интеллект все больше встречается в нашей жизни. Отдел качества не является исключением. Тестировщики стремятся максимально автоматизировать свой рабочий процесс. Создаются автоматизированные тесты, которые почти покрывают все то, что делается при ручном тестировании. Но с расширением системы или программного продукта количество необходимых тестов растет, как и время на их создание и поддержку.

Предметом исследования стали такие важные аспекты, как оптимизация рабочего процесса, уменьшение времени, затраченного на рефакторинг кода, и, конечно, уменьшение использования ресурсов. Используя разработанную программу, тестировщик сможет существенно экономить время на написание новых автоматизированных тестов, время на поддержку уже существующих тестов и анализ результатов.

Цель работы – исследовать методы класификации текстов для применения их в интеллектуальной автоматизации веб-приложений.

Методы исследований – алгоритмы машинного обучения для классификации текстов, а именно классификатор Naive Bayes, полиномиальный Naive Bayes, классификатор TWCNB, k-ближайших соседей (kNN) , метод опорных векторов (SVM), классификация на основе N-грам.

ABSTRACT

Explanatory note: 111 p., 7 fig., 5 tabl., 1 ann., 38 sources.

AUTOMATION, INTELLIGENT METHODS, MACHINE LEARNING, QUALITY ASSURANCE, TESTING, TEXT CLASSIFICATION, WEB-APPLICATION

The object of the study is the methods and algorithms of artificial intelligence and machine learning for their application in software automation. Recently, artificial intelligence is increasingly found in our lives. The quality assurance department is no exception. Testers seek to automate their workflow as much as possible. Automated tests are created that almost cover all that is done with manual testing. But with the expansion of a system or software product, the number of required tests is increasing, as is the time for their creation and support.

The subject of the study were such important aspects as optimization of the work process, reducing the time spent on refactoring code, and, as a rule, reducing the use of resources. Using the developed program, the test can significantly save time writing new automated tests, time to support existing tests and analysis of test results.

The purpose of the project is to study the algorithms of artificial intelligence to apply them in the automation testing of forms of web applications.

Methods – machine learning algorithms for text classification – Naive Bayes classificator, Multinomial Naive Bayes, TWCNB, k-nearest-neighbour (kNN), Support Vector Machine(SVM), N-Gram based classification.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

AI – Artificial Intelligence – штучний інтелект;

API – програмний інтерфейс програми;

AUT – Application Under Test – це особливий випадок програмної системи – програма, яка тестується в даний момент;

JVM – Java Virtual Machine;

ML – Machine Learning – машинне навчання;

NB – Naïve Bayes – наївний метод Баєса;

NLP – Natural Language Processing – опрацювання природної мови;

QA – Quality Assurance – забезпечення якості, сукупність заходів, що охоплюють всі технологічні етапи розробки, випуску та експлуатації програмного забезпечення, інформаційних систем;

SUT – System Under Test – відноситься до системи, яка перевіряється на правильність роботи;

TF-IDF – Term Frequency - Iinverse Document Frequency – це статистичний показник, що використовується для оцінки важливості слів у контексті документа;

Автоматизоване тестування – тестування з використанням програмних засобів для виконання тестів, що допомагає скоротити час та полегшити процес виявлення помилок;

Баг – помилка, некоректна робота програми або не функціонування її зовсім;

Ручне тестування – це процес тестування шляхом ручної перевірки на помилки інформаційної системи, програмного забезпечення;

Тест кейс – тестовий випадок.

ВСТУП

Тестування і контроль якості (QA) швидко змінюваних систем – поширена, але непроста задача програмної розробки. Оптимальним підходом в цьому випадку є автоматизація тестування. Але в той час як близько 60% комерційних проектів використовують в своїй роботі Agile-розробку, тільки 16% автоматизують процес контролю якості. В одній з публікацій інтернет-видання ReadItQick проведений аналіз вказує, що провідну роль в процесі автоматизації розробки і тестування грають сучасні дослідження в області штучного інтелекту і машинного навчання.

Згідно з Світовим звітом про якість, тестування має занадто великі витрати в парадигмі Agile-розробки. Процес розробки в багатьох компаніях сильно гальмується через відставання на етапі тестування і забезпечення якості. Застосування AI і машинного навчання дозволяє усувати людські помилки і підвищити прозорість усіх етапів створення ПО. Такий підхід вже зараз допомагає підвищити рівень автоматизації і конкурентоспроможності найбільш передових підприємств. При цьому штучний інтелект проводить тестування не тільки швидше, але і якісніше. Контрольовані AI системи на практиці показують більш високий відсоток якості тестових кейсів. А QA-заходи під керівництвом штучного інтелекту не тільки допомагають економити час і гроші, але також ефективніше вирішують проблему масштабованості процесу оцінки якості. Якщо говорити просто, то автоматизація це кращий варіант для проведення простих повторюваних тестів. При цьому AI-тестування краще виявляє і пророкує типові проблеми, виявлені в ході такого аналізу.

Як можна застосувати штучний інтелект для тестування програмного забезпечення? Він може бути застосований у великому програмному забезпеченні з багатьма діями та можливими потоками, такими як вебзастосунки, корпоративне програмне забезпечення тощо. Насправді, на тестування всіх можливих сценаріїв за допомогою традиційного підходу

витрачається велика кількість часу та зусиль. Таким чином, побудова моделі машинного навчання є цікавим підходом до вирішення цих проблем.

Кожен раз, коли розробники модифікують код, або додають нові функції, тестування стає не тільки трудомістким, але й надзвичайно дорогим. Автоматизація допомагає зберегти цей час шляхом написання автоматизованих тестів, які в подальшому можна запускати одним кліком. Це безумовно економить час та кошти. Але це все ж не вбереже від зміни коду тестів, якщо розробники зроблять будь-які зміни до системи або програмного продукту. В даному випадку на допомогу приходить автоматизація з впровадженням штучного інтелекту.

Останнім часом все більше розвивається впровадження штучного інтелекту у тестування. Крім того, деякі компанії, як Appdiff, намагаються включити AI в тестування програмного забезпечення з мобільними застосунками, але подібне мислення можна розумно розширити на вебзастосунки. Автоматизація тестування за допомогою AI готова грати все більш важливу роль у майбутньому автоматизованого тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Тестування програмного забезпечення

Тестування програмного забезпечення – це процес дослідження ПЗ з метою отримання інформації про якість продукту. Також це процес перевірки на відповідність вимогам до продукту та реально реалізованої функціональності, що здійснюється шляхом спостереження за її роботою у штучно створених умовах та на певному обмеженому наборі тестів. У більш широкому сенсі, тестування – одна з технологій контролю якості (Quality Control), яка містить в собі планування роботи (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) та аналіз отриманих результатів (Test Analysis) [1].

Цілі тестування:

- підвищити ймовірність того, що розроблений програмний продукт буде правильно працювати при будь-яких обставинах;
- підвищити ймовірність того, що розроблений програмний продукт буде відповідати усім описаним вимогам;
- надати актуальну інформацію про стан продукту на даний момент.

1.1.1 Роль тестування у процесі розробки програмного продукту

Людські фактори можуть призводити до появи помилок на усіх стадіях розробки програмного продукту. Ці помилки можуть нести різні наслідки – від незначних до катастрофічних. Прикладом незначної помилки може бути описка, форматування, тощо, а до критичних можна віднести повне не функціонування системи, що було викликано, наприклад, невеликим відхиленням від головного сценарію користувача, або навпаки, деякі дії, які майбутній користувач обов’язково буде робити, можуть призвести до повного «падіння» системи або продукту.

Тестування допомагає виявляти та виправляти помилки, тим самим знижуючи рівень ризику та підвищуючи якість продукту. Перевіряються обов’язково ті місця інтерфейсу користувача, де він може зробити помилку або не правильно зрозуміти роботу продукту. Також система перевіряється на стійкість до хакерського втручання.

Чому процес тестування важливий?

- процес розробки програмного забезпечення неможливий без контролю якості продукту, що розроблюється;
- процес тестування програмного забезпечення являє собою таку ж невід’ємну частину процесу розробки, що й проектування;
- тестування дозволяє оцінити якість продукту, що розробляється.

1.1.2 Види тестування програмного забезпечення

Усі види тестування, в залежності від цілей, можна умовно розділити на наступні групи:

- функціональні (Functional testing);
- нефункціональні (Non-functional testing);
- пов’язані зі змінами (Regression testing).

Функціональні тести базуються на функціях та особливостях, а також на взаємодії з іншими системами та можуть бути представлені на усіх рівнях тестування: модульному або компонентному (Unit/Component testing), інтеграційному (Integration testing), системному (System testing) та приймальному (Acceptance testing). Функціональні види тестування розглядають зовнішню поведінку системи. Найбільш поширені види функціональних тестів:

- функціональне тестування (Functional testing);
- тестування безпеки (Sequrity and Access Control Testing);
- тестування взаємодії (Interoperability Testing).

Нефункціональне тестування описує тести, які необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, як система працює. Далі перераховані основні види нефункціональних тестів:

- всі види тестування продуктивності:
    a) тестування навантаження (Performance and Load Testing);
    b) стресове тестування (Stress Testing);
    c) тестування стабільності або надійності (Stability / Reliability Testing);
    d) об’ємне тестування (Volume Testing);
- тестування установки (Installation testing);
- тестування зручності користування (Usability Testing);
- тестування на відмову та відновлення (Failover and Recovery Testing);
- конфігураційне тестування (Configuration Testing).

Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинне бути протестоване знову для підтвердження того факту, що проблема була дійсно вирішена. Це і є тестування, пов’язане зі змінами, або регресійне тестування [7]. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення бага:

- димове тестування (Smoke Testing);
- регресійне тестування (Regression Testing);
- тестування збірки (Build Verification Test);
- санітарне тестування або перевірка узгодженості (Sanity Testing).

Ще тестування можна розділити за ступенем автоматизації, а саме:

- ручне тестування;
- напівавтоматизоване тестування;
- автоматизоване тестування.

Перераховані види тестування – це лише основні та найбільш поширені. Їх існує велика кількість, деякі з них застосовуються лише на специфічних проектах. У своїй роботі я буду розглядати та застосовувати на практиці саме функціональне тестування.

1.2 Функціональне тестування та автоматизація

Функціональне тестування є одним з ключових видів тестування. Основна його задача – це порівняти роботу розробленого програмного продукту з початковими функціональними вимогами замовника. Іншими словами ми імітуємо діє реального користувача щоб впевнитися, що програмний продукт функціоную правильно, згідно специфікації.

1.2.1 Основні відомості про функціональне тестування

Існує декілька видів тестування, в залежності від рівня доступу до коду:

- тестування «білого ящика» (white box testing) – розробник тестів має повний доступ до коду системи, та використовує ці знання для написання своєї тестів;
- тестування «чорного ящика» (black box) – тестувальник має доступ лише до користувацького інтерфейсу та тих компонентів системи, що і замовник або кінцевий користувач. Як правило, тестування методом «чорного ящика» виконується з використанням специфікацій або інших документів, що описують вимоги до функціонування системи. В даному виді тестування критерій покриття складається з покриття структури вхідних даних, покриття вимог та покриття моделі (у тестування на основі моделі);
- тестування «сірого ящика» (grey box) – тестувальник має доступ до коду, але при виконанні безпосередньо тестів доступ до коду, зачасту,

не потрібен. При цьому мається на увазі, що розробник тестів чітко знає внутрішні механізми роботи програми.

Тестування функціоналу, як і системне тестування або перевірка інтеграції, грають важливу роль у забезпеченні високої якості продукту. Усі типи тестування мають як свої переваги, так і недоліки. До недоліків можна віднести можливість допущення логічних помилок у програмному забезпеченні, а також ймовірність надмірного тестування.

Серед переваг варто відзначити те, що функціональне тестування ПЗ повністю імітує фактичне використання системи. Також є можливість своєчасного виявлення системних помилок ПЗ та, тим самим, запобігти багато проблем при роботі с ним надалі. Виявлення та виправлення помилок на ранньому етапі життєвого циклу ПЗ теж можна віднести до переваг.

Зараз є поширеною автоматизація функціонального тестування.

1.2.2 Автоматизація функціонального тестування

Автоматизоване тестування – аналог ручного функціонального тестування, яке виконується програмою, а не людиною. Сучасне ПЗ є складним багатофункціональним об’єктом. Саме тому ручне тестування такого продукту потребує значних трудових та часових витрат. Засоби автоматизації тестування підвищують якість та забезпечують повторне використання тестів при внесенні змін до ПЗ. Основні переваги автоматизації тестування:

- підвищення якості тестування, оскільки при використанні засобів автоматизацій «людський фактор» не впливає на якість тестування;
- прискорення процесу тестування без втрати якості, оскільки проведення того ж об’єму робіт ручним методом займає значно більше часу;
- використання засобів автоматизації для тестування дозволяє

запускати вже готові скрипти без додаткових правок;

- можливість виконання таких тестів, які не можуть бути виконані вручну або потребують значних затрат часу та обладнання;

- в ході тестування звіти з результатами створюються та зберігаються автоматично.

Автоматизоване тестування є ефективним для рутинних операцій. Наприклад, форми, в яких є велика кількість полів для набору даних (перебір даних). Тестовий процес дозволяє автоматично виконувати заповнення полів, а також після збереження здійснювати їх перевірку.

1.2.3 Коли виникає необхідність автоматизувати тести?

Відповідь на вищезазначене питання можна шукати лише після ретельного розгляду кількох факторів, що впливають на тестування. Завдяки автоматизованим інструментам тестування впроваджено кілька передових функцій автоматизації тестування, які дозволяють зробити автоматичне тестування легким, доступним за ціною та забезпечують покриття більшої кількості тестових кейсів. Оскільки тестувальники починають покладатися на автоматизовані тести, необхідно вибрати найкращий тип тестування відповідно до ваших потреб та ресурсів. Отож, щоб визначитись, чи потрібно автоматизувати той чи інший сценарій, потрібно чітко визначитись з основними питаннями [8].

Як часто ви плануєте запускати набір тестів? Якщо вам потрібно багато разів запускати тести, то автоматизація – це те, що ви повинні розглянути. Це може забезпечити вам велику віддачу від ваших інвестицій. Регресійне тестування не буде займати багато часу, бо автоматизувати тести потрібно лише один раз, а потім лише запускати їх.

Скільки тестових випадків ви розглядаєте? Цей фактор теж слід враховувати, адже є конкретна кількість тестових кейсів, які ви маєте обов’язково перевірити. Як правило, автоматизація десяти тестових

випадків не буде доброю інвестицією для автоматизації тестування, незалежно від того, як часто вони будуть запускатися. Однак, якщо кількість тест кейсів є високою, і якщо вони будуть виконуватися кілька разів, автоматизація може забезпечити кращі результати. Кількість тест кейсів повинна виправдати кількість інструментів автоматизації та затрати на неї [37].

Чи є ця функція пріоритетною? Деякі функції мають більші шанси на збій, ніж інші. Такий високопріоритетний компонент краще перевірити за допомогою інструментів автоматизації, оскільки з ручним тестуванням, оскільки є шанс, що тестовий випадок може бути пропущений під час циклу тестування. Автоматизовані тести будуть виконуватися кожного разу аналогічним чином та у ідентичних умовах, тож можна уникнути шансі на помилку, що пов’язана з людським фактором.

Чи потрібно виконувати тест кейси у визначеному порядку? Є такі тести, які можуть або повинні виконуватися після інших конкретних тестів. Дотримання усіх вимог може бути громіздким та трудомістким для ручного тестування. Але при автоматичному тестуванні скрипти можуть бути синхронізовані та виконуватися відповідно до наших вимог.

Чи планується запускати тести у декілька потоків? Бувають випадки, коли тестування вимагає одночасно виконувати один і той же набір тест кейсів на декількох машинах. За допомогою ручного тестування цього зробити майже неможливо. Однак за допомогою автоматизації ви можете налаштувати сценарії так, що тести виконуються одночасно в декількох системах.

Чи потрібно перевіряти одну функціональність з кількома наборами даних? Чи потрібно запускати один і той же тестовий сценарій з різними даними? Тоді без автоматизації не обійтися. Адже це значна економія часу, варто лише написати один раз сценарій та створити файл з тестовими даними, налаштувати їх взаємодію і все – не потрібно вручну проходити один і той же тест кейс з різними даними, це все зробить за вас система.

Це далеко не весь перелік випадків, коли необхідно автоматизувати, але тут були вміщені основні. Наведу приклад з власного досвіду. Ми розробляємо величезний проект, з великою кількістю вебформ та надскладною бізнес-логікою, проте не маємо достатньо ресурсів (тестувальників) для охоплення і тестування усіх випадків. Отож ми створили автоматизовані тести (їх кількість вже перейшла межу в 7 тисяч).

Оскільки маленькі зміни, внесені розробником, можуть привести до великих проблем, тести запускаються щоночі. Згадуємо, що їх в нас понад 7 тисяч, тож проходитимуть вони приблизно за 24-30 годин. Звичайно, це нас не влаштовувало, тож ми вирішили запускати їх у 8 потоків. Витратили не мало часу на визначення того, як краще розділити тести у потоки, щоб вони не заважали один одному, але тепер увесь скоуп тестів проходить не більше, ніж за 4 години. Тож на ранок ми вже маємо актуальні результати тестування, та бачимо загальну картину, в якому стані зараз наш проект. Тепер для підтримки автоматизованих тестів вистачає 2 тестувальників. Звичайно, є ще в команді люди, які виконують ручне тестування, адже є такі сценарії, які не можна автоматизувати через складну бізнес-логіку. Таким чином, автоматизація суттєво зменшила ресурсовитрати, а це дуже важливо, особливо для великих проектів.

1.2.4 Роль автоматизації у функціональному тестуванні

«Тестування має бути обов’язковою частиною розробки програмного забезпечення, а не окремим етапом. Коли цей підхід застосовується, якість продукту належить всій команді» – Джеймс Сівак.

Кілька років тому компанія, яка продає акції за 3,3 мільярда доларів на день, зазнала збитків у розмірі 440 мільйонів доларів США через некоректне оновлення програмного забезпечення. У сьогоденній цифровій економіці дуже важливим є якісне тестування програмного забезпечення

перед його виходом. Будь-яка помилка може не тільки призвести до великих витрат, але також може завдати серйозної шкоди репутації компанії, бренду та бізнесу. Такий негативний випадок також може спричинити міграцію користувачів.

Автоматизоване тестування – це техніка, яка автоматизує процес тестування за допомогою сценаріїв та відповідного програмного забезпечення. Після того як скрипти будуть готові, вони використовуються для автоматичного виконання тестових випадків, щоб поліпшити загальне тестування та ефективність. Компанії, які бажають підвищити продуктивність тестування, заощадити час та ресурсовитрати, запроваджують автоматизацію. Хоча автоматичне тестування не є чарівною палицею, яка може повністю замінити ручне тестування, однак, автоматичне функціональне тестування може суттєво поліпшити час роботи, якість та стабільність програми [9].

Функціональне тестування – це техніка, яка забезпечує функціонування програмного забезпечення та гарантує, що програмний продукт відповідає необхідним вимогам. Також необхідно забезпечити, щоб усі сценарії, включаючи граничні випадки та негативні кейси, не залишились без уваги. Функціональне тестування досить досконало перевіряє, що програма «робить».

Кожна функція, що розробляєтсья, повинна бути ретельно протестована, і цей процес повинен тривати до завершення розробки усього програмного продукту. Оскільки кінцевий користувач буде використовувати розроблений продукт наряду з іншими, то необхідно переконатися, що система витримає різні навантаження користувача. Ручне функціональне тестування може зайняти не лише багато часу, а й не виключає можливості помилки зі сторони тестувальника. Стратегія функціонального тестування повинна включати такі пункти:

- мета випробувань;
- створення проекту;

- тестова конструкція;
- автоматизація випробувань;
- тестування виконання;
- перевірка результатів.

Автоматизація полегшує виконання потужних та комплексних функціональних тестів, які допоможуть розробити якісний та стійкий програмний продукт.

1.3 Інтелектуальні методи в оптимізації тестування програмного забезпечення

1.3.1 Роль штучного інтелекту у тестуванні

Напевно, ви чули про штучний інтелект. Цей термін існує приблизно з того часу, коли Аллен Ньюел, Герберт А. Саймон і Кліф Шоу написали «Logic Theorist» у 1950-х роках. Історично склалося враження, що ми мало знаємо про штучний інтелект та автоматизоване тестування, які працюють у тандемі. Але це змінюється. Штучний інтелект грає все більшу роль в автоматизації тестування.

Автоматизація тестування з використанням штучного інтелекту є відносним поняттям для мене, але я активно досліджую його, бо хочу застосовувати знання на практиці. Щоб максимально застосовувати штучний інтелект у тестування, необхідно розглянути декілька основних питань.

Автоматизація тестування програмного забезпечення є певним обов'язковим. Кожен охоплює важливість побудови якісних тестів. Але якою є роль штучного інтелекту у тестуванні? Теоретично, є два або більше потенційних рішень, що запровадження штучного інтелекту у тестовій екосистемі.

Перше обґрунтоване використання AI фокусується на тестовому

менеджменті та створенні тестових кейсів автоматично. Це знижує рівень зусилля (LOE), з вбудованими стандартами, і забезпечує відповідність. Друге розумне використання AI фокусується на генерації тестового коду або псевдокоду автоматично, читаючи критерії прийняття користувача. Третій варіант, автоматичне створення та запуск тестів у веб- або мобільному додатку без написання будь-якого коду.

В наш час AI є скрізь – від Siri, Alexa та Google Search до Google Assistant, Slackbot та багато іншого. Кожен із цих додатків AI має певні ролі та цілі. Для того, щоб робот працював, потрібно визначити конкретну мету вашого AI – незалежно від того, чи створюються тестові випадки автоматично, створюється тестовий код, виконуючи тести чи щось інше.

Загальна концепція AI – це здатність машини розуміти навколишнє середовище та обробляти вхідні дані для виконання інтелектуальної дії, а потім навчитися автоматично вдосконалюватися. Голосова активація пошуку розроблена декілька років тому в Android Auto. Натиснувши кнопку на рульовому колесі Volkswagen GTI, щоб активувати Google Assistant необхідно сказати: «Включи Кріста Стейплтона», Google Assistant використовує AI для обробки отриманої інформації та виконання розумної дії. Через кілька секунд вже буде грати музика Кріса Стейплтона. Це підвищує безпеку для щоденного переміщення та дозволяє швидше знаходити своїх улюблених виконавців музики.

Навіть найрозумніші розробники допускають помилки, і в більшості випадків команди команда реагує, коли це вже сталося, а не запобігають. Якщо ви є тестувальником або працюєте з ним, ви знаєте, що їм подобається задавати багато запитань. Щоб створити тестові боти AI, ми повинні навчати ботів обробляти вхідні дані, ставити питання, щоб виконати інтелектуальну дію, як і Android Auto Google Assistant. Боти будуть тільки краще, коли ми постійно посилюємо алгоритми, щоб розпізнати шаблони вводу та поведінку.

1.3.2 Машинне навчання для автоматизації тестування

Головна мета, яку ми намагаємося досягти, використовуючи машинне навчання для автоматизації в тестуванні, – це динамічно створювати нові тестові випадки, засновані на взаємодії користувачів, шляхом вилучення їх даних та поведінку в програмному продукті, для якого повинні бути написані тести. Машинне навчання в автоматизації тестування може допомогти запобігти деякі із наступних, але не обмежених, випадків:

- збереження ручної праці в написанні тест кейсів;
- тест кейси є крихкими, тому, коли щось йде не так, система, швидше за все, або припиняє тестування, або пропускає деякі кроки, які можуть призвести до неправильного або невдалого результату;
- тести не перевіряються до тих пір, поки конкретний тест не буде запущений, тобто, якщо скрипт написано для перевірки кнопки «ОК», то ми не будемо знати про його існування, поки ми не запустимо тест.

З використанням машинного навчання система допоможе відновити результати тестів на льоту, застосувавши нечітку відповідність, тобто якщо об'єкт буде змінений або вилучений потім програмою, то скрипт повинен мати можливість знайти найближчий об'єкт до того, що шукав, а потім продовжити тест. Наприклад, якщо вебсервіси мають спочатку параметри «малий, середній, великий», і сценарій написаний відповідно до цього, і якщо додається інший вибір, тобто «надвеликий», то скрипт повинен мати можливість адаптуватися до цього і передбачати що зміниться так, що тест може продовжуватися без проблем.

HP Unified Function Testing є одним із відомих інструментів, доступних на ринку, що використовуються для автоматизованого тестування. Він має графічний інтерфейс, крім того, є інші інструменти, такі як бібліотеки Selenium (реалізовані декількома мовами, як Java, C ++, C #, Python тощо) та Cucumber. Такі інструменти дозволяють писати свої

сценарії та дозволяють сценаріям виконувати тестування.

Перед початком тестів система має вивчити випадки. Нам потрібно дати щось спочатку. Перед початком тренувальної частини нам довелося встановити систему, де деякі оголошення, які демонструвалися звичайним користувачам в індивідуальному вікні часу, і в цей час збирали та реєстрували журнали, які допомогли нам створити гендерний коефіцієнт та вікові групи людей, хто подивився на конкретні оголошення. Мета полягала в тому, щоб з'ясувати, яка вікова група та стать користувачів спонукали купувати щось із вебсайту після перегляду певних оголошень, які їм пропонувалися. Результати зберігалися як дані навчання, щоб тести могли бути виконані на них.

Машину навчили писати тести на основі зібраної інформації. Дані щодо тренувань оновлювалися щоразу, щоб тести могли виконуватися на основі останніх даних – на основі різних демографічних показників, а відповідні об'яви можуть бути передані потенційним клієнтам. Якщо це було зроблено вручну, можливо, доведеться внести зміни в скрипт або додавати тестові випадки вручну, коли зміниться тенденція або зміниться вебсайт.

1.4 Постановка задачі

Темою атестаційної роботи є «Інтелектуальні методи автоматизації тестування вебзастосунків». Тема автоматизації тестування була обрана не випадково, адже зараз я працюю саме в цій сфері. Тестування є невід’ємною частиною розробки програмного забезпечення. За допомогою автоматизації можна суттєво зекономити час на тестування повторюваних сценаріїв чи регресії. Але як зекономити час на написання самих автоматизованих тестів? Беручи до уваги те, що штучний інтелект розвивається дуже стрімко з кожним роком, то, звичайно, він допоможе і в написанні тестів [1]. Отож я маю на меті розробити програму, яка буде

складатися з наступних кроків:

- на вхід подаємо посилання на вебсторінку;
- виконуємо аналіз тексту (коду сторінки) для визначення типу вебформи;
- в залежності від виявленого типу форми формуємо Page Object цієї сторінки;
- запрограмована система автоматично тестує форму на функціональність, виконуються як позитивні, так і негативні тести;
- далі формуються тест кейси та в результаті ми маємо готовий набір тестів, який вже можна запускати та підтримувати.

В ідеалі – створити таку програму з використанням штучного інтелекту, яка буде сама класифакувати задану вебформу та далі автоматично тестувати її згідно за стратегією тестування тієї чи іншої форми, та в результаті видавати готовий набір тестів, які потім будуть запускатися за допомогою .bat файлу або у будь-якій системі CI/CD. Звичайно, це ще не буде повністю автоматизований процес, оснований на штучному інтелекті, але все ж таки розроблена система допоможе зберегти час на написання тестів, особливо в тих випадках, коли до програмного продукту, що тестується, вносяться зміни, додається новий функціонал тощо.

Для початку необхідно буде проаналізувати існуючі інтелектуальні методи класифікації тексту, які допоможуть реалізувати перший крок тестування – визначення типу сторінки. Для експериментів та роботи було поділено сторінки на таки типи: форма реєстрації, форма авторизації, формаа замовлення, форма інтернет оплати, форма оформлення доставки, форма відгука та інші форми. Отож на першому етапі буде визначено, до якої категорії належить задана сторінка.

Далі необхідно буде розробити унікальні алгоритми для кожного типу сторінки. Для початку це буде реєстраційна сторінка. Чому розробка почнеться саме реєстраційних форм? Саме тому, що саме зі сторінки

реєстрації починається робота користувача з системою. У більшості випадків, без проходження цього етапу, робота з системою неможлива для користувача. Так, звичайно, можна тестувати одразу функціонал, адже для тестування завжди є тестовий аккаунт, який створений незалежно від того, чи працює сторінка реєстрації. Зараз моя система буде направлена саме на реєстраційні сторінки – вони мають специфічні поля, типи полів, тощо.

Далі планується впровадження алгоритмів для усіх видів сторінок. Для цього необхідно буде створити складну систему та впровадити в неї машинне навчання, навчити її розрізняти різні типи полів, аназізувати зміни і з часом вона зможе не просто визначити, що за сторінка з вже відомих, а й проаналізувати та протестувати раніше невідому сторінку. Також, у майбутньому, можна навчити систему розрізняти «добрі» помилки від «поганих». До добрих відносятся коректні зміни в системі, розширення функціоналу, додавання нових полів чи сторінок. До поганих відносяться ті ситуації, коли програмний продукт працює не правильно, коли результат роботи програми не співпадає з очікуваннями, з вимогами. Тобто, те, що не відповідає специфікації – це баг.

Тестові дані для навчання були взяті з відкритих репозиторіїв на Github. Для тестування автоматизованих аглоритмів для реєстраційних сторінок була створена тестова сторінка. Вона була розроблена саме тому що, буде набагато легше тестувати розроблену програму незліченну кількість разів не завдаючи шкоди існуючим сервісам з реєстрацією. Тестовий сайт написаний на .NET та розгорнутий локально на http://localhost:2333.

Для формування Page Object використовується патерн «PageObject Generator» від Selenium. Тестую я у Хромі, тож буду використовувати відповідний вебдрайвер – ChromeDriver.

Основна розробка проходить на мові Java у середовищі IntelliJ IDEA.

2 АНАЛІЗ ВИБОРУ МОВИ ПРОГРАМУВАННЯ, СЕРЕДОВИЩА РОЗРОБКИ ТА ІНСТРУМЕНТІВ АВТОМАТИЗАЦІЇ

2.1 Розробка на мові Java

Основна розробка проходить на мові Java у середовищі IntelliJ IDEA. Java є найпоширенішою мовою програмування у світі. Великі організації у державному та приватному секторах мають величезні бази Java-коду і в значній мірі залежать від JVM як обчислювального середовища [2]. Зокрема, більша частина великого стека даних з відкритим кодом написана для JVM.

Java, мабуть, є однією з найпопулярніших мов програмування серед розробників і використовується для створення вебпрограм, індивідуального програмного забезпечення та вебпорталів, включаючи рішення для eCommerce та m-Commerce. Для багатьох розробників мови програмування починаються і закінчуються Java.

На сьогоднішній день Java – найбільш часто використовувана платформа розробки для корпоративних систем (понад 97% настільних комп'ютерів). Але більше того, її віртуальна машина підтримує пакети та індивідуальні бізнес-додатки.

Зараз, за даними Oracle, більш ніж 3 мільярди пристроїв працюють на Java у певній формі. Більшість великих компаній використовують Java для деяких своїх функцій, а сервери Java обробляють десятки мільйонів запитів щодня. Завдяки своїй обчислювальній природі, Java особливо добре підходить для вирішення складних реалізацій бізнес-логіки. Візьмемо, наприклад, програму управління фінансами Fintech, що пропонує найкращі варіанти інвестування та portfolio management.

Клієнтська програма повинна мати змогу скасування декількох пропозицій та порівняння після аналізу вхідних даних. Все відбувається протягом декількох секунд.

Вибравши розробку програмного забезпечення на мові Java, ви зможете створювати більш складні аналітичні системи та архітектуру, без «перевантаження» вашого продукту. Вся технічна магія надійно відбудеться на бекенді вашого продукту.

2.2 Розробка штучного інтелекту на мові Java

Навіщо використовувати Java для штучного інтелекту? Оскільки доступ до даних є передумовою для побудови інтелектуальних та інженерних рішень, інструменти інтелектуальної власності повинні добре інтегруватися з цими технологіями. AI починається з даних, які ви збираєте. Саме тому штучний інтелект та машинне навчання має вирішальне значення. Правильні інструменти вирішують багато інтеграційних завдань (багато data science проектів фейляться, коли прототипи не можуть інтегруватися з виробничим стеком), і вони прискорять цифрові перетворення багатьох підприємств у світі.

Що робить Java особливою? Одне з найкращих рішень Java – Java Virtual Machine Technology [3]. Ця технологія дозволяє розробникам створювати єдину версію додатка, яка буде працювати на всіх обчислювальних платформах, що підтримують Java.

Основні сильні сторони мови програмування Java:

- ремонтопридатність;
- переносимість;
- прозорість.

Штучний інтелект тісно пов'язаний з пошуковими алгоритмами, генетичним програмуванням і використанням штучних нейронних мереж. Java у сфері штучного інтелекту може бути більш ніж просто корисною. Програмування AI в Java має безліч переваг: простота використання, легка утилізація, спрощена робота з великомасштабними проектами, полегшена візуалізація, краща взаємодія з користувачами [5]. Ще однією причиною

програмування AI в Java є включення Swing і SWT (Standard Widget Toolkit). Ці особливості роблять графіку та інтерфейси привабливими та витонченими.

Ще однією причиною використання Java в програмуванні штучного інтелекту є велика кількість інформації в Інтернеті. Якщо порівнювати з іншими мовами програмування – Java використовується в рази більше. Можна просто ввести «розробка штучного інтелекту в Java», і отримати велику кількість сторінок на вибір. Java є універсальною [4]. Вона використовується для створення багатороботних систем, сенсорних мереж та комплектів для машинного навчання.

2.3 Середовище розробки. Чому Intellij IDEA краще ніж Eclipse?

З мовою програмування визначилися, тепер необхідно зрозуміти, яке середовище розробки краще використовувати Intellij IDEA чи Eclipse?

Є «вічні» питання, які не мають єдиної правильної відповіді. Наприклад, що краще: Windows або Linux, Java або C #. Одним з таких питань є якраз вибір кращої Java IDE.

Є багато суперечок щодо цієї теми, де обговорюється, яка з них має більше плагінів, клавіш і так далі. Існує так багато відмінностей, що важко вирішити, що з них найважливіше. Як результат, люди зазвичай заявляють, що обидва інтегровані середовища є однаковими у своїх можливостях, а вибір одного з них – справа смаку. Я ж хочу привести декілька об’єктивних причин, чому Intellij IDEA (як Java IDE), безумовно, краще, ніж Eclipse.

Основна відмінність між IDEA та Eclipse – IDEA відчуває контекст. Саме це мали на увазі співробітники JetBrains, коли вони звали її IDEA intelligent. Що це насправді означає? IDEA індексує весь ваш проект, аналізує все, що є, і навіть створює дерево синтаксису. Завдяки цьому, в будь-який час, де б ви не помістили курсор, IDEA знає, де ви знаходитесь і

що там можна зробити. Ця здатність розуміти контекст виражається багатьма способами, нижче будуть розглянуті деякі з них.

Процес налагодження або дебагу. У процесі налагодження ми часто хочемо оцінити деякий вираз, щоб побачити його значення. У Eclipse слід вибрати цей вираз повністю. Важливо точно вибрати цілий вираз, інакше Eclipse не зможе його зрозуміти. Далі натискаємо Ctrl + Shift + I та переглядаємо значення виразу. З IDEA ж не потрібно нічого вибрати. Можна просто помістити курсор у будь-яке місце всередині виразу і натиснути Alt + F8. IDEA розуміє, який вираз ви, напевно, потребуєте, і показує діалогове вікно, яке пропонує декілька можливих варіантів для вашого виразу. Також можна редагувати та негайно оцінити вираз у цьому ж діалоговому вікні. Дуже зручно. Виявляється, обидва IDE, в принципі, дозволяють вам робити те ж саме. Але з IDEA це набагато простіше і швидше. Я серйозно, різниця величезна – це просто небо і земля. У цьому невеликому діалоговому вікні IDEA забезпечить автозаповнення, підсвічування синтаксису та все, що вам потрібно.

Автозаповнення – те, що відрізняє будь-який IDE від блокнота. У цьому випадку відчуття контексту дає IDEA якісну перевагу. Наприклад, ми почали писати рядок коду (приклад 2.1).

```
assertElement (By.id ("errorMessage"), vi
```

Приклад 2.1 – Фрагмент коду у Java

Тепер ми хочемо знайти, які варіанти ми маємо: що може починатися з літер «vi».

Що робить IDEA? Не чекаючи натискань клавіш, вона негайно розуміє, що метод assertElement хоче, щоб екземпляр класу Condition був другим параметром, і статична змінна в класі Condition з ім'ям visible. IDEA негайно пропонує єдиний дійсний варіант:

І що робить Eclipse? На жаль, він не розуміє контексту. Він не знає, що курсор розташовано там, де повинен бути другий параметр методу assertElement. Тому, коли ви натискаєте священний Ctrl + Space, Eclipse просто показує все у Всесвіті, яке починається з букв «vi». У прекрасному вспливаючому вікні ми бачимо багато добре висвітлених добре документованих непотрібних даних.

Професійні програмісти можуть бути продуктивними, використовуючи рефакторинг, що надають їм IDE. Всі сучасні IDE пропонують дуже вражаючий набір рефакторингів. Але знову ж таки, редактори IDEA є розумними. Вони усвідомлюють, що ви хочете, і пропонує різні варіанти, які підходять для більшості ситуацій. Наприклад, припустимо, що у нас є метод assertErrorMessageIsHidden (приклад 2.2).

```
    public void assertErrorMessageIsHidden () {
      assertElement      (By.id      ("errorMessage"),
Condition.visible);
    }
```

Приклад 2.2 – Метод assertErrorMessageIsHidden

Ми хочемо, щоб строкова величина «errorMessage» могла прийти як параметр для цього методу.

Почнемо з IDEA. Помістимо курсор на будь-яке місце всередині рядка «errorMessage», далі натискаємо Ctrl + Alt + P (тобто «параметр»), і IDEA пропонує, який вираз ми, можливо, можемо отримати для параметра. Як тільки буде виділено вираз «errorMessage», IDEA пропонує кілька можливих назв для цього параметра. Ви будете багато разів здивовані тим, наскільки розумно поводиться IDEA, пропонуючи параметри для імені чогось. Вона враховує ім'я методу, тип змінної та навіть значення, а також назви таких змінних в інших місцях, а також ті імена, які ви дали таким змінним раніше.

І тепер подивімося, що пропонує Eclipse. Не забувайте: виділіть весь вираз «errorMessage» (це треба робити завжди, бо отримаєте повідомлення «Вираз повинен бути вибраний повністю, щоб активувати цей рефакторинг»), далі виберіть рефакторинг «Введення параметру» (через меню, бо немає гарячої клавіші) і отримати приблизно один і той же результат. Тим не менш, Eclipse не пропонує варіанти для назви параметра, але дякую за це.

Отож можна зробити висновок, що якщо ми обираємо Java IDE, то Intellij IDEA, безумовно, краще, ніж Eclipse. Це не просто питання смаку. IDEA об'єктивно краще. Це дозволяє вам швидко та легко писати і змінювати код, вона пропонує відповідні імена, знаходить відповідні методи, простіше робити рефакторинг. Це не вимагає від вас точно обрати вирази, але здогадується, що ви хочете зробити, і як ви хочете назвати це. IDEA передбачає і пропонує.

2.4 Автоматизація за допомогою Selenium WebDriver

За призначенням Selenium WebDriver – це драйвер браузера, тобто програмна бібліотека, яка дозволяє розробляти програми, що керують поведінкою браузера.

За своєю сутністю Selenium WebDriver є:

- специфікацією програмного інтерфейсу для управління браузером;
- референсними реалізаціями цього інтерфейсу для декількох браузерів;
- набором клієнтських бібліотек для цього інтерфейсу на декількох мовах програмування.

З драйвером користувачі не працюють на пряму. Вони працюють з прикладними програмами, які, за допомогою драйверів, взаємодіють з тими чи іншими пристроями. Драйвер не має призначеного для користувача інтерфейсу. Але ж іноді буває призначений для користувача

інтерфейс для налаштування драйвера? Буває. Але це інтерфейс програми для налаштувань, а не самого драйвера. Драйвер має тільки програмний інтерфейс, його призначення полягає в тому, щоб дати можливість прикладним призначеним для користувача програмам взаємодіяти з пристроєм.

Так ось, Selenium WebDriver, або просто WebDriver – це драйвер браузера, тобто програмна бібліотека, яка не має призначеного для користувача інтерфейсу та яка дозволяє різним іншим програмам взаємодіяти з браузером, управляти його поведінкою, отримувати від браузера якісь дані та змушувати браузер виконувати якісь команди. Отже, на основі цього визначення ясно, що WebDriver не має прямого відношення до тестування. Він всього лише надає автотестам доступ до браузера. На цьому його функції закінчуються. У своєму проекті я буду використовувати фреймворк тестування TestNG та інструмент для збірки Java проекту – Maven.

Найголовніша відмінність WebDriver від всіх інших драйверів полягає в тому, що це «стандартний» драйвер, а всі інші – «нестандартні». І це не проста фігура мови.

Організація W3C дійсно прийняла WebDriver за основу при розробці стандарту інтерфейсу для управління браузером. І тепер реалізація інтерфейсу WebDriver покладено на виробників браузерів, а WebDriver як незалежний драйвер, можливо, в майбутньому зникне зовсім, тому що він буде вбудований безпосередньо в браузери.

Таким чином, можна сказати, що Selenium WebDriver це взагалі не інструмент, а специфікація, документ, стандарт, що описує, який інтерфейс браузери повинні надавати назовні, щоб через цей інтерфейс можна було керувати браузером.

2.5 Фреймворк тестування TestNG

TestNG – тестовий фреймворк Java схожий на JUnit (Java) і NUnit (C#), але має нову потужну функціональність та більш простий у використанні. TestNG – це фреймворк з відкритим кодом, де NG означає Next Generation (Нове покоління). TestNG схожий на JUnit, але набагато функціональніший за нього. Він розроблений краще, ніж JUnit, особливо, якщо тестувати вкладені класи.

Фреймворк TestNG має багато переваг, але якщо розглядати у зв’язці з Selenium, основними є:

- дозволяє генерувати HTML репорти;
- анотації спрощують життя тестувальникам;
- тест-кейси можна згрупувати та пріоритезувати;
- можливо паралельне тестування;
- генеруються логи;
- можливо створювати параметризовані тести.

TestNG набрав високу популярність серед Java-розробників за короткий час, і на даний момент є одним з найбільш поширених тестових фреймворків серед Java розробників. Він використовує Java анотації для конфігурації та написання тестових методів.

TestNG написаний на Java і може бути використаний як з мовою Java, так і з іншими заснованими на Java мовами (наприклад, Groovy, Scala). У розглянутому тестовому фреймворку тести та тестові набори налаштовуються або описуються в основному через файли XML. За замовчуванням файл називається testng.xml, але розробник може дати йому будь-яке інше ім'я на свій розсуд.

TestNG дозволяє користувачам створювати конфігурації тестів через XML файли та дозволяє включати або виключати з них різні пакети, класи та методи в тестових наборах.

2.6 Інструмент для збірки проектів Maven

Maven – це інструмент для збірки Java проекту: компіляції, створення jar, створення дистрибутива програми, генерації документації. Прості проекти можна зібрати в командному рядку. Якщо збирати великі проекти з командного рядка, то команда для збірки буде дуже довгою, тому її іноді записують в bat скрипт. Але такі скрипти залежать від платформи. Для того щоб позбутися від залежності від платформи та спростити написання скрипта використовують інструменти для збірки проекту.

Для платформи Java існують два основні інструменти для збірки: Ant і Maven. Основні переваги Maven:

- незалежність від OS. Збірка проекту відбувається в будь-якій операційній системі. Файл проекту один і той же;

- управління залежностями. Рідко які проекти пишуться без використання сторонніх бібліотек (залежностей). Ці сторонні бібліотеки часто теж у свою чергу використовують бібліотеки різних версій. Maven дозволяє управляти такими складними залежностями. Що дозволяє вирішувати конфлікти версій і в разі потреби легко переходити на нові версії бібліотек;

- можлива збірка з командного рядка. Таке часто необхідно для автоматичного складання проекту на сервері (Continuous Integration);

- чудова інтеграція із середовищами розробки. Основні середовища розробки на Java легко відкривають проекти, які збираються с допомогою Maven. При цьому найчастіше проект налаштовувати не потрібно – він відразу готовий до подальшої розробки.

Як наслідок – якщо з проектом працюють в різних середовищах розробки, то Maven зручний спосіб зберігання налаштувань. Конфігураційний файл середовища розробки та файл для збірки один і той же – менше дублювання даних і відповідно помилок.

В основі проекту Maven лежить опис проекту на мові xml, але також ми можемо згенерувати опис з архетипу. Архетип визначається як оригінальний зразок або модель, з якої всі інші речі того ж роду робляться. У Maven архетип, шаблон проекту, який в поєднанні з деякими введенням даних користувачем отримуємо робочий Maven проект.

У Maven реалізована можливість змінювати настройки проекту в залежності від середовища, це можна зробити з використанням профілей. Профіль представляє собою настройки, які будуть додані до основних, або навпаки замінять їх. Тож Maven є дуже сильним інструментом для тестування і має багато переваг.

3 ЗАСТОСУВАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ ЗАДАЧ КЛАСИФІКАЦІЇ ТЕКСТУ

3.1 Машинне навчання

Існує кілька видів машинного навчання. Навчання з вчителем використовує наданий навчальний набір прикладів із правильними відповідями. На основі цього навчального набору алгоритм узагальнює, щоб правильно реагувати на всі можливі вхідні значення. Навчання без вчителя, навпаки, не покладається на надані правильні відповіді під час навчання, а натомість намагається самостійно визначити схожість між вхідними даними, так, щоб вони мали щось спільне та могли бути згруповані. Поєднання цих двох типів – це навчання з підкріпленням, де алгоритму повідомляється про те, що він допустив помилку, але не повідомляється, як виправити її. Натомість алгоритм повинен навчатися та випробувати різні можливі рішення до тих пір, поки він не зробить правильний прогноз [10].

Інші типи машинного навчання – це еволюційне навчання [10], напівавтоматичне (часткове) навчання [11] та багатозадачне навчання [12].

3.2 Обробка природньої мови

Обробка природної мови, або скорочено NLP, – це сфера інформатики та лінгвістики, що стосується взаємодії людської мови та комп'ютерів. Деякі основні програми NLP включають пошук інформації, агенти діалогу та розмови, машинний переклад [13]. NLP – це велика галузь, тому в цьому розділі буде лише коротко зосереджено увагу на поняттях, необхідних для подальшої імплементації в атестаційній роботі.

3.2.1 N-грама

N-грама – це суцільна послідовність n елементів із заданої послідовності тексту. Елементи – це, як правило, букви (символи), але можуть бути й іншими речами, наприклад, словами. Як правило, один фрагмент слова складається у набір N-грам, що перекриваються, і зазвичай прокладають слова пробілами, щоб визначити початок та закінчення слова [13]. Розглянемо, для прикладу, розбиття слова «apple» на біграми, триграми, та 4-грами (таблиця 3.1).

Таблиця 3.1 – Розбиття слова «apple» на N-грами

N	Результуюча N-грама
2	_A, AP, PP, PL, E_
3	_AP, APP, PPL, LE_, E__
4	_APP, APPL, PPLE, PLE_, LE__, E___

3.2.2 Лексичний розбір

Лексичний розбір, або токенізація, – це процес розбиття тексту на слова, що називаються токени. Токен, або лексема, – це послідовність літералів, що впорядковані відповідно до правил. Токенізація є відносно простою у таких мовах, як англійська, але є особливо складною для таких мов, як китайська, яка не має меж слова [14].

В англійській та українській мовах слова часто розділяються між собою пробілами або розділовими знаками. Однак це не завжди можна застосувати, оскільки, наприклад, словосполучення такі як «Los Angeles» та «rock 'n' roll» часто вважаються одним словом.

3.2.3 Частота слова та обернена частота документа

Розглянемо випадок, коли ми запитуємо систему про словосполучення «the car». Оскільки «the» є дуже поширеним словом в англійській мові, воно з’явиться у багатьох документах. Результат такого запиту поверне документи, які зовсім не мають відношення до «car», оскільки «the» відображається у всіх документах системи. Саме тому ми хочемо підкреслити важливість терміна «car». Частота слова (англ. term frequency) – обернена частота документа (англ. inverse document frequency), або скорочено TF-IDF, – це міра того, наскільки важливим є термін у документі [13].

Частота TF_i слова і може бути обрана як необмежена частота терміна в документі. Інші можливості включають булеві частоти та логарифмічно масштабовані частоти [13]. Проблема з частотою слова полягає в тому, що усі терміни вважалися однаково важливими. Таким чином, ми вводимо коефіцієнт, який позначай вагу цього терміна, та коефіцієнт, який знижує його значення, якщо він відображається у багатьох документах набору. Цей підхід визначає вагу w_i терміна і як:

$$w_i = tf_i \log \frac{n}{n_i}, \tag{3.1}$$

де n – це загальна кількість документів у наборі;

n_i – кількість виявленя терміна і у всьому наборі документів.

3.2.4 Стемінг

Багато природних мов є змінюємими, тобто слова, що мають один і той же корінь, можуть бути пов’язані з однією і тією ж темою [15]. При пошуку та класифікації інформації часто хочеться групувати слова з однаковим значенням до одного і того ж терміна. Прикладом цього є

терміни «cat» та «cats», які можна представити терміном «cat». У деяких алгоритмах – стемінги, які є результатами алгоритму – це не слова, які можуть бути «неправильними» з точки зору природної мови. Однак це не сприймається як недолік, більше як особливість [15]. Алгоритми стемінгу можна приблизно класифікувати як відсічення афіксів, статистичні або змішані.

Алгоритми видалення афіксів можуть бути настільки ж простими, як видалення n лексем із терміна або видалення закінчення s з множини у англійській мові. Одним із алгоритмів для видалення суфіксів є Стемінг Портера (Porter Stemming Algorithm), який дуже популярний і, можливо, може вважатися стандартом для англійських слів [15]. Він базується на виконанні набору кроків для зменшення слів до стемінгу, використовуючи правила для перетворення. Наприклад, одним з правил Стемінгу Портера є: SSES → SS, що означає наступне – якщо слово закінчується на «sses», то необхідно змінити суфікс на «ss» [16].

3.2.5 Модель bag-of-words

Модель bag-of-words – це спрощене представлення документів. Bag-of-words – це не упорядкований набір слів з ігноруванням їх точного положення [17]. Найпростіше представлення bag-of-words – це двійковий векторний термін, де кожна двійкова ознака вказує на те, чи є словникове слово в документі чи ні [13]. Наприклад, припущення лексики є V = {dog, cat, fish, iguana}, в такому випадку вектор документу, що містить лише слова «cat» та «iguana» буде виглядати наступним чином:

$$\vec{d} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}. \qquad (3.2)$$

З іншого боку, в деяких випадках більш користим, поданням для моделі bag-of-words є використання частоти слів як елементів функції. Для пошуку та класифікації інформації використовуються різні схеми зважування термінів, які підкреслюють відносну важливість терміна в контексті (наприклад, зважування TF-IDF).

3.3 Автоматична класифікація тексту

Автоматична класифікація тексту – це контрольоване навчальне завдання, визначене як заздалегідь визначені мітки категорій для нових документів на основі запропонованої ймовірності навчальним набором маркованих документів [18]. Де може використовуватися класиіфкація тексту? Насправді, майже будь де, наприклад, для категорізації сайтів за тематикою, для розпізнання емоційного забарвлення текстів, для фільтрації спаму. Зазвичай, задача класифікації тектів складається з декількох етапів: попередня обробка тексту (застосування стемінгу або токенізації), виявлення ознак з тексту (TF-IDF, N-грам та інші), вибір класифікатора.

3.3.1 Класифікатор Naive Bayes

Класифікатор Naive Bayes – це імовірнісний класифікатор, заснований на застосуванні теореми Баєса, з наївним припущенням незалежності між ознаками. Це значить, що ці ознаки відрізняються від програми до програми; у категоризації тексту характеристики, як правило, мають ваги, обчислені за допомогою TF-IDF або іншою схемою зважування. Ці класифікатори зазвичай вивчаються в машинному навчанні [18] і часто використовується, оскільки їх швидко та легко реалізувати [19].

Основна ідея класифікатора Naive Bayes полягає в тому, що ми хочемо побудувати правило d, яке визначає документ з класом, який дає найвищу ймовірність:

$$d(X_1, \dots, X_n) = \arg\max_c P(C = c | X_1 = x_1, \dots, X_n = x_n). \quad (3.3)$$

Це відоме правило рішення MAP. Суть полягає в тому, щоб зробити досить наївне припущеннящо всі ознаки X_1, ..., X_n є умовно незалежними. Правило Баєса стверджує, що:

$$P(C = c | X_1 = x_1, \dots, X_n = x_n) = \frac{P(C=c)P(X_1 = x_1, \dots, X_n = x_n | C = c)}{P(X_1 = x_1, \dots, X_n = x_n)}. \quad (3.4)$$

Застосовуючи наївного припущення про незалежність між ознаками:

$$P(C = c | X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | C = c). \quad (3.5)$$

З цього бачимо, що:

$$P(C = c | X_1 = x_1, \dots, X_n = x_n) \propto P(C = c) \prod_{i=1}^{n} P(X_i = x_i | C = c). \quad (3.6)$$

Використовуючи вищесказане, ми можемо записати правило 3.3 у такому вигляді:

$$d(X_1, \dots, X_n) = \arg\max_c P(C = c) \prod_{i=1}^{n} P(X_i = x_i | C = c). \quad (3.7)$$

У деяких випадках Naive Bayes є одним із найповільніших з класифікаторів, це через те, що зі збільшенням кількості ознак алгоритм витрачає ресурси на шум у даних замість визначення взаємозв'язку між даними.

3.3.2 Поліноміальний Naive Bayes

Класичним підходом [20] для класифікації тексту є поліноміальний наївний баєсовий (Multinomial Naive Bayes) класифікатор, який моделює розподіл слів (ознак) у документі як поліном, тобто ймовірність документа, що задається його класом, є поліноміальним розподілом [21]. Розрахункові індивідуальні ймовірності для $P(X_t = x_t \mid C = c)$ дорівнює, як правило:

$$\hat{P}(X_t = x_t|C = c) = \frac{N_{ct}+\alpha}{N_c+\alpha|V|}, \tag{3.8}$$

де |V| – розмір словника;

N_{ct} – кількість разів, коли функція t з'являється у класі c у навчальній виборці;

N_c – загальна кількість ознак класу c.

Константа α – це константа згладжування, яка використовується для вирішення проблем перенавчання та крайній випадок, де $N_{ct} = 0$. Встановлення $\alpha = 1$ відоме як згладжування Лапласа [20], [21]. Застосувавши ці розрахунки до формули 3.7, ми отримаємо:

$$\begin{aligned} d(X_1, \dots, X_n) &= \arg\max_c P(C = c) \prod_{i=1}^{n} \hat{P}(X_i = x_i|C = c) \\ &= \arg\max_c P(C = c) \prod_{i=1}^{n} \frac{N_{ci}+\alpha}{N_c+\alpha|V|}. \end{aligned} \tag{3.9}$$

У літературі частіше використовується правило класифікації мінімальних помилок [19], [22]:

$$\begin{aligned} d(X_1, \dots, X_n) &= \arg\max_c \left[\log P(C = c) + \sum_{i=1}^{n} f_i \hat{P}(X_i = x_i|C = c)\right] \\ &= \arg\max_c \left[\log P(C = c) + \sum_{i=1}^{n} f_i \frac{N_{ci}+\alpha}{N_c+\alpha|V|}\right], \end{aligned} \tag{3.10}$$

де f_i – частота слова x_i.

3.3.3 Класифікатор TWCNB

Трансформований ваго-нормалізований комплекс наївного баєса (Transformed Weight-Normalized Complement Naive Bayes) – це один з варіантів класифікатора MNB, який, на думку оригінальних авторів, вирішує багато проблеми класифікатора, при цьому не роблячи його повільніше або значно складніше у впровадженні [19].

Незважаючи на схожість з MNB, одна з відмінностей полягає в тому, що нормалізація TF-IDF перетворення є частиною визначення алгоритму. Основною відмінністю між ними є те, що TWCNB оцінює умовну функцію ймовірності, використовуючи дані всіх класів [19], [22]. Розрахункова ймовірність визначається як [19]:

$$\hat{\theta}_{ic} = \frac{\alpha + \sum_{k=1}^{|C|} d_{ik}}{\alpha|V| + \sum_{k=1}^{|C|}\sum_{x=1}^{|V|} d_{xk}}, k \neq c \wedge k \in C, \quad (3.11)$$

де |V| – розмір словника;

d_{ik}, d_{xk} – TF-IDF ваги слів n та x у класі k.

Тепер ми застосуємо цю оцінку як нормалізовану вагу слів [19]:

$$w_{ci} = \frac{\log \hat{\theta}_{ic}}{\sum_k \log \hat{\theta}_{ic}}. \quad (3.12)$$

3.3.4 Метод k-найближчих сусідів (kNN)

Метод k-найближчих сусідів (kNN) – добре відомий статистичний підхід для класифікації, який широко вивчався протягом багатьох років, і застосовувався до завдать класифікації тексту [18]. У задачі класифікації тексту, компонентами векторного елемента зазвичай є ваги терма (слова), як і у класифікаторі NB.

Алгоритм досить простий. Він визначає категорію тестового

документа t на основі голосування набору k документів, які ближче всього до t – це відстань між термами, як правило, Евклідова відстань [23]. У деяких додатках Евклідова відстань може почати втрачати сенс, якщо кількість ознак завелика і, як наслідок, знижується точність класифікатора. Однак kNN як і раніше вважається одним з найкращих. Основне правило, яке задається тестовим документом t для класифікатора kNN, є [24]:

$$d(t) = \arg\max_{c} \sum_{x_i \in kNN} y(x_i, c), \tag{3.13}$$

де $y(x_i, c)$ – двійкова класифікаційна функція навчального документа x_i (яка повертає значення 1, якщо x_j позначено с, або 0 в іншому випадку). Це правило відмічає за допомогою t категорію, яка містить найбільше голосів у k-найближчому сусідстві. Правило можна також розширити, ввівши функцію подібності $s(t, x_i)$, де мітки t з класом з максимальною подібністю до t [18], [24]:

$$d(t) = \arg\max_{c} \sum_{x_i \in kNN} s(t, x_i) y(x_i, c). \tag{3.14}$$

Остання зважена вирішальна ознака вважається кращою, ніж перша і є більш актуальною [24].

3.3.5 Метод опорних векторів (SVM)

Метод опорних векторів або SVM – це порівняно новий підхід до навчання, укладений Вапником у 1995 р. [25], для вирішення двох задач розпізнавання шаблонів класу. Емпіричні дані свідчать про те, що SVM – одна з найкращих методик виконання автоматичної категоризаціяї тексту [18]. Проблема SVM зазвичай вирішується за допомогою методів квадратичного програмування [26].

Метод лінійних SVM визначається у векторному просторі, де потрібно знайти проблему рішення, яка найкраще розділяє два класи точок даних. Для того щоб це зробити, необхідно ввести маржу між двома класами [29]. Аналогічно до kNN та NB класифікаторів, точки даних представляють термін (слово) ваг, коли завданням є класифікація тексту. На рисунку 3.1 показаний приклад маржі, що розділяє два класи точок даних у двовимірному просторі функцій лінійно розділених класів.

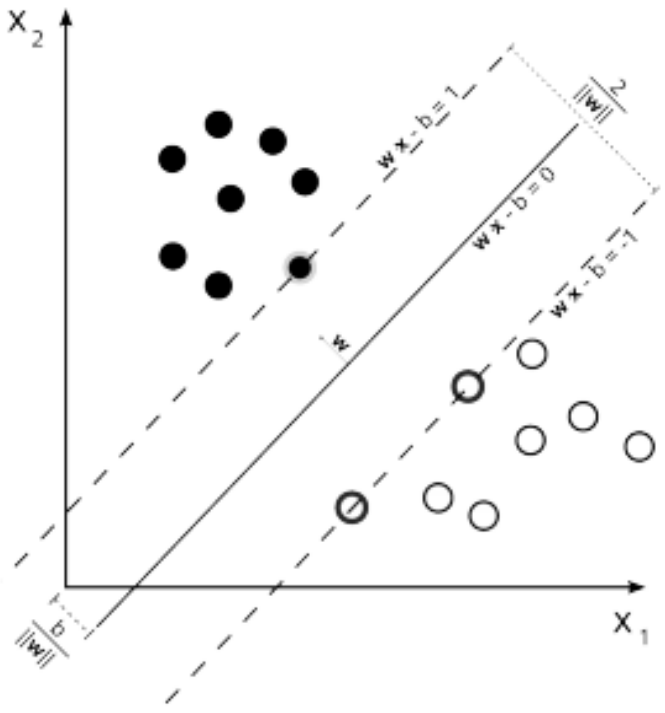


Рисунок 3.1 – Приклад максимального запасу, що розділяє два класи точок даних

Суцільна лінія є прикладом поверхонь рішення, що розділяє два класи, а штрихові лінії, паралельні суцільній, показують, яка поверхня рішення можна переміщувати, не ризикуючи викликати неправильну класифікацію точок даних. SVM вирішує проблему пошуку поверхні рішення, яка максимально збільшує запас. Поверхня рішення, як видно на рисунку 3.1, виражається як [18], [26]:

$$\vec{w} \cdot \varphi(\vec{x}) - b = 0, \tag{3.15}$$

де x – умовна точка даних, яку слід класифікувати;

φ – перетворення функції в точках даних.

Вектор w і константа b вивчаються як навчальний набір лінійно відокремлюваних даних. Нехай D = {(yi, xi)} розміром N позначає значення навчальний набір, де y ∈ {± 1} – класифікація для x (+1 є додатним приклад для даного класу, -1 негативний приклад) [18]. Проблема SVM полягає в тому, щоб знайти w і b, які задовольняють наступним двом обмеженням [26]:

$$\begin{matrix} \vec{w} \cdot \varphi(\vec{x}_i) - b \geq +1 \text{ для } y_i = +1 \\ \vec{w} \cdot \varphi(\vec{x}_i) - b \geq -1 \text{ для } y_i = -1 \end{matrix}. \tag{3.16}$$

Відповідно до рисунку 3.1, обведені точки на пунктирній лінії – це вектори підтримки. Рішення функції для немаркованого документа d, представленого функціональним вектором x_d, є:

$$d(\vec{x}_d) = \begin{cases} +1, \text{якщо } \vec{w}^T \cdot \varphi(\vec{x}_d) + b > k \\ -1, \text{в інших випадках} \end{cases}, \tag{3.17}$$

де k – визначений користувачем поріг.

Розглянемо метод нелінійних SVM. Якщо ми візьмемо довільний текстовий документ, видається, що дані в простір функцій буде лінійно відокремленим. Таким чином, ми хочемо, щоб рішення, яке прийняте, могло відокремлювати нелінійні точки даних. Можемо переформулювати функцію прийняття рішення як:

$$d(\vec{x}_d) = \begin{cases} +1, \text{якщо } \sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_d, \vec{x}_i) + b > k \\ -1, \text{в інших випадках} \end{cases}, \tag{3.18}$$

де $\alpha_i \geq 0$.

Функція K(x, xi) називається функцією ядра і дозволяє нам отримати поверхню прийняття рішень для нелінійно-відокремлюваних даних. Приклад такої функції ядра є поліном ядра K(x,xi) = (xT · xi + 1)d.

Розглянемо також багатокласовий SVM. Як описано вище, SVM є двійковим класифікатором. Однак у багатьох практичних додатків зазвичай існує декілька категорій, до яких може належати точка даних і, безумовно, це стосується нетривіальних програм класифікації документів. Існує декілька вивчених методів багатокласової класифікації [27], [28]; звичайний підхід полягає в об'єднанні декількох бінарних SVM для отримання єдиного багатокласовий SVM.

У 2005 році Дуан та Керті провели емпіричне дослідження деяких багатокласових методів. Для даної багатокласової проблеми, нехай M позначає кількість класів, а ω_i, i = {1, ..., M} позначатимемо M класи. Для двійкової класифікації ми будемо називати обидва класи як позитивні і негативні.

Перший метод, який ми розглянемо, – це так званий «one-versus-all winner takes-all» метод. Метод будує M бінарний класифікатор. Вихідна функція p_i класифікатора навчається, приймаючи приклади з ω_i як позитивні і всі інші, як негативні. Для нового документа t він присвоює йому клас з найбільшим значення p_i [27].

Інший метод – це метод «one-versus-one max-wins voting», який структурує один бінарний класифікатор для кожної пари різних класів, разом M (M -1) / 2 класифікатори. Двійковий класифікатор C_{ij} навчається, беручи приклади з ω_i як позитивний, а приклади з ω_j – як негативний. При класифікації нового документа t, всі класифікатори приймають голос за те, яким з класів ω_i , ω_j має бути призначений до t. Після того, як всі класифікатори проголосували, клас з найбільшою кількістю голосів призначається t [27].

Ще один підхід відомий як «pairwise coupling». Він працює з припущенням, що вихід кожного двійкового класифікатора слід

інтерпретувати ймовірності позитивного класу. Тоді стратегія полягає у поєднанні виходів всіх бінарних класифікаторів один проти одного, щоб отримати оцінку попередньої ймовірності $p_i = P(\omega_i|t)$. Класифікатор вибирає клас, який дає найвищий рівень p_i.

3.3.6 Класифікація на основі N-грам

У 1994 році Кавнар і Тренкле запропонували метод текстової категоризації, заснований на N-грамах [28], який використовувався для класифікації мови та тематики статтей на USENET (всесвітньо розповсюджена система Інтернет-дискусій). Здоровий глузд диктує, що людські мови незмінно мають деякі слова, які трапляються частіше, ніж інші. Закон Ципфа виражає це так «Частота n-го слова є обернено пропорційною його порядковому номеру n».

Наслідком цього закону є те, що в мові завжди є набір слів, якой переважає більшість інших слів за частотою вживання. Наприклад, в англійській мові найчастіше використовувані слова – це функціональні слова, такі як, «the», «be» та «to» [29]. Кавнар і Тренкле заявляють, що закон також передбачає, що завжди існує набір слів, частіший для конкретної галузі. Наприклад, у статті про спорт може бути багато згадок про футбол, гравців і фанатів, а у статтях про комп'ютери та технології частіше згадують слова комп’ютер, програміст тощо.

Можливий висновок на даний момент, якщо статті на певній мові або у певній галузі мають деякий набір слів більше, ніж інші, то подібні статті повинні також мати набір N-грамів, які частіше за інших. Це здається розумний висновок, оскільки експерименти показують, що використання N-грамів для мови ідентифікація є надійною та досить надійною для ідентифікації галузі [28].

Метод, запропонований Кавнаром і Тренкле, заснований на порівнянні N-грамів частоти профілів з набору навчальних виборок для

перевірки документів, а також узагальнюванні останніх на основі дистанційних заходів. Кроки для профілювання документа такі:

- токенізувати текст, відкинути цифри та розділові знаки. Накласти токени на достатню кількість заготовок до і після;
- створити усі можливі N-грами (включаючи пробіли) для кожного токену, для N = [1, 5];
- порахувати кількість випадків для кожного N-грама;
- відсортувати N-грами у зворотному порядку за порядком виникнення, тобто за найчастіший N-грам є першим елементом у відсортованому списку.

Створюючи N-грам для всіх навчальних документів та об'єднуючи профілі документів до однієї категорії, в такому випадку профіль частоти N-грам для кожної категорії зроблений. Перші 300 або близько того N-грамів вважаються дуже залежними від мови, і таким чином видаляються з профілю [28].

Класифікація вхідного документа проста: генерація N-грам частоти профіля для документа, вимірювання відстані до категорії та вибір категорії з мінімальною відстаню. Використовувана Кавнаром і Тренкле міра відстані профілю досить проста. Необхідно взяти два N-грамові профілі та обчислити просту статистику порядку ранжування (так званий «out-of-place» показник) шляхом вимірювання того, наскільки далеко розміщується N-грам в одному профілі від свого місця в іншому профілі. Для кожного N-грама в документі профілю, необхідно знайти його аналог у профілі категорії та обчислити, наскільки далеко місця з точки зору положення (рангу) в профілі. Ми можемо сформулювати out-of-place міру N-грама n у профілях i та j як

$$d_n(i,j) = \begin{cases} M & \text{, якщо } n \text{ не } \in \text{ одразу } i \text{ та } j \\ |rank(n,i) - rank(n,j)| & \text{, в інших випадках} \end{cases}, \quad (3.19)$$

де M – заздалегідь визначене максимальне out-of-place значення, яке

задається, якщо N-грам не існує в обох профілях, а rank(n, i) – це ранг N-грама n у профілі [28].

Використовуючи міру відстані в рівнянні 3.19, ми можемо побудувати рішення правило:

$$d(\mathrm{e}) = \arg\min_{c} \sum_{i \in P_t} d_i(P_t, P_c), \tag{3.20}$$

де P_c , P_t – N-грам профілі для класу c і документа t відповідно.

Експерименти показують, що цей метод дуже добре працює для класифікації мов, і досить добре для предметної класифікації, досягаючи високий 80% класифікаційний показник для останнього [28]. Кавнар і Тренкле зазначають, що вищий показник класифікації галузі, можливо, може бути досягнутий шляхом видалення дуже часто використовуваних слів мови з даних, тобто з використанням стоп списку. Що дуже цікаво, так це те, що можна досягти досить хороший показник класифікації, враховуючи, що метод є одним з менш складних у порівнянні з іншими підходами.

4 ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ У ТЕСТУВАННІ ВЕБЗАСТОСУНКІВ

4.1 Рівні тестування, основаного на штучному інтелекті

У майбутньому інструменти автономного тестування з використанням штучного інтелекту суттєво допомагатимуть, але не замінять живих людей. Крім того, тестувальникам та командам з управління якістю слід витрачати більше часу на думки про вартість бізнесу, тобто покладати більше зусиль на те, як протестувати та забезпечити якість, а не виконувати тестування автоматизації як кінцевий засіб. Я розгляну шість рівнів тестування, в залежності від рівня інтеграції у ньому штучного інтелекту.

Нульовий рівень – повна відсутність автономії. Цей рівень не дарма називається нульовим, а не першим, адже інтеграція штучного інтелекту у ньому дуже мала, або відсутня зовсім. Отож, тестувальник пише код, який тестує програму, і він щасливий, бо може запускати однакові тести щоразу, як виходить новий реліз. З одного боку – це. звичайно. добре. Але ніхто не допомагає тестувальнику писати цей код автоматизації. І самостійне написання коду повторюється знову і знову.

Додавання будь-якого поля на форму означає додавання нового тесту, або, щонайменше, нових методів. Додавання будь-якої форми до сторінки означає додавання нових тестів, що перевіряють всі поля. І додавання будь-якої сторінки означає перевірку всіх компонентів і форм на цій сторінці. І чим більше тестів маємо, тим більше вони будуть падати, коли розробники вносять глобальні зміни в програму. Тому необхідно перевіряти кожний впавший тест, щоб перевірити, чи це справжній баг, чи нові зміни в системі [38].

Перший рівень – підтримка управління. Автономний автомобіль – це гарна метафора для цього рівня тестування. Чим краще бачення

автономного автомобіля, тим автономніше він є. Так само, чим краще система штучного інтелекту може бачити застосунок, тим автономніше буде тестування.

Штучний інтелект повинен мати змогу бачити не тільки снепшот моделі об'єктів документів (DOM) сторінки, але і його візуальну картинку. DOM – це лише половина зображення – той факт, що елемент із текстом на ньому є, ще не означає, що користувач може це побачити. Інші елементи можуть накластися на сторінку, або сторінка може не бути в правильному положенні.

Візуалізація дозволяє зосередитися на даних та елементах, які бачить користувач. Після того, як система тестування зможе побачити сторінку та подивитися на неї в цілому, через DOM або скріншот, це може допомогти створити перевірки, які тестувальник будь-яким чином напише вручну. Якщо зробити скріншот сторінки, то можна перевірити всі поля форми та текст на ній одним махом. Замість того, щоб писати код, який перевіряє кожне поле, можна перевіряти всі їх одразу, порівнюючи з попередніми тестами.

Для цього рівня тестування для роботи тестових інструментів необхідні алгоритми штучного інтелекту, які дозволяють визначити, які зміни насправді є реальними, а де все ж таки з’явився баг. Технології AI сьогодні можуть допомогти у написанні коду тестів шляхом створення перевірок. На цьому рівні тестувальник все ще керує тестами, але AI може робити деякі перевірки автоматично. Крім того, AI може перевірити, що тест проходить. Але коли це не вдається, AI все ж таки повинен повідомити, щоб тестувальник міг перевірити, чи це справді баг, чи це трапилось через заплановані зміни програмного забезпечення. Тестувальник повинен підтвердити, що зміни є коректними, або, що з’явився баг.

Тепер якщо AI «бачить» програму, то він може перевіряти візуальні аспекти на відповідність еталонам – все те, що дотепер можна було

перевіряти лише шляхом ручного тестування. Але на цьому рівні все ще потрібно підтверджувати кожні нові зміни в системі.

Другий рівень – часткова автоматизація. З автономією першого рівня тестувальник може уникнути виснажливого аспекту написання перевірок для всіх полів на сторінці за допомогою тестів на еталони з використанням AI. Але перевірка кожного впавшого тесту на те, чи це «правильна» помилка, чи баг, може бути нудною, особливо якщо це повторюється в багатьох тестах. Отож на другому рівні наш штучний інтелект повинен розуміти різницю, яку вміє розуміти користувач системи. таким чином, AI повинен мати можливість групувати зміни з безлічі сторінок, оскільки вони допоможуть зрозуміти семантично, що це однакові зміни.

На цьому рівні AI може групувати ці зміни, повідомляти тестувальника, коли ці зміни однакові, та питати, що робити з ними далі – підтвердити чи відхилити всі зміни як групу. Можна зробити висновок, що на другому рівні AI допомагає перевіряти зміни в порівнянні з еталонами та перетворює повторювані та складні дії у більш прості.

Третій рівень – умовна автоматизація. На другому рівні людина все ще повинна перевіряти будь-які помилки або зміни, що були виявлені у програмному забезпеченні. Другий рівень може аналізувати зміни, але не може визначити їх правильність. Для цього знову потрібні еталони, щоб з ними порівнювати. На третьому ж рівні AI може це зробити, та навіть більше, використовуючи методи машинного навчання. Наприклад, AI може розглянути візуальні аспекти сторінки та вирішити, чи зміни базуються на основі стандартних правил дизайну, включаючи вирівнювання, використання пробілів, кольорів та шрифтів, а також розмітки.

А як щодо аспектів даних? Тут AI може перевірити дані та визначити, що це поле лише числове та в якому діапазоні повинні бути числа, чи поле є електронною поштою, чи дані в полі повинні бути сумою полів над ним. Також він може визначити, наприклад, що на певній

сторінці таблиця має бути відсортована за певним стовпцем. Тепер AI може оцінювати сторінки без втручання людини, просто розуміючи правила дизайну та дані. І навіть якщо щось змінилося на сторінці, AI може зрозуміти, що сторінка все ще залишається правильною і не треба її передавати людині для розгляду. AI розглядає сотні результатів тестів і може побачити, як змінюється ситуація з часом. І, застосувавши методи машинного навчання, система штучного інтелекту може виявляти аномалії в змінах та повідомляти лише про ті, які необхідно перевірити людиною.

Четвертий рівень – високий рівень автоматизації. До цього часу штучний інтелект лише робить перевірки автоматично, а людина все ж таки керує тестами та запускає їх (за допомогою програмного забезпечення для автоматизації). На четвертому ж рівні штучний інтелект вже буде сам керувати тестами.

Оскільки AI на цьому рівні може вивчати та розуміти сторінку так, як би це зробила людина, він розуміє та може відрізнити сторінку входу до профілю, сторінку реєстрації або сторінку кошика. Саме тому що, він розуміє сторінку семантично, як сторінку, що є частиною потоку взаємодії, AI може керувати тестами. Хоча такі сторінки, як вхід та реєстрація є стандартними, проте більшість з них не є такими. Але AI зможе розглядати взаємодію користувачів з часом, візуалізувати взаємодію та розуміти сторінки, навіть якщо вони є такими, що система AI ще ніколи не бачила. Коли AI розуміє тип сторінки, використовуючи такі методи, як навчання з підкріпленням (один із типів машинного навчання), він може автоматично розпочати тестування. Тепер штучний інтелект може писати повноцінні тести, а не лише невеликі перевірки для них.

П’ятий рівень – повна автоматизація. Поки що цей рівень є науковою фантастикою. На цьому рівні AI зможе взаємодіяти з менеджером продукту, розуміти розроблений програмний продукт та самостійно проводити тестування. Але, враховуючи те, що ще ніхто не зміг досконало та з першого разу зрозуміти опис програмного продукту від менеджера, то

штучний інтелект на п’ятому рівні має бути набагато розумніший, ніж людина.

Отже, на якому рівні зараз знаходиться тестування? Розширені інструменти знаходяться в даний час на першому рівні, але вони вже добре розвиваються і на другому рівні. І хоча інструменти працюють на другому рівні, штучний інтелект третього рівня потребує роботи, але це ще допустимо. На шляху до втілення вже четвертий рівень, тож протягом наступного десятиліття ми вже сподіваємося побачити тестування з використанням AI без неприємних побічних ефектів.

4.2 Як штучний інтелект змінить тестування програмного забезпечення

Площа поверхні для тестування програмного забезпечення ще ніколи не була такою широкою. Програми сьогодні взаємодіють з іншими програмами за допомогою інтерфейсу API, вони впливають на системи, і їх складність нелінійно зростає з кожним днем. Що це означає для тестувальників? У повідомленні з World Quality Report сказано: «Ми вважаємо, що найважливішим рішенням для подолання складності тестування та тестових завдань стане запровадження машинного інтелекту». Отож, як ми, як тестувальники, будемо використовувати AI для перевірки постійно зростаючих програм і додатків?

Як зміниться тестування, коли AI нарешті з’явиться у наших виробничих додатках? Ці питання я хочу детально розглянути на прикладі п’яти шляхів розвитку AI, які я вважаю початком змін у тестуванні програмного забезпечення [1].

Перший шлях – зміна інструментів. Джейсон Арбон є генеральним директором та засновником AppDiff, компанії, яка використовує AI для тестування мобільних додатків. Він також розробник і тестер, який працював у Google та Microsoft. Він був співавтором книги «How Google

Tests Software». Хто ж може бути кращим, ніж він, щоб коментувати, як AI вплине на тестувальників?

Другий шлях – позбавлення від детермінізму. Під час вивчення AI був момент, коли я зрозуміла, що проблеми, які ми вирішуємо з AI, не детерміновані. Якщо б вони були такими, то ми б не використовували AI для їх вирішення. Крім того, вирішення проблем, які ми намагаємося вирішити, змінюється, оскільки наші системи розширюються та містять нові дані.

Третій шлях – штучний інтелект стане вашим другом. Якщо AI змінить нашу точку зору так само, як вікна змусили сміятися дітей Арбона, можливо, наше життя, як тестувальників, стане набагато простіше. «Взаємодія AI з системою помножує результати, які ви мали б при ручному тестуванні», – говорить Джереміс Реслер. Реслер, який має докторський ступінь інформатики, провів останні три роки пряцюючи над програмою тестування на основі AI під назвою ReTest.

В даний час є бета-версія, ReTest пропонує розкіш вироблення тестів для додатків Java Swing. Якщо для генерації тест кейсів недостатньо взяти на себе зобов'язання щодо статусу кращого друга з AI, Infosys тепер має пропозицію для «забезпечення штучного інтелекту». Ідея полягає в тому, що система InfoSys використовує дані в реальних системах контролю якості (дефекти, рішення, вихідний код, тестові випадки, ведення журналу тощо), щоб допомогти виявити проблемні зони у продукті.

Висловлюючи свій погляд на AI-as-testing-assistant, спроектований Реслером та Infosys, Мілман і Кармі стверджують: «По-перше, ми побачимо тенденцію, коли люди будуть мати менш механічну ручну роботу, пов'язану з впровадженням, виконанням та аналізом результатів випробувань, але вони залишатимуться необхідною частиною процесу тестування, щоб затвердити та керувати результатами. Це вже можна побачити сьогодні в тестових продуктах на базі AI, таких як Applitools Eyes».

Четвертий шлях – ми станемо містиками. Що відбувається, коли випробувальні програми та тестові системи використовують штучний інтелект? Реслер відразу озвучив «проблему Oracle», яка була виявлена під час спроби автоматизувати процес тестування. Автоматизація може знати, як взаємодіяти з системою, але відсутня «процедура, яка відрізняє правильну та неправильну поведінку SUT». Іншими словами, як тести, основані на штучному інтелекті, знають, що SUT є правильною?

Люди роблять це шляхом комунікації з замовником, або власником продукту, або менеджером проекту. Але ж що буде джерелом правди для AI? Хоча AI може дати нам містичне уявлення про те, що робитиме система, проблема Oracle повинна бути вирішена для тестування SUTs на базі AI.

Як тестування AI вплине на нас як на тестувальників? Як зазначає Мілман і Кармі, «Тест-інженери потребують іншого набору навичок, щоб створити та підтримувати набір тестів на основі AI, які перевіряють продукти на базі AI. Вимоги до роботи включатимуть більше уваги до навичок в галузі обробки даних, і тест-інженери повинні розуміти принципи глибокого навчання (deep learning)».

П’ятий шлях – тестувальники зникнуть. Чи будуть тестери йти шляхом динозаврів? Якщо ви хочете отримати надію на тестування, треба послухати Арбона: «Я відверто не можу згадати про те, що раніше в минулому я виконував одну тестову діяльність, яку в кінцевому підсумку не можна було б зробити краще за допомогою AI з достатньою кількістю навчальних даних.

Врешті-решт це на слуху вже досить давно. Але я все ще відчуваю потребу вказувати музику, яка надихає мене на роботу». Легко закріпити нашу власну важливість у своїх ролях, що ми незамінні, тому що ми можемо зробити те чи інше. Але не робіть помилок: як астероїд, який вбив динозаврів, AI наближається.

4.3 Як AI змінює автоматизацію тестування

Перш ніж розглянути приклади автоматизації тестування, на які впливає машинне навчання, потрібно визначити, чим саме є машинне навчання насправді [40]. За своєю суттю, ML – це технологія розпізнавання шаблонів, вона використовує шаблони, ідентифіковані алгоритмами вашого машинного навчання, для прогнозування майбутніх тенденцій. ML може споживати багато складної інформації та знаходити шаблони, які є прогностичними, а потім сповіщати про ці відмінності. Ось чому ML настільки потужний. Даля я хочу розглянути п'ять сценаріїв автоматизації тестування, до яких вже застосований штучний інтелект, та як його успішно використовувати у роботі.

4.3.1 Автоматизація тестування візуального інтерфейсу користувача

Які типи об’єктів можна визначити за допомогою ML? Один із тих, що стає все більш популярним, – тестування на основі зображення, що використовує інструменти для автоматичної перевірки візуалізації. «Візуальне тестування – це діяльність із забезпечення якості, яка призначена для перевірки того, що інтерфейс користувача відображається правильно», – пояснив Адам Кармі, співзасновник та технічний директор Applitools, постачальник інструментів для розробки.

Багато людей це плутає з традиційними інструментами для функціонального тестування, які були розроблені, щоб допомогти тестувальнику перевірити функціональність програми через інтерфейс користувача. З візуальним тестуванням «ми хочемо переконатися, що інтерфейс користувача виглядає правильно для користувача і, що кожен елемент інтерфейсу відображається у правильному кольорі, формі, положенні та розмірі», – сказав Кармі. «Ми також хочемо переконатися, що він не приховує або не перекриває інші елементи інтерфейсу».

Насправді, багато що з подібних видів тестування надто складно автоматизувати, саме тому в решті решт це віддається на ручне тестування. Це робить їх ідеальними для тестування AI.

Використовуючи інструменти візуального тестування на базі ML, можна знайти відмінності, які, найімовірніше, пропустять живі тестувальники. Це вже змінило те, як я реалізовую автоматизацію тестування. Я можу створити простий тест з використанням машинного навчання, який автоматично виявляє всі візуальні помилки в моєму програмному забезпеченні. Це допомагає перевірити візуальну правильність програми без необхідності неявного твердження про те, що я хочу це перевірити.

4.3.2 Тестування API

Інша зміна з використанням ML, яка впливає на те, як ви здійснюєте автоматизацію, – це відсутність користувацького інтерфейсу для автоматизації. На сьогодні багато тестів працюють у фоновому режимі (back-end-related) замість застосування користувацького інтерфейсу.

Енжі Джонс, інженер з автоматизації в Twitter, у своєму інтерв'ю для TestTalks, «Реальність тестування в штучному світі», зазначила, що більша частина її нещодавньої роботи в значній мірі покладалася на автоматизацію тестування API, щоб допомогти їй випробувати ML. Джонс пояснила, що в своїй автоматизації тестування вона зосередила увагу на алгоритмах машинного навчання. «Так, звичайно, програмування, що мені довелося робити, але було також багато чого іншого ... Я повинна була зробити багато аналітики в рамках моїх тестових скриптів, і мені довелося зробити багато викликів API».

4.3.3 Запуск більше автоматизованих тестів, які необхідні

Скільки разів ви запускали весь набір тестів через дуже невеликі зміни у програми, яку ви не могли простежити? Не дуже стратегічно, чи не так? Чи не було б здорово, якщо б ви могли відповісти на класичне питання тестування: «Якщо я змінив цей фрагмент коду, якою буде мінімальна кількість тестів, я повинен мати можливість працювати, щоб з'ясувати, чи не привела ця зміна до поганих наслідків?». Багато компаній використовують інструменти AI, які роблять саме це. Використовуючи ML, вони можуть точно сказати, яка найменша кількість тестів необхідна для тестування частини зміненого коду.

4.3.4 Створення більш надійних автоматизованих тестів

Як часто ваші тести падають завдяки тому, що розробники вносять зміни до системи, такі як перейменування ідентифікатора поля? Зі мною це трапляється постійно. Але інструменти можуть використовувати машинне навчання, щоб автоматично пристосовуватися до цих змін. Це робить тести більш якісними та надійними. Наприклад, поточні інструменти тестування AI та ML можуть почати вивчати програму, розуміти відносини між частинами об'єктної моделі документа та дізнаватися про зміни протягом усього часу.

Коли такий інструмент починає навчання та спостерігає, як застосунок змінюється, він може автоматично приймати рішення в реальному часі, щоб визначити, які локатори він повинен використовувати для ідентифікації елемента – все це без втручання людини. Тепер, якщо програма постійно змінюється, це вже не проблема, тому що з ML сценарій може автоматично налаштувати себе.

Це було однією з головних причин, за яким Ден Белчер, співзасновник інструменту тестування Mabl, і його команда розробили

алгоритм тестування ML. У своєму нещодавньому інтерв'ю він сказав: «Хоча Selenium – це найбільш широко використовуваним, проблема з цим полягає в тому, що він досить тісно пов'язаний з конкретними елементами на інтерфейсі. Через це ви будете постійно правити скрипт автотестів, коли розробники роблять те, що виглядає як досить невинна зміна інтерфейсу. На жаль, в більшості випадків ці зміни спричиняють падіння тесту через неможливість знайти елементи, з якими вона повинна взаємодіяти. Одне з того, що ми зробили на самому початку створення Mabl, полягало в тому, щоб розробити набагато розумніший спосіб з посиланням на елементи інтерфейсу в нашій автоматизації тестування, так, що внесені зміни до цих елементів фактично не порушують ваші тести».

Отже, необхідно стати експертом з доменної моделі. Підготовка алгоритму ML вимагає наявності моделі тестування. Ця діяльність потребує когось зі знаннями домену; багато інженерів з автоматизації займаються створенням моделей, які допоможуть досягти цього розвитку. З цими змінами також існує потреба у тих, хто не тільки вміє автоматизувати, але й може аналізувати та розуміти складні структури даних, статистику та алгоритми.

4.4 Алгоритми машинного навчання для автоматизаціі тестування

Алгоритм SVM (Support Vector Machine). Він належить до сімейства алгоритмів ML, які намагаються знайти лінійну гіперплощу, яка відокремлює приклади від різних класів. На навчальному етапі SVM обробляє дані тренувань у вигляді вектору розмірів (k-1). Основна мета – знайти максимальну межу (відстань). Метод SVM в основному використовується для бінарної класифікації. Крім того, ще є MartiRank, алгоритм рейтингу, у фазі навчання. Це займає декілька ітерацій, і під час кожної ітерації дані розбиваються на N під-списків, кожен з яких містить 1/n загальної кількості несправностей пристроїв / додатків.

У випадку регресійного тестування необхідно розробити набір тест кейсів, розробка MartiRank є постійним процесом і використовується для виявлення нових помилок. Наприклад: розробник міг відрефакторити деякий код у додатку та додати нову функцію. Тести на регресії показали нам, що отримані моделі відрізняються від попередніх.

Коли ми пишемо тест кейси, ми перевіряємо, як програмне забезпечення повинно поводитися теоретично, але в нас немає реальних даних, також деякі тестові випадки ніколи не можуть бути використані в реальному житті, а деякі, які пропустили тестові випадки, можуть бути найважливішими. Необхідно дозволи машині зчитувати дані журналів і написати тест кейси за цими журналами, це автоматично зберігає багато робочих годин і допомагає в практичному тестуванні. Сервіси, такі як HockeyApp та TestFlight, забезпечують автоматичне тестування мобільних додатків як сервісу.

Що стосується тестування графічного інтерфейсу, є кілька дослідницьких робіт там, де говориться про Deep Learning та Reinforcement Learning для автоматизації тестування. Системи, були протестовані з метою отримання значущих кліків, текстів та натискань кнопки на інтерфейсі графічного інтерфейсу, що дало хорошу кількість навчальних даних. Ці дані потім використовувались для тестування програм протягом декількох годин. Найкраще було те, що нема потреби створювати моделі або тест кейси, а помилки були знайдені впродовж деякого часу, але деякі випадки не були перевірені, що може бути пов'язано з відсутністю навчальних даних. Підхід підсилення покращував тестування, оскільки тести виконувалися через кілька ітерацій.

5 РЕАЛІЗАЦІЯ АЛГОРИТМІВ КЛАСИФІКАЦІЇ ВЕБСТОРІНОК

5.1 Автоматична класифікація вебсторінок

У цьому розділі розглянемо загальний підхід до класифікації вебсторінок. Для вирішення проблеми автоматичної класифікації вебсторінок скористаємось методами обробки природних мов, які були описані в попередніх розділах. Перш ніж розібратися в деталях, необхідно визначити, що мається на увазі під вебсторінкою.

Отже, вебсторінка представляє собою вебдокумент, який закодований в HTML форматі або XHTML та доступ до нього можна отримати локально або з віддаленого вебсервера із використанням веббраузера. Це визначення введено для того, щоб підкреслити, що, коли ми посилаємось на вебсторінку, то ми звертаємося до фактичного документа, що містить інформацію в HTML форматі. Основне припущення цього методу полягає в тому, що класифікація документів може бути застосована до автоматичної категоризації вебсторінок, враховуючи, що елементи вебдокумента перетворюються на звичайний текстовий документ. Цей загальний метод, очевидно, дуже узагальнений. Причиною цього є те, що він досить легко кастомізується під специфічні додатки.

5.1.1 Збір даних

Щоб вебсторінку можна було використовувати для категоризації, її спочатку потрібно зберігати у зручному форматі (що залежить від вимог вебзастосунку). Наприклад, документ можна завантажити з вебсервера та його вміст, що зберігається у реляційній базі даних. Ще один, дуже простий, підхід полягає в тому, щоб завантажити документ і зберегти його так, як є у локальній файловій системі. Доцільно зібрати весь набір вебсторінок (як навчальні, так і тестові зразки) один раз і зберігати його

локально у випадку, якщо параметри класифікатора будуть налаштовані при навчанні. Це важливо з тієї простої причини, що збережені тисячі вебсторінок з Інтернету забирають багато часу, навіть якщо вони автоматизовані.

5.1.2 Вибір класифікатора та навчання

Вибір класифікатора залежить від практичних вимог програми. Звичайно, бажано використовувати класифікатор, який дає хороші результати в усталеній літературі для класифікації документів. Однак можуть бути конкретні вимоги до програми, що звужує вибір класифікатора, наприклад, складність обчислень для фази навчання та/або тестування.

Навчальний процес вимагає, щоб набір категорій був визначений, і навчальні документи потрібно якось позначати відповідною категорією. Обраний класифікатор повинен пройти підготовку на навчальному наборі вебсторінок і вміло оцінюватися на меншому тестовому наборі. Оцінка корисна для визначення оптимального значення будь-яких параметрів класифікатора на доступній підготовці даних. Рисунок 5.1 ілюструє навчальний процес. У наступних розділах ми детальніше розглянемо окремі етапи навчального процесу.

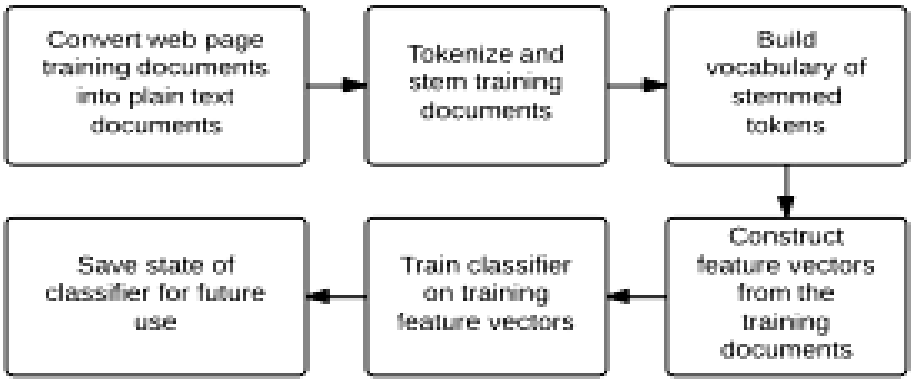


Рисунок 5.1 – Навчання класифікатора для вебсторінок

5.1.3 Вибір та вилучення ознак

Сам документ вебсторінки не дуже підходить для категоризації, адже він, як визначено, містить HTML або XHTML. Для того, щоб використовувати методи машинного навчання для класифікація документа, нам потрібно звести документ вебсторінки до простого текстового документу. Цей звичайний текстовий документ повинен бути перетворений у вектор ознак, придатний для використання з обраним алгоритмом машинного навчання.

Перетворення вебсторінки у звичайний текстовий документ можна зробити досить просто – застосувавши регулярний вираз, який відповідає тегам HTML та/або XHTML і замінити їх порожніми рядками. Однак вебсторінки містять елементи, які не є частиною вмісту, але є частиною дизайну вебсторінки (наприклад, записи в навігаційному меню), який може не стосуватися категоризації. Кращим підходом було б використання аналізатора для вилучення бажаного вмісту з вебсторінки. Точний вміст, який вилучається, залежить від застосунка, але загальним підходом може бути вилучення абзаців та заголовків. Аналізатор повинен застосовувати стандарти таких організацій, як World Wide Web Consortium (W3C) [30] або Web Hypertext Application Technology Working Group (WHATWG) [31].

Після перетворення вебдокумента в звичайний текстовий документ наступним кроком є визначення слів, що складають текстовий зміст. Сліпо застосовувати токенізатор на даний момент не було б доцільно, розглядаючи як кількість відомих вживаних слів протягом усього навчального набору, ймовірно, неймовірно великий. Тому нам потрібно зменшити кількість різних лексем, тобто зменшити кількість розмірів у вектор функції. Як ми можемо це зробити?

Перший, простий метод зменшення розмірів функціонального вектора використовувати стоп список для видалення дуже поширених слів, таких як «the» в англійській мові. Ці слова є загальними у всіляких текстах

і нецікаві для цілей категоризації. Однак кількість окремих токенів все ж, ймовірно, буде великою.

Для тих токенів, що залишилися, виконуємо стемінг, щоб остаточно визначити набір кінцевих наборів токенів, що складають словниковий запас. Це призначить значення одному маркеру (наприклад, слова «car» та «cars» будуть присвоєні одному і тому ж токену «car»). Кількість чітко виражених стемів, швидше за все, буде меншою, ніж кількість чітких слів, отже, цілком ймовірно, що зменшення кількості значно зменшить розміри. Алгоритм, що буде застосований, залежить від мови вебсторінки, тому може бути необхідним класифікувати мову певної вебсторінка за допомогою мовного класифікатора до того.

5.1.4 Представлення документа

Нехай V – словник навчального набору. Далі представимо вебсторінку як вектор функції:

$$\vec{d} = \begin{pmatrix} w_1 \\ \vdots \\ w_i \\ \vdots \\ w_N \end{pmatrix}, \tag{5.1}$$

де w_i – це вага терму i;

N – розмір словника V.

Менш формально, вебсторінка представлена у вигляді мішка слів (bag-of-words) вектора із вагами, що представляють стемінгові токени.

Існує кілька схем зважування, які слід враховувати для ваг w_i. Хорошим вибором є популярна схема зважування TF-IDF, описана в розділі 3.2.4. У деяких алгоритмах, таких як TWCNB, схема зважування є частиною алгоритму. Тому ваги також можуть бути просто частотами для маркерів, якщо алгоритм виконує власну схему зважування. У всіх, крім

найпростішого зважування, словниковий запас повинен бути розширений із частотою токенів. Це найбільш легко зробити з відображенням стмінгових токенів на частоти, або як частина словника, або як окрема структура.

5.2 Опис фреймворку та його основних пакетів

Впровадження системи автоматичної категоризації вебсторінок (далі тут буду називати його фреймворком) було зроблено на Java, завдяки тому, що вона кросплатформена та має високу доступність до сторонніх бібліотек для вирішення завдань, що стосуються машинного навчання та обробки природних мов. Фреймворк дає нам доступ до великої кількості бібліотечних пакетів для забезпечення методів машинного навчання та природничих обробка мови, які необхідні для автоматичної категоризації вебсторінок, як описано у розділі 5.1. Також запропоновано способи перетворення даних документа вебсторінки у формати, необхідні для деяких сторонніх пакетів, що використовуються. Основні пакети фреймворка такі:

- content – інтерфейси та класи для вилучення вмісту із завантаженого вебдокумента;
- ml – являє собою інтерфейс для категоризації вебсторінок в набір заздалегідь визначених категорії, та впровадження класів з використанням конкретних класифікаторів;
- nlp – класи, які використовують засоби обробки природньої мови для перетворення сирого тексту документи в токени;
- tools – інструменти командного рядка для сканування конкретної бази даних вебсторінок, що використовується в експериментах та для пакетної обробки вебсторінок в інші формати, які необхідні деяким стороннім бібліотекам;
- web – класи для представлення та управління вебсторінками.

Пакет content відповідає за вилучення контенту з вебсторінок. Для зручності пакет містить в собі ряд різноманітних методів, які спеціалізуються на витягуванні різних наборів даних із вебсторінки, для якої використовують експериментування.

Екстрактори контенту самі по собі дуже прості: вони отримують необхідний вміст сторінки, далі виконують обробку природньої мови, використовуючи пакет nlp.

Пакет ml містить в собі предиктори категорії використовуючи класифікатори. Класифікатори навчаються за допомогою набору навчальних документів, які були оброблені пакетом nlp , перетворюють їх у вектори функцій і фактично навчаються на них. Категоризація виконується аналогічно, перетворюючи немарковану сторінку у векторний елемент та запускаючи функції прийняття рішення класифікатора. Реалізації класифікаторів MNB, kNN та SVM забезпечуються бібліотекою WEKA для машинного навчання у Java [32], а впровадження TWCNB надане бібліотекою машинного навчання Java Apache Mahout [33].

Пакет nlp пропонує інструмент для обробки природних мов для англомовних документів. По суті, він потребує простий текстовий документ, тобто він робить синтаксичний розбір вебсторінки та виконує токенізацію, видалення стоп-слів та стемінг. Цей інструмент використовується пакетом content для повного вилучення контенту вебсторінки у відповідному форматі (тобто потік виділених токенів). Основна частина роботи виконується бібліотекою Java Apache Lucene, що має багато пакетів для обробки природних мов для різних мов [34].

Пакет tools реалізує різні інструменти командного рядка. Найбільш важливі інструменти – це ті, які перетворюють набір навчальних документів у формати, придатні для читання класифікаторами. Класифікатори WEKA очікують, що документи будуть у форматі ARFF – це формат файлу, який описує список екземплярів, якими надається спільний доступ до набору атрибутів [35]. Класифікатор TWCNB

використовує формат файлу послідовності, який є двійковим форматом у вигляді ключа / пари-значення [36].

Пакет web реалізує класи для подання і управління вебсторінок з бази даних посилань. Представлення вебсторінок використовує jsoup HTML-аналізатор для вилучення тексту, що викликається пакетом content.

5.3 Застосування алгоритмів автоматичної класифікації вебсторінок

5.3.1 Класифікатори

Для оцінки порівнємо метод опорних векторів (SVM), k-найближчих сусідів (kNN), поліноміальний Naive Bayes (MNB), трансформований ваго-нормалізований комплекс Naive Bayes (TWNCB) та класифікатор Кавнара і Тренкле на основі N-грам.

SVM налаштований на використання радіально-базисної функції (або RBF) при $\gamma = 0{,}01$. RBF вважається найпопулярнішою основою, яка використовується для підтримки опорно-векторних машин. Для kNN число найближчого сусіда $k = 30$, деякі попередні тестування свідчать про те, що це був досить хороший компроміс між високим і низьким k. Для класифікатора N-грам профілі встановлені так, щоб мати максимальну довжину 400.

5.3.2 Аналіз даних

Для навчальної виборки були взяті різноманітні вебсторінки з відкритого репозиторія на GitHub. GitHub – це величезний вебсервіс для розробки програмного забезпечання. Сервіс є безкоштовним для проектів з відкритим кодом, тобто кожен може переглянути будь-який публічний проект.

Для тестування було видітено 7 категорій вебсторінок, а саме: форма

реєстрації, форма авторизації, формаа замовлення, форма інтернет оплати, форма оформлення доставки, форма відгука та інші форми.

Для вібірки були взяті лише вебсторінки с контентом англійською мовою. Таким чином, навчання та оцінка фокусуються на англійській мові, щоб уникнути помилкового групування даних, що не належать до однієї категорії. Розподіл зразків на різні категорії показаний в таблиці 5.1.

Таблиця 5.1 – Розподіл вибірок за категоріями даних

Категорія	Кількість тестових прикладів
Форма реєстрації	153
Форма авторизації	78
Форма замовлення	97
Форма інтернет оплати	52
Форма оформлення доставки	112
Форма відгука	139
Інші форми	160

Щоб визначити, які частини вебсторінок стосуються категоризації, було виделено кілька джерел, які використовуються для збору текстового вмісту з вебсторінок, які були використані. Було використано сім різних текстових джерел, а саме:

- T, вміст тегу <title>;
- H, вміст усіх тегів <h1> , ... , <h6>;
- P, вміст усіх тегів <p>;
- TH, вміст джерел T та H;
- HP, вміст джерел H та P;
- TП, вміст джерел T та P;
- THP, вміст джерел T, H та P.

Звичайно, ці джерела були оброблені та кодувалися як вектори функцій, тож вони не використовуються «сирими».

5.3.3 Показники оцінки

Поширеною метрикою для оцінки ефективності класифікатора [18] є мікро-усереднення балів F_1. Вона надає однакові ваги кожному документу і тому вважається середнім показником усіх пар документів/категорій, і, як правило, переважає ефективність класифікатора на загальних категоріях.

Оцінка F_1 приймає значення між 0 і 1, де 0 – найгірший з можливий бал та 1 – найкращий можливий бал. Вона обчислюється за допомогою точності (p) та згадування (r), визначені як:

$$p_i = \frac{TP_i}{TP_i + FP_i}, r_i = \frac{TP_i}{TP_i + FN_i}, \quad (5.2)$$

де TP_i – кількість документів, правильно позначених класом i (відома як справжні позитивні);

FP_i – кількість документів, неправильно позначених класом i (відомий як помилкові позитивні);

FN_i – кількість документів, які повинні були позначені класом i, але не були (відомі як помилкові негативні).

F1 міра для класу i виражається як:

$$F_i = \frac{2 p_i r_i}{p_i + r_i}. \quad (5.3)$$

Загальні значення точності та відкликання отримуються шляхом підсумовування всіх роздільних рішеннь:

$$p = \frac{\sum_{i=1}^{M} TP_i}{\sum_{i=1}^{M} (TP_i + FP_i)}, r = \frac{\sum_{i=1}^{M} TP_i}{\sum_{i=1}^{M} (TP_i + FN_i)}, \quad (5.4)$$

де M – кількість категорій.

Потім визначається мікро-усереднена оцінка F1 як у формулі 5.3, але з використанням глобальних значень точності та виклику:

$$F_1(micro - avereged) = \frac{2pr}{p+r}. \quad (5.5)$$

У таблиці 5.1 видно, що в даних переважають кілька категорій, тому також видається доцільним мати показник, який показує, наскільки добре працює класифікатор для менш поширених категорій. Макро усереднений F_1 бал буде робити саме це. Спочатку він обчислює окремі значення F_1 для різних категорій та беручи середній показник:

$$F_1(macro - avereged) = \frac{\sum_{i=1}^{M} F_i}{M}. \quad (5.6)$$

Можна зробити невеликий висновок, що «сира» точність результатів також використовується як міра ефективності. Цей захід є менш інформативним, оскільки він не показує, наскільки добре працює класифікатор для окремих категорій.

4.3.4 Оцінка ефективності

Розглянемо мікро-усереднену F-міру. У таблиці 5.2 можна побачити мікро-усереднену оцінку F_1 для різних класифікаторів, усереднену за п’ятьма екземплярами для кожного джерела зі стандартним відхиленням у дужках. Такі самі результати також побачити на рисунку 5.1. Тут спостерігається те, що бали для SVM, kNN та MNB значно нижчі порівняно з іншими дослідженнями класифікації тексту. У дослідженні оцінки класифікаторів коливаються приблизно від 0,8 до 0,9, але тут вона становить приблизно 0,45. Оцінки класифікаторів N-грам та TWCNB не мають порівняння у досліджуваному в літературі, однак слід підкреслити, що TWCNB має найвищий бал серед всіх класифікаторів. Також слід зазначити, що джерело HP має найкращі бали і натякає на те, що найкраще із загальних джерел, які були порівняні.

Однак, якщо порівнювати мікро-усереднену F-міру дослідження Робіні, продуктивність більше відповідає дійсності. Методи, як описано у Робіні досягають мікро-усередненого показника F-міри 0,6506 для класифікатора NB як найкращий результат. Цікаво, що в статті Робіні найкращий результат був досягнутий, використовуючи найбільше джерело тексту, яке включає в себе заголовок сторінки, метатег та весь вміст тексту. Кращий результат у цій роботі використовує менше даних як джерело, використовуючи лише вміст заголовки окремих абзаців. TWCNB досягає порівнянної ефективності результатів, але слід підкреслити, що Робіні використовує інший набір даних і термінні схеми зважування.

Таблиця 5.2 – Мікро-усереднена оцінка F1 для різних класифікаторів, усереднена для всіх екземплярів

Source	N-Gram	MNB	TWCNB	kNN	SVM
T	0,1908 (0,0156)	0,3415 (0,0276)	0,4152 (0,0347)	0,4366 (0,0109)	0,4343 (0,0038)
H	0,1553 (0,0112)	0,3929 (0,1131)	0,4877 (0,0082)	0,4453 (0,0022)	0,4926 (0,0197)
P	0,2122 (0,0269)	0,5626 (0,0061)	0,6078 (0,0039)	0,4408 (0,0244)	0,5358 (0,0037)
TH	0,1398 (0,0268)	0,4361 (0,0076)	0,4013 (0,0579)	0,4103 (0,0083)	0,4453 (0,0078)
TP	0,1521 (0,0176)	0,4385 (0,0026)	0,4453 (0,0113)	0,4423 (0,0135)	0,4854 (0,0221)
HP	0,1636 (0,0269)	0,5681 (0,0181)	0,6101 (0,0078)	0,4363 (0,0221)	0,5646 (0,0119)
THP	0,1618 (0,0078)	0,3877 (0,0069)	0,3812 (0,0406)	0,3899 (0,0011)	0,4531 (0,0109)

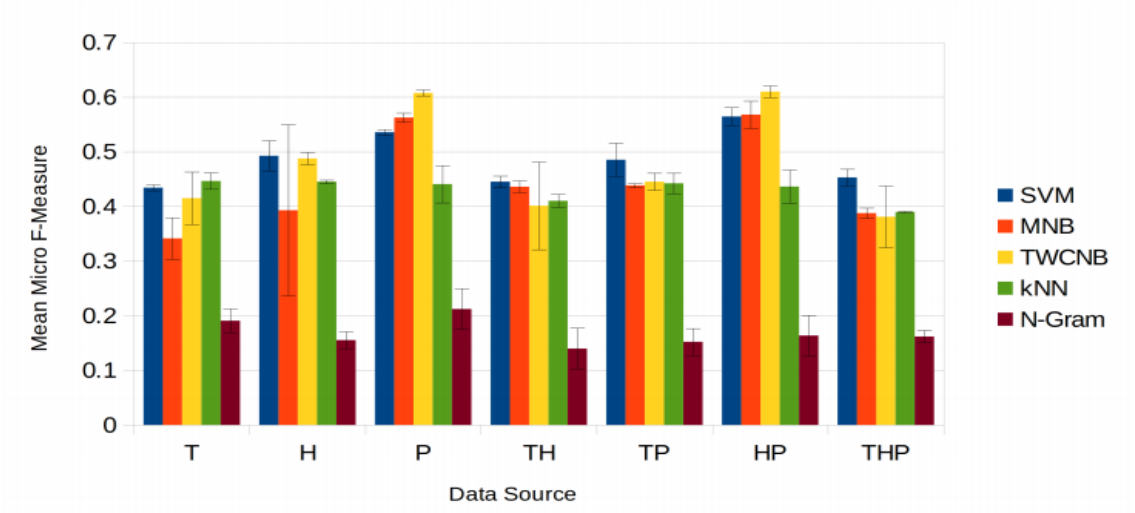


Рисунок 5.1 – Середній показник мікро-вимірювання з показаним інтервалом 95%

Розглянемо макро-усереднену F-міру. У таблиці 5.3 показано макро-усереднений бал F_1 для різних класифікаторів, усереднене за п’ятьма методами для кожного джерела зі стандартним відхиленням у дужках.

Такі самі результати також показані на рисунку 5.2. У порівнянні з вивченою літературою [18], макро-усереднена F-міра для класифікаторів SVM та kNN є значно гіршою. Однак бали за два байєсівські класифікатори (MNB та TWCNB) у цих експериментах перевершують NB у досліджуваному текстовому класифікаторі. Для макро-усередненого F-вимірювача джерело, яке має найкращі результати – це джерело P, на другому місці – джерело HP .

Цікаво, що порівняно з результатами макро-усередненого показника F-вимірювання, ми бачимо, що найкращі результати класифікаторів NB у цій роботі приблизно рівні до найкращих результатів для класифікатора SVM.

Таблиця 5.3 – Макро-усереднена оцінка F_1 для різних класифікаторів, усереднена для всіх екземплярів

Source	N-Gram	MNB	TWCNB	kNN	SVM
T	0,1282 (0,0208)	0,2227 (0,0228)	0,3023 (0,0225)	0,1225 (0,0019)	0,1219 (0,0244)
H	0,1095 (0,0178)	0,2439 (0,1034)	0,3357 (0,0191)	0,1087 (0,0038)	0,1763 (0,0049)
P	0,1783 (0,0369)	0,4435 (0,0177)	0,4457 (0,0236)	0,1689 (0,0132)	0,2451 (0,0195)
TH	0,1078 (0,0089)	0,2777 (0,0079)	0,2782 (0,0615)	0,1224 (0,0068)	0,0801 (0,0091)
TP	0,1114 (0,0149)	0,3165 (0,0186)	0,3141 (0,0042)	0,1614 (0,0046)	0,1964 (0,0101)
HP	0,1253 (0,0294)	0,4359 (0,0033)	0,4497 (0,0140)	0,1519 (0,0087)	0,2925 (0,0033)
THP	0,1246 (0,0059)	0,2793 (0,0239)	0,2592 (0,0306)	0,1422 (0,0036)	0,1846 (0,0053)

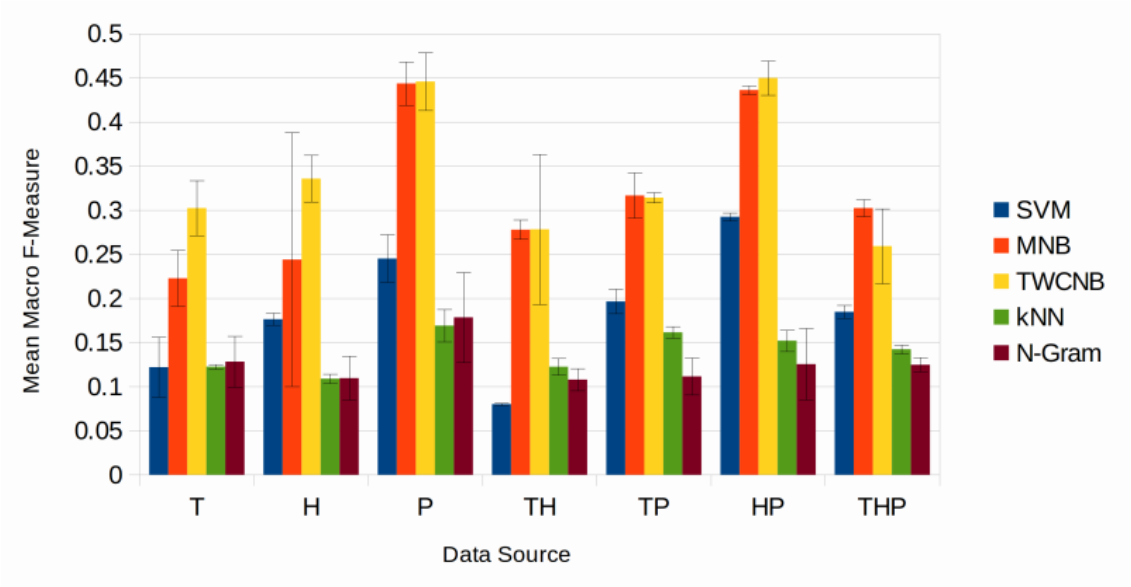


Рисунок 5.2 – Середній макро-показник F-вимірювання

Розглянемо далі «сиру» точність класифікації. У таблиці 5.4 показана точність різних класифікаторів з точки зору правильних класифікацій, усереднений за п’ятьма методами для кожного джерела зі стандартом відхилення в дужках. Такі самі результати також показані на рисунку 5.3. Вивчаючи ці результати, можна швидко побачити, що загальний показник досить поганий. Також зауважимо, що стандартні відхилення обох класифікаторів NB відносно високі, вони більш чутливі до фактичного використовуваного екземпляра. Цікаво, що класифікатор N-грам працює однаково для всіх джерел даних, з відносно незначними відмінностями. Стандартне відхилення низьке. Однак точності немає ніде.

Таблиця 5.4 – Точність різних класифікаторів з точки зору правильної класифікації, усереднена для всіх примірників

Source	N-Gram	MNB	TWCNB	kNN	SVM
T	19,542 (1,165)	40,565 (7,518)	33,643 (7,397)	43,569 (1,1561)	43,515 (2,619)
H	15,429 (0,765)	39,702 (9,097)	39,216 (9,069)	43,713 (1,701)	46,048 (2,486)
P	15,333 (3,130)	40,501 (14,777)	43,111 (6,068)	41,867 (3,543)	46,949 (6,639)
TH	15,213 (0,514)	41,157 (3,428)	38,431 (4,366)	41,529 (1,141)	44,920 (1,896)
TP	14,824 (1,111)	40,186 (4,653)	40,451 (5,568)	42,535 (2,017)	46,822 (2,365)
HP	17,154 (1,003)	48,894 (8,482)	53,271 (9,389)	41,717 (0,369)	52,328 (4,201)
THP	14,785 (1,938)	34,136 (4,732)	35,685 (5,851)	39,021 (1,011)	43,243 (2,434)

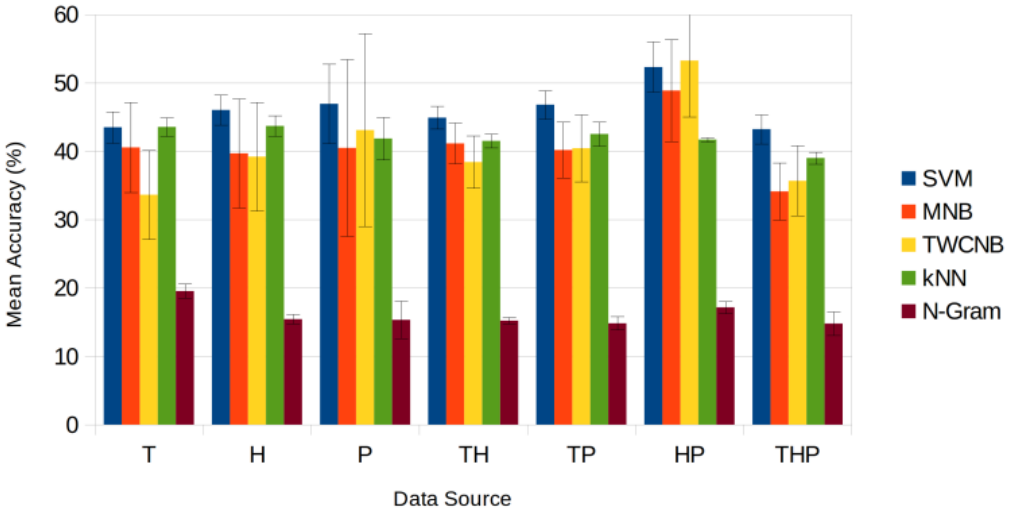


Рисунок 5.3 – Середня точність класифікації

Короткий недокументований експеримент із набором даних HP показує, що точність, коли зразки категорії «Інші форми» видаляються з наближенням 70% для класифікаторів NB та SVM. Це натякає на те, що категорія «Інші форми» занадто загальна, щоб ефективно включати її в дані.

6 РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

6.1 Аналіз та реалізація алгоритму MartiRank

Необхідно представити структуру, розроблену для тестування додатків та відстеження помилок, з метою зробити їх більш надійними. Мої перші результати зосереджені на реалізації MartiRank. Одне з ускладнень у цьому алгоритмі виникло через суперечливу технічну номенклатуру: «тестування», «регресія», «перевірка», «модель» та інші відповідні терміни мають дуже різні значення для експертів з машинного навчання, ніж для програмістів. Тут я використовую терміни «тестування» та «регресійне тестування», як це підходить для аудиторії у сфері розробки програмного забезпечення, а також сенс «моделі» у машинному навчанні (тобто правила, отримані під час тренувань на наборі прикладів) та «перевірки» (вимірювання достовірності, досягнутого при використанні цих правил для класифікації набору даних навчання, а не інший набір даних).

6.1.1 Алгоритм MartiRank

Розглянемо псевдокод алгоритму MartiRank на прикладі 6.1. Псевдокод алгоритму вказує на збої фідера, де мітка вказує на кількість збоїв (нуль означає, що фідер ніколи не зазнав поразки); однак алгоритм може бути застосований до будь-якого набору даних, присвоєним значенням атрибутів, позначеним невід'ємними значеннями. На кожній ітерації MartiRank сукупність навчальних даних розбита на під-списки. Для кожного під-списку MartiRank сортує цей сегмент за кожним атрибутом, зростанням та зменшенням, і вибирає атрибут, який дає найкращу «якість».

```
    inputs: list L of attribute-value descriptions of
feeders
    with associated nr. of failures; nr of boosting
rounds T
    output: marti-model M
    1. let M be the empty model
    2. for each round t=1,..,T do:
    - partition L into t sub-lists L1, .., Lt s.t. each
Lj has
    same nr. of failures; let th2, .., tht be the
location
    of the splits in terms of the normalized fraction
     of feeders that fall above the split.
    - for each sub-list i=1,..,t do:
    i. compute quality of Li sort
    ii. for each attribute A do:
    1. sort Li according to A in ascending
    order, compute quality of resulting sort
    2. sort Li according to A in descending
    order, compute quality of resulting sort
    - if there exists attribute A and polarity P that
    improves Li’s sort, then:
    i. if i > 1, add thi to M at level t, position i
    ii. add A to M at level t, position i.
    iii. sort Li according to (A,P)
    - else:
    i. if i > 1, add thi to M at level t, position i
    ii. add “NOP” to M at level t, position i.
    3. output M
```

Приклад 6.1 – Алгоритм MartiRank

Для порівняння якості, у всіх реалізаціях використовується невеликий варіант, адаптований для класифікації, що відповідає характеристиками приймача Curve (AUC). AUC – це звичайна метрика якості, яка використовується у спільноті ML: найкращий варіант 1.0, найгірший – 0.0, а вибір 0.5 – випадковий. Таким чином, на кожній ітерації визначення кожного сегменту має три аспекти: відсоток прикладів із вихідних наборів даних, які знаходяться в сегменті, атрибути, на які їх слід сортувати, та напрямок (зростання або зменшення) сортування. У згенерованій моделі N-й круг з'являється на N-му рядку простого

текстового файлу, сегменти розділені крапками з комою та атрибути сегмента розділені комами.

Розглянемо приклад, який може з'явитися в третьому рядку файлу моделі представляючи третю ітерацію: 0.4000,32,a;0.6500,12,d;1.0000,nop. Це означає, що перший сегмент містить 40% прикладів у наборі даних і сортує їх за атрибутом 32, зростає. Другий сегмент містить наступні 25% (65 мінус 40) і сортує їх за атрибутом 12, зменшуючись. Останній сегмент містить решту атрибутів і робить «NOP» (no-op), тобто не сортує їх знову, оскільки порядок, отриманий внаслідок попередньої ітерації, був найвищим у порівнянні з повторним сортуванням на будь-якому атрибуті.

Цю модель потім можна повторно застосувати до навчальних даних (так звана «перевірка» в термінології ML) або застосувати до іншого, раніше не розглянутого набору даних (так звані «тестові дані»). В обох випадках вихід є ранжируванням прикладів набору даних, і загальна якість всього рейтингу може бути розрахована.

Перша з трьох реалізацій була написана на Perl, далі іменована PerlMarti, як пряма реалізація алгоритму, який не включав оптимізацію. Однак при застосуванні до великих наборів даних, наприклад, тисяч прикладів із сотнями атрибутів, PerlMarti досить повільний. Версія C, надалі CMarti, була написана для підвищення продуктивності (швидкості). CMarti також представив деякі експериментальні параметри для покращення якості.

Інша реалізація, написана також у C, називається FastCMarti, була розроблена таким чином, щоб мінімізувати накладні витрати для повторного сортування значень атрибутів. Він відсортував повний набір даних на кожен атрибут на початку виконання перед першою ітерацією і запам'ятовував результати; він також використовував швидший алгоритм сортування, ніж CMarti (звідси і назва). Ця реалізація також представила декілька експериментальних варіантів від тих, що містяться в CMarti.

6.1.2 Набори даних

Алгоритм MartiRank заснований на сортуванні з неявним припущенням, що сортовані значення є чисельними. Хоча в принципі можуть використовуватися лексикографічні сортування, нечислові сортування не здаються інтуїтивно привабливими, як прогнозування ML; наприклад, це може не означати, що електричний пристрій, вироблений компанією Westinghouse, більше або менше, ніж щось зроблене компанією General Electric, просто завдяки алфавітному замовленню. Таким чином, реалізація передбачає, що всі вхідні дані будуть чисельними.

Хоча значна частина цікавих даних реальних даних дійсно складається з числових значень – в тому числі десяткових знаків з плаваючою точкою, дати та цілих чисел – деякі дані натомість є категоричними. Категоричні дані відносяться до атрибутів, у яких є різні значення (як правило, алфавітно-цифрові), але немає порядок сортування, який буде відповідати алгоритму рейтингу. У цих випадках даний атрибут з K окремими значеннями розширюється до K різних атрибутів, кожен з двома можливими значеннями. Тобто, серед атрибутів K, кожен приклад повинен мати рівно один і K-1 0.

Деякі атрибути в наборі даних у реальному світі повинні бути видалені або проігноровані, наприклад, оскільки ці значення складаються з вільних текстових коментарів.

6.1.3 Підхід до тестування

Алгоритми CMarti та FastCMarti надають параметри у реальному часі, які вмикають/вимикають «оптимізацію», призначені для підвищення якості результатів. Вони, як правило, включають рандомізацію (ймовірнісні рішення), але важко оцінити результати тестів, коли результати не детерміновані. Тому ці параметри були відключені для

всіх випробувань дотепер: мета у порівнянні цих реалізацій полягала не в тому, щоб досягти кращих результатів, а для отримання послідовних результатів. Спочатку я вважала, що PerlMarti є потенційним «золотим стандартом», оскільки він був найвірнішим для алгоритму, як і спочатку кодований винахідником алгоритму, але там знайшлися помилки. Проте той факт, що є три реалізації MartiRank, дуже допоміг: можна було б загалом припустити, що буде, якщо всі параметри вимкнені, – якщо буде узгоджено дві реалізації.

Необхідно було зосередитися на двох типах тестування: тестування порівняння, щоб побачити, чи всі три реалізації дали однакові результати, і тестування регресії для порівняння нових версій даної реалізації з попередніми (після виправлення помилок, рефакторинга та вдосконалень оптимізації).

Набори даних для деяких тест кейсів були побудовані вручну, наприклад, так, щоб ручне моделювання алгоритму MartiRank дало «ідеальний» рейтинг, з усіма позитивними прикладами (збої з подачею) у верхній частині та всі негативні приклади внизу. Ці набори даних були дуже малими, наприклад, 10 прикладів з 3 атрибутами.

Також потрібні великі набори даних, для реалізації розумної кількості ітерацій MartiRank (за замовчуванням – 10), причому в кожному сегменті все ще досить багато прикладів у наступних ітераціях. Я протестувала з деякими (великими) наборами даних реального світу, які, як правило, мають багато категоріальних атрибутів, багато повторюють числові значення та багато відсутніх значень. Проте для того, щоб більше контролювати тестові випадки, наприклад, зосередитись на граничних умовах з визначених класів еквівалентності, більшість великих наборів даних автоматично генеруються за допомогою F-збоїв (з позитивними мітками), N числових атрибутів і K категоріальних атрибутів. F – будь-який відсоток від 0 до 100. N числові атрибути були вказані як включені чи не включені будь-які повторювані значення, з відсутніми від 0 до 100

відсоткових значень; набори значень для кожного атрибуту були незалежними. Для кожного з категорійних атрибутів K було вказано кількість різних значень і відсотків на категорію та їх відсутність.

Оцінка результатів тестування зосереджена, перш за все, на моделях, оскільки практично завжди в тому випадку, якщо дві версії виробляють дві різні моделі, то рейтинги будуть різними: якщо різні моделі виробляють однакові рейтинги, це, імовірно, випадково (тобто ефект самого набору даних, а не моделі), і це не означає, що версії створюють «послідовні» результати. Проте, навіть якщо дві реалізації або зміни дають таку ж модель, ми не можемо припустити, що рейтинги будуть однаковими: CMarti і PerlMarti генерують рейтинги за допомогою програм, що відрізняються від коду, що використовується для генерації моделей, тож можливо, що відмінності можуть існувати.

FastCMarti не дотримується типової наглядної ML-конвенції, в якій для створення моделі використовується набір навчальних даних, а потім для цієї моделі надається окремий набір даних «тестування» з невідомими мітками для ранжирування. Замість цього два набори даних об'єднані, і кожен приклад позначається відповідним чином. FastCMarti працює на комбінованому наборі даних, але для створення моделі використовуються лише навчальні дані.

Дані тестування сортуються та сегментуються разом із даними тренувань, а остаточний рейтинг даних тесту є виходом – сама модель є лише побічним ефектом, який нам потрібно було видобувати для порівняння між версіями.

6.1.4 Тестування на реальних даних

Спершу були проведені тести з деякими реальними даними для всіх трьох реалізацій. Ці набори даних містять категоріальні дані, а також некоректні та повторювані значення. Надія полягала в тому, що при всіх

«оптимізаціях», три реалізації дадуть ідентичні моделі та рейтинги. PerlMarti та FastCMarti не тільки дали різні моделі, але і CMarti відтворював сегментні помилки. Використовуючи утиліти трасування для випадку CMarti, ми виявили, що якийсь код, який був потрібний лише для однієї з варіантів оптимізації, все ще викликався, навіть коли цей прапор був вимкнений, але внутрішній стан був неприйнятний для цього шляху виконання. Після рефакторингу коду дефекти зникли. Тим не менш, тоді модель, створена алгоритмом CMarti, все-таки відрізнялася від тієї, яку створила будь-яка з двох інших. Ці випробування продемонстрували необхідність «підроблених» (контрольованих) наборів даних, для вивчення класів еквівалентності неповторюваних чи повторюваних значень, або відсутніх значень, а також не категоріальних або категоріальних атрибутів (які обов'язково повторюються).

6.1.5 Корисність розглянутих алгоритмів

Структура тестування полегшила роботу, допомагаючи у створенні, виконанні та аналізі тестів. Можливість керувати властивостями наборів даних була критичною для обмеження обсягу окремих тестів і для виявлення конкретних проблем у тому, як код обробляв різні класи еквівалентності та їхні межі. Інструмент створення даних виявився простим та надійним у порівнянні з альтернативними підходами.

Інструмент порівняння моделі надав багато переваг перед «diff», оскільки він чітко перераховує відмінності та усвідомлює різні аспекти сегмента MartiRank (кількість прикладів, атрибут сортування та напрямок). Інструмент порівняння рейтингу, звичайно, не набагато корисніший, ніж «diff» для тестування.

В решті решт, інструмент аналізу виявився надзвичайно корисним для визначення того, де відбуваються відмінності в моделях та порядок сортування. Це дало чудове розуміння внутрішньої частини реалізацій.

6.2 Формування Page Object сторінки

Page Object – один з найбільш корисних і використовуваних архітектурних рішень в автоматизації. Даний шаблон проектування допомагає инкапсулювати роботу з окремими елементами сторінки, що дозволяє зменшити кількість коду і його підтримку. Якщо, наприклад, дизайн на одній зі сторінок змінений, то нам потрібно буде переписати тільки відповідний клас, що описує цю сторінку. Основні переваги:

- поділ коду тестів і опису сторінок;
- об'єднання всіх дій по роботі з вебсторінкою в одному місці.

Розглянемо фрагмент класу Page Object для форми реєстрації (приклад 6.2).

```
public class Registration_page {
    private Map<String, String> data;
    private WebDriver driver;
    private int timeout = 15;

    @FindBy(id = "Email")
    @CacheLookup
    private WebElement email1;


    @FindBy(id = "UserName")
    @CacheLookup
    private WebElement name;
```

Приклад 6.2 – Фрагмент коду класу Page Object, сформованого за допомогою плагіну Selenium Page Object Generator

6.2.1 Інструмент Selenium Page Object Generator

Page Object – це шаблон дизайну, який став популярним у автоматизації тестування для покращення технічного обслуговування та зменшення дублювання коду. Це об'єктно-орієнтований клас, який слугує

інтерфейсом до сторінки AUT. Тести використовують методи класу Page Object, коли їм потрібно взаємодіяти з інтерфейсом цієї сторінки. Перевага полягає в тому, що якщо змінюється інтерфейс для сторінки, то тести самі по собі не потрібно змінювати, необхідно лише змінити деякі методи у класі Page Object. Згодом всі зміни для підтримки нового інтерфейсу знаходяться в одному місці. Це дуже зручно, адже всі локатори та методи знаходяться в одному класі, а тестів, основаних на цьому класі може бути безліч, тож набагато легше внести правку в одному місці, ніж правити кожен клас з тестами, які використовують ці локатори та методи.

Для формування Page Object я використовую Page Object Generator, розроблений Selenium. Це досить гнучкий генератор моделі об'єктів для скорочення часу на тестування. Selenium Page Object Generator є важливим інструментом для покращення робочого процесу. Він може створювати модель об'єкта сторінки на активній вкладці Chrome одним натисканням, за умови, що всі параметри та шаблон налаштовані.

Згенерована модель об'єктів сторінок буде збережена в попередньо налаштованій папці. Він прагне спростити ручне налаштування, але все ж таки необхідна участь людини у налагодженні параметрів. Раніше це була лише beta-версія, яка передбачала обмежену функціональність. Сьогоднішня версія підтримує 3 різних напрямки: Java, C# і Robot Framework.

Selenium Page Object Generator автоматично генерує код для об'єктів сторінки. Я вважаю, що це дуже зручний інструмент, який скорочує години, витрачені на кодування в декілька разів. Серед особливостей можна відзначити такі аспекти:

- генерує код для об'єктів сторінки;
- генерує методи, які використовуються для заповнення форм даної сторінки;
- генерує код для методу, який фіксує повідомлення про помилки зі сторінки;

- генерує методи, які перевіряють валідацію на певній сторінці (використовуючи постачальника даних TestNG);

- генерує постачальника даних TestNG для методу перевірки рівня поля.

Розробка великих вебзастосунків є викликом для будь-якої компанії. У сьогоднішньому швидкодіючому середовищі зусилля, спрямовані на адаптацію програмної системи, що працює в Інтернеті, до вимог змінюється безперервно. Це створює серйозні проблеми при тестуванні вебсистем, що вимагають підвищення рівня автоматизації. З цих причин інструменти автоматизованого тестування стали популярними в галузі протягом останніх 10 років, а також завдяки великій різноманітності таких тестових завдань, як регресія, тестування систем або графічного інтерфейсу.

Автоматизовані тести можуть виконуватися швидко і часто, що робить їх досить рентабельними для вебпрограмного забезпечення з використанням довгострокового очікуваного технічного обслуговування та еволюційного життя. Саме тому, щоб зекономити час та ресурси необхідно використовувати максимальну кількість інструментів для автоматизації, які зараз дуже поширені. Один з них, безперечно, Selenium Page Object Generator.

Ще одним його плюсом є те, що він безкоштовний. Це робить його ще більш затребуваним. Звичайно, можна створити і свій інструмент для формування моделі об’єктів сторінки, та інші допоміжні інструменти. Деякі компанії якраз розробляють такі інструменти не лише для себе, а й продають їх іншим. Наприклад, компанія розробляє внутрішній проект, який допомагає автоматично, швидко та надійно перейти з десктопного програмного продукту у вебзастосунок та змігрувати бази даних з Oracle у MS SQL Server. Цей інструмент компанія застосовує як для проектів, які замовили, так і продає його для користування іншим. Саме тому зацікавлена компанія може придбати лише інструмент, та займатися

внутрішньою міграцією. Це може бути повїязано не лише з фінансовою частиною, а й з безпекою персональних даних.

Отже, використання інструменту Page Object Generator допоможе зберегти час та ресурси на етапі розробки автоматизованих тестів.

6.2.2 Застосування Page Object Generator на практиці

Для того, щоб сформувати Page Object за допомогою розглянутого інструменту, я спочатку його налаштувала. В налаштуваннях можна обрати необхідні типи полів, такі як наприклад, поле вводу, текст, посилання, кнопка тощо. Далі сформувала класс, який вже можна було б використовувати для тестування (у прикладі 6.2 наведено фрагмент сформованого класу Page Object).

Для початку я вирішила випробувати його, чи працює він. У фрагменті коду, що наведений у прикладі 6.2, видно, що елементи позначаються через анотацію @CacheLookup. Ця анотація виконує досить просту але потрібну функцію. Даний маркер має сенс використовувати для тих елементів, які точно не будуть змінюватися. Знайшовши перший раз елемент, реалізація WebDriver кешує його, та в майбутньому все використовує кеш, що дає більшу швидкість проходження автотестів. Звичайно, це досить корисний інструмент, але з моєю тестовою формою він відмовлявся працювати. Після усіх можливих налаштувань я отримувала помилку NullPointerException саме через те, що драйвер не міг знайти потрібний елемент. І якщо цю помилку ще можна було виправили, то була ще одна особливість написання даного класу. Замість того, щоб створити один метод, наприклад для вводу у поле, використаний інструмент сгенерував окремі методи для кожного поля, що в подальшому ускладнить пітримку тестів. Я вважаю, то краще створити окремі але загальні методи, для вводу в поле, натискання кнопки, тощо. Далі просто передавати параметри, такі як локатор (для кнопки), чи локатор та текст

(для вводу в поле).

Через нелодіки у Page Object Generator, використання здається дуже незручним, тому я прийняла рішення відмовитися від цього інструмента та формувати модель об’єктів власноруч.

Розроблена програма аналізує усі елементи на сторінці. Виявлено, що на тестовій сторінці реєстрації у потрібному нам місці на сторінці усі поля типу input знаходяться за сокроченим локатором .//form/div[i]//input, де і – порядковий номер від 2 до 11 (всього 10 елементів зазначеного типу). Далі програма аналізує ці 10 елементів, визначаючи ім’я кожного з них. У результаті формуються локатори для моделі об’єктів (приклад 6.3).

```
    public           By           UserName           =
By.xpath(".//form/div[2]//input");
    public           By            Surname           =
By.xpath(".//form/div[3]//input");
    public          By          Patronomic           =
By.xpath(".//form/div[4]//input");
    public           By            Comment           =
By.xpath(".//form/div[5]//input");
    public          By          PhoneNumber           =
By.xpath(".//form/div[6]//input");
    public            By            Email            =
By.xpath(".//form/div[7]//input");
    public           By           Password           =
By.xpath(".//form/div[8]//input");
    public         By         ConfirmPassword         =
By.xpath(".//form/div[9]//input");
    public           By            Photo            =
By.xpath(".//form/div[10]//input");
    public         By         RegisterButton         =
By.xpath(".//form/div[11]//input");
```

Приклад 6.3 – Сформовані локатори для об’єктів типу input

Імена цих локаторів взяті з html коду сторінки, а саме, це параметр name. Тож програма виконується у циклі 10 разів, кожного разу формуючи локатор для кожного елемента.

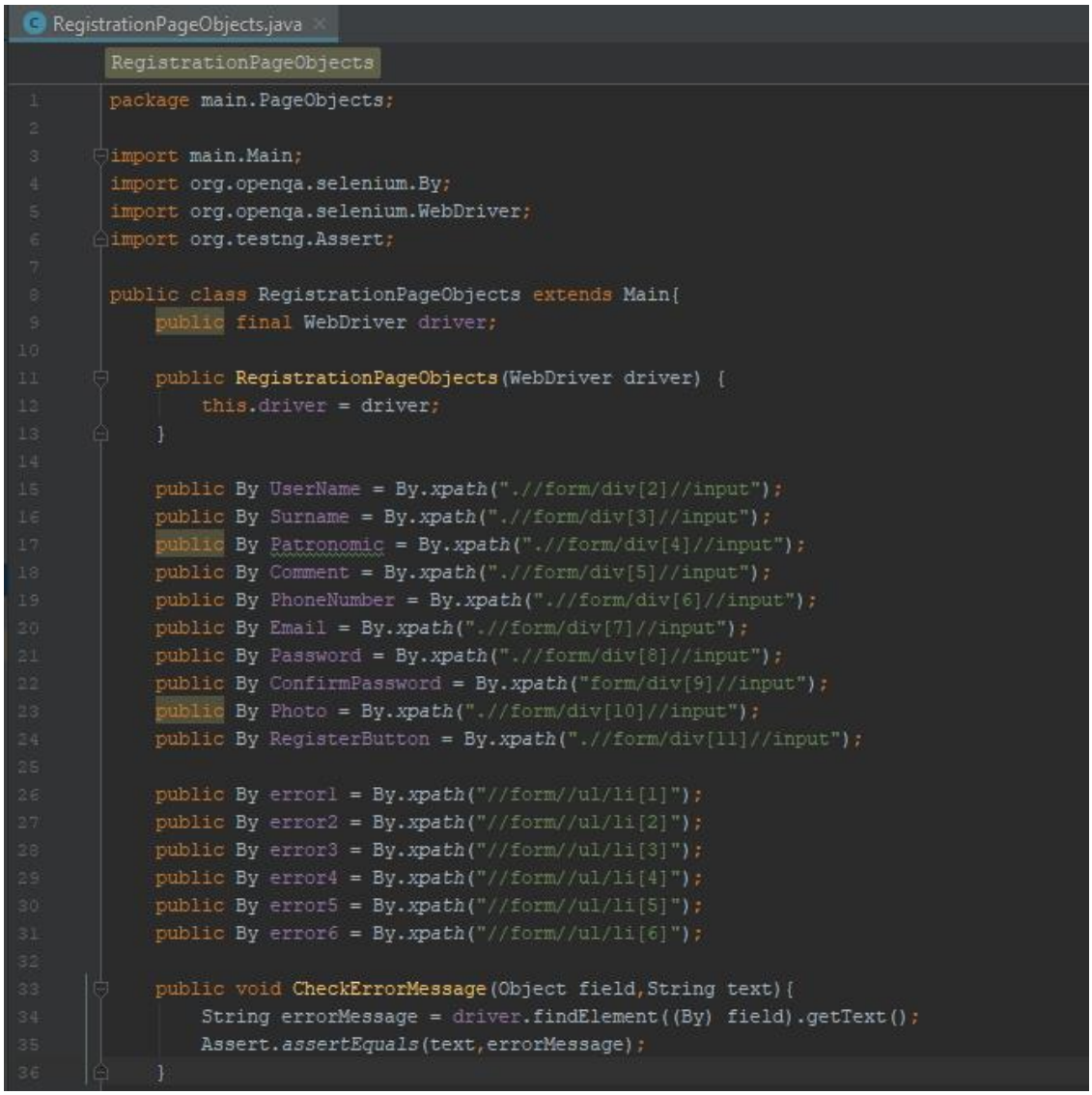
Далі необхідно визначити поля, які обов’язково будуть перевірятися на коректність та обов’язковість. Ці поля легко визначити за флагом data-val=«true». Цей флаг означає, що до цього елемента прив’язана ненав’язлива валідація. Далі йдуть також необхідні флаги, які допоможуть у визначенні помилок, що будуть з’являтися при валідації. Наприклад, з data-val-required можна витягнути повідомлення, яке ми отримаємо, якщо залишимо поле пустим. Є також, наприклад, data-val-email, де прописане повідомлення, яке отримаємо, якщо введемо некоректний формат email.

Є поля типу li, в яких відображаються помилки або підказки, які ми могли бачити у атрибуті data-val-required розглянутих об’єктів. Кількість помилок співпадає з кількістю обов’язкових полів, розрашиваних за таким локатором: .//form//ul/li[i], де i – порядковий номер помилки.

До моделі об'єкту сторінки також був доданий метод CheckErrorMessage, який буде перевіряти помилки та підказки, що були викликані валідацією полів.

На рисунку 6.1 зображено вже сформований клас з об’єктами сторінки – RegistrationPageObjects. Він містить у собі локатори для основних полів вводу, які будуть тестуватися; локатори для полів з помилками та підказками; метод CheckErrorMessage, який необхідний для перевірки повідомлень з підказками.

Клас RegistrationPageObjects наслідується від головного класу Main.



```java
RegistrationPageObjects.java
RegistrationPageObjects
package main.PageObjects;

import main.Main;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.testng.Assert;

public class RegistrationPageObjects extends Main{
    public final WebDriver driver;

    public RegistrationPageObjects(WebDriver driver) {
        this.driver = driver;
    }

    public By UserName = By.xpath(".//form/div[2]//input");
    public By Surname = By.xpath(".//form/div[3]//input");
    public By Patronomic = By.xpath(".//form/div[4]//input");
    public By Comment = By.xpath(".//form/div[5]//input");
    public By PhoneNumber = By.xpath(".//form/div[6]//input");
    public By Email = By.xpath(".//form/div[7]//input");
    public By Password = By.xpath(".//form/div[8]//input");
    public By ConfirmPassword = By.xpath("form/div[9]//input");
    public By Photo = By.xpath(".//form/div[10]//input");
    public By RegisterButton = By.xpath(".//form/div[11]//input");

    public By error1 = By.xpath("//form//ul/li[1]");
    public By error2 = By.xpath("//form//ul/li[2]");
    public By error3 = By.xpath("//form//ul/li[3]");
    public By error4 = By.xpath("//form//ul/li[4]");
    public By error5 = By.xpath("//form//ul/li[5]");
    public By error6 = By.xpath("//form//ul/li[6]");

    public void CheckErrorMessage(Object field,String text){
        String errorMessage = driver.findElement((By) field).getText();
        Assert.assertEquals(text,errorMessage);
    }
```

Рисунок 6.1 – Сформований клас RegistrationPageObjects

6.3 Створення класу Main

Клас Main у автоматизації тестування зазвичай має основні та найголовніші методи, які використовуються у всьому проекті. Цей клас створений власноруч, а не автоматично запрограмованою програмою. Він включає в себе роботу з WebDriver – без нього реалізація тестів неможлива. Тож, у методі BeforeClass() прописані всі ті дії, які будуть виконуватися щоразу, коли буде запущений конкретний клас з тестами. Цей метод описує створення драйвера – саме chromeDriver. Це все

індивідуально, але для мене він є комфортнішим та привабливішим, ніж драйвер для Firefox (приклад 6.4).

```
@BeforeClass
public void BeforeClass() throws InterruptedException
{
    System.setProperty("webdriver.chrome.driver",
"chromeDriver/chromedriver.exe");
    ChromeOptions chromeOptions = new
ChromeOptions();
    chromeOptions.addArguments("start-maximized");
    chromeOptions.addArguments("--no-sandbox");
    chromeOptions.addArguments("--enable-precache");
    chromeOptions.addArguments("--enable-offline-
cache-access");
    chromeOptions.addArguments("disable-infobars");
    driver = new ChromeDriver(chromeOptions);
    driver.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);
    driver.manage().timeouts().pageLoadTimeout(20,
TimeUnit.SECONDS);
    }
```

Приклад 6.4 – Реалізація метода BeforeClass()

Також обов’язково присутній afterClass() – він закриває вікно з драйвером та сам драйвер. Дуже важливо винести ці дії саме в afterClass() у головному класі Main (приклад 6.5). Це пов’язано з тим, що з часом доведеться щось змінювати, або зміняться технології та доведеться рефакторити код, то в даному випадку необхідно буде внести зміни лише в одному класі, а не у кожному окремо.

В мене траплялися такі ситуації на різних проектах – WebDriver закривався некоректно, залишав за собою незавершені задачі, з часом кількість незакритих задач ставала настільки великою, що машина з автотестами переставала відповідати. В одному випадку ця проблема вирішилась шляхом додавання лише однієї нової строки у afterClass(), а в іншому випадку довелося правити кожен клас, бо в них локально були

прописані методи закривання драйвера.

```
@AfterClass
public void afterClass() throws IOException {
    driver.close();
    driver.quit();
}
```

Приклад 6.5 – Реалізація метода afterClass()

Клас Main містить в собі ще декілька методів. Один з них – ClickButton (By button). Як зрозуміло з назви, метод натискає на кнопку. На вхід передаємо локатор кнопки, на яку необхідно натиснути.

Метод InputInField(Object field, String text) – створений для вводу даних у поля. На вході маємо об’єкт (поле), в який необхідно вводити текст, та власне сам текст строкового типу.

Як бачимо, лише два різних методи, а скільки кнопок можна натиснути, скільки полів заповнити, передаючи лише локатор необхідного поля. Я вважаю, що такий підхід є більш правильним, ніж створення окремих методів на кожне поле, як це відбувається при використанні інструмента Page Object Generator.

Ще один метод – GetPassword(int num) – це метод для отримання строки необхідної довжини. Планується використовувати для генерації пароля. Так як є валідація та конкретні обмеження на вхід, то для зручної генерації великих паролей – це оптимальний варіант. Метод повертає змінну типу String.

6.4 Формування класу з тестами

Створення класу RegistrationTests з тестами вже лежить за розробленою програмою. Після аналізу Page Objects, система починає тестувати реєстраційну форму. Перш за все, класс RegistrationTests

наслідується від Main. Тепер можна використовувати методи, описані в класі Main. Я завжди починаю тестувати такі форми з тесту SubmitEmptyForm. Це з першого ж погляду дає зрозуміти, чи є хоч якась валідація на формі, чи ее вже можна тестувати. Розглянемо приклад 6.6.

```
    @Test
    public void Test01_SubmitEmptyForm() {
    RegistrationPageObjects register = new
RegistrationPageObjects(driver);
    ClickButton(register.RegisterButton);
    register.CheckErrorMessage(register.error1, "Field
\"Name\" is required");
    register.CheckErrorMessage(register.error2, "Field
\"Surname\" is required");
    register.CheckErrorMessage(register.error3, "Field
\"About me\" is required");
    register.CheckErrorMessage(register.error4, "Phone
number is required");
    register.CheckErrorMessage(register.error5, "Field
\"Email\" is required");
    register.CheckErrorMessage(register.error6, "Field
\"Password\" is required");
    }
```

Приклад 6.6 – Сформований фрагмент коду – перший тест

Як бачимо, у наведеному фрагменті коду – перший тест на перевірку валідації обов’язкових полів. Після натискання на кнопку з’являються 6 повідомлень/підказок, що деякі поля є обов’язковими. Кожна помилка має свій локатор, але не унікальний. Усі вони розташовані за таким локатором .//form//ul/li[i], де i – порядковий номер помилки. Але йдуть вони обов’язково у тому порядку, у якому розташовані відповідні поля.

Зараз, наприклад, перше поле «Name» не заповнене, тож першою помилкою буде повідомлення «Field «Name» is required» з локатором .//form//ul/li[1]. Точно так із з іншими полями. У цьому випадку, наприклад, поле «About me» є третім незаповненим обов’язковим полем. Помилка має локатор .//form//ul/li[3]. Але, якщо, заповнити перших два

поля – «Name» та «Surname», то помилка на валідацію поля «About me» буде вже першою – через те, что тепер це поле є першим незаповненим обов'язковим полем.

Користуючись цим принципом, можна далі тестувати форму, перевіряючи кожне поле по одинці, тим самим, роблячи локатор на помилку валідації єдиним. Система бере в якості очікуваного результату для повідомлення – значення атрибуту data-val-required у кожного обов'язкового поля. Наприклад, для поля «Name» помилка виглядає так: data-val-required=«Field «Name» is required». Тож саме цю строку тест візьме за еталон. А в актуальний результат вже бере фактичний текст з поля з помилкою.

У наведеному фрагменті (приклад 6.7) перевіряється валідація поля «Phone Number». Таким же чином будуть побудовані й інші тести на перевірку валідації одного поля. У всі поля вводимо коректні дані, а у поле, що перевіряється – некоректні. Перевіряємо, що з'явилася помилка, що введені дані не коректні. Тест помилки беремо з html-коду, з властивостей конкретного поля.

```
    @Test
    public void Test02_CheckPhoneNumberValidation() {
        RegistrationPageObjects register = new
RegistrationPageObjects(driver);
        InputInField(register.UserName, "Name");
        InputInField(register.Surname, "Surname");
        InputInField(register.Comment, "About me");
        InputInField(register.PhoneNumber, "123");
        InputInField(register.Email,
"example@example.com");
        String pass = GetPassword(6);
        InputInField(register.Password, pass);
        InputInField(register.ConfirmPassword, pass);
        ClickButton(register.RegisterButton);
        register.CheckErrorMessage(register.error1,
"Enter number in format (XXX) XXX-XX-XX");
    }
```

Приклад 6.7 – Фрагмент коду – перевірка валідації поля «Phone Number»

Розглянемо ще ситуацію з полями паролів. На поля стоять обмеження – мінімальна кількість символів 6, максимальна – 100. Необхідно перевірити валідацію у випадках, коли кількість введених символів не входить в заданий проміжок. Наприклад, ввели 5-значний пароль. З'являється повідомлення «The Password must be at least 6 characters long». В цілому все правильно – саме таке повідомлення і в характеристиках поля.

Далі тестуємо з 101 – значним паролем. Знову з'являється повідомлення «The Password must be at least 6 characters long». Воно вже некоректне. Але, саме це ми маємо у властивостях поля. Тест же бере текст помилки саме з властивостей поля, тому в даному випадку він навіть пройде, бо очікуваний результат відповідає актуальному. Але, з точки зору специфікації – це звичайно баг. Просто система, яка тестує, не мала змоги знайти інше повідоблення на цей тестовий випадок, бо його просто немає. Це вже баг не розробленої системи, а розробників, які не додали коректну валідацію на поле «Password». Дана ситуація показує, що розроблена система все ж потребує участі та перевірки людиною. Вона поки що не взмозі без допомоги людини зрозуміти, чи це коретна поведінка програмного продукту, чи ні.

На рисунку 6.2 можна побачити вже готовий набір тестів, сформований розробленою системою. Сгенеровані 10 тестів проходять. Але, вони все ж таки потребують контроля зі сторони людини. Це пов'язано з тим, що є деякі помилки, які система ШІ не змогла виявити в силу своєї невеликої автономності та самостіності. Тому тести все ж потребують участі людини у розробці. Але, все ж таки, не зважаючи на те, що людина все ж буде приймати участь у контролі тестів, ця система полегшить роботу тестувальника, бо вона сама формує майже всі тестові випадки та перевіряє їх. Людині залишається лише перевірити правильність згідно специфікації та написати необхідні баги, внести правки до коду тестів.

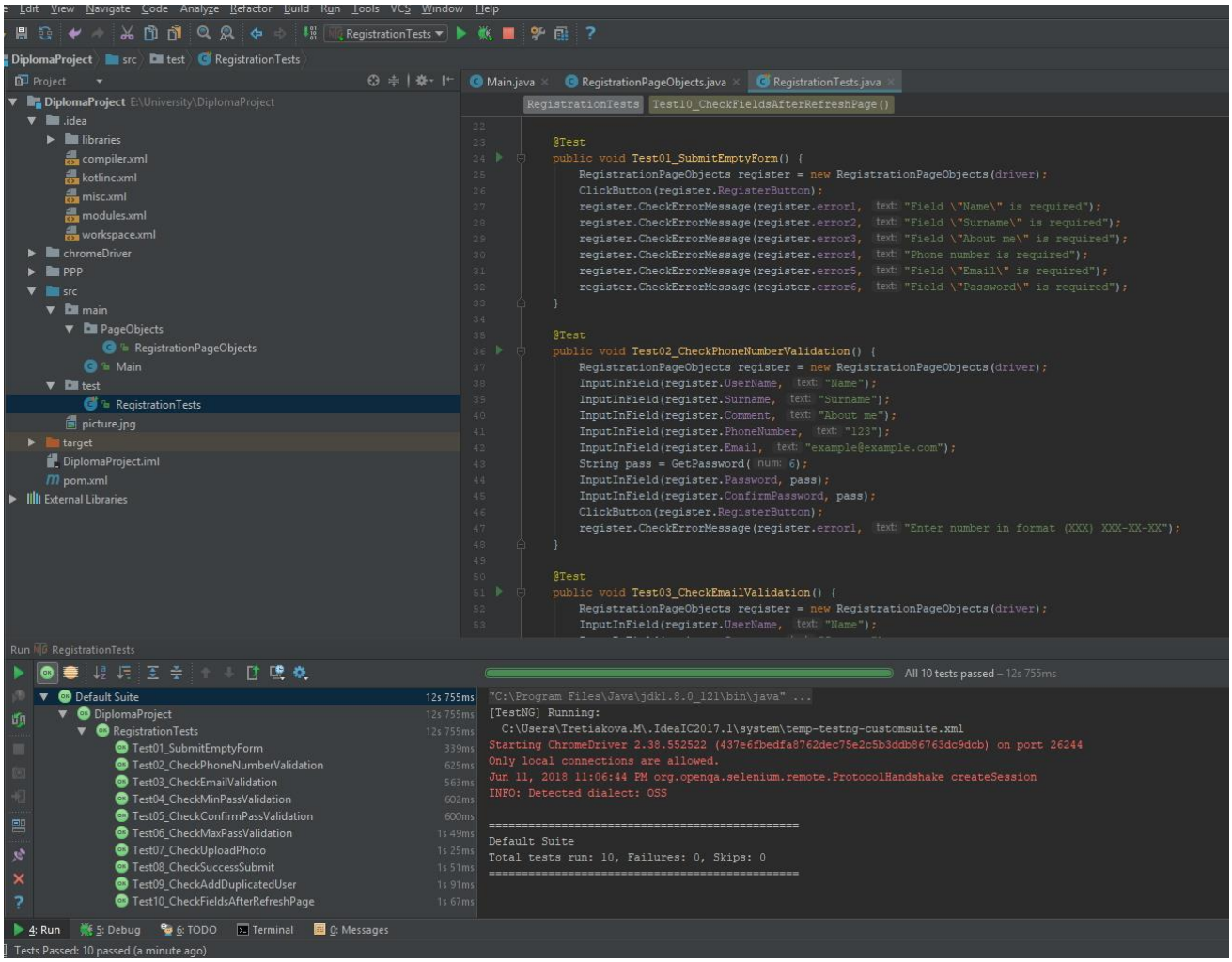


Рисунок 6.2 – Результати роботи програми, сгенеровані розробленою системою тести; результати проходження тестів

На наведеному рисунку 6.2 можна побачити 10 сгенерованих тестів, що покривають практично весь функціонал даної форми. Тестова форма є не дуже складною, тож тестів не багато. Можна лише уявити, скільки тестів може бути на велику сторінку з багаться елементами різних типів на обмежень.

ВИСНОВКИ

В атестаційній роботі були досліджені алгоритми класифікації текстів, була проведена порівняльна характеристика алгоритмів для виявлення найбільш точного для застосування у задачах класифікації вебсторінок.

Розглянуті методи та алгоритми штучного інтелекту, які можна застосовати у автоматизації тестування програмного забезпечення. Використання штучного інтелекту у тестування дозволяє усунути помилки, пов’язані з людським фактором та підвищити прозорість усіх етапів створення програмного забезпечення. Зараз вже багато компаній застосовують методи машинного навчання для того, щоб підвищити рівень автоматизації та конкурентоспроможності. Серед переваг штучного інтелекту, безумовно, є те, що він проводить тестування не лише швидше, ніж людина, а й якісніше.

Створена система з використанням штучного інтелекту, яка допомагає тестувальникам класифікувати нову сторінку та створити нові автоматизовані тести для цієї сторінки. На вхід подаємо посилання на вебсторінку. Далі програма аналізує контент та класифікує її до однієї з категорій, наприклад форма замовлення, доставки, реєстрації. Далі аналізує об’єкти, що знаходяться на цій сторінці, знаходить усі поля типу input, далі формує Page Object для цієї сторінки з локаторами усіх знайдених об’єктів. Я планувала використовувати для цих цілей вже готовий інструмент, що генерує Page Object, але під час тестування та аналізу стало зрозуміло, що він не підходить до вимог та не працює з моєю формою. Тому було прийнято рішення написати аналог для моєї роботи.

В результаті роботи ми отримуємо готовий набір тестів, який був сформований системою, оснований на попередньому аналізі та тестуванні. Отримані тести необхідно попередньо перевірити, адже система ще не бездоганна. Хоч вона і допомагає у написанні автоматизованих тестів, все

ж вона ще потребу втручання людини. Це пов’язано ще з тим, що програмний продукт, який ми тестуємо, теж не ідеальний, містить баги, та деякі з них можуть збивати створену систему та це призведе до помилок. Їх скоріш за все буде не так складно виявити, якщо тестувальник знає специфікацію. Це ще один нюанс, розроблена система поки що примітивна та не може аналізувати специфікацію до проекту, або цієї специфікації просто нема.

Все ж таки людина буде приймати участь у налаштуванні та підтримці тестів, але розроблена система значно полегшить роботу тестувальника, адже вона сама аналізує всі тестові випадки та перевіряє їх. На даному етапі розроблений алгоритм інтелектуальної автоматизації вебсторінок реєстрації, але в подальшому планується розробили алгоритми для автоматизації інших категорій сторінок. Це пов’язано з тим, що різні сторінки тестуються за різними сценаріями, тож необхідно до кожного типу застосовувати свій підхід.

Чи замінить AI людину в майбутньому? Безумовно, ні, найближчим часом штучний інтелект не зможе повністю замінити людину. Протягом наступного десятиліття ми вже сподіваємося побачити тестування з використанням AI без неприємних побічних ефектів. Не можна поки що сказати, що AI збирається повністю автоматизувати тестування програмного забезпечення, включаючи UI-тести. Тестування програм аналогічне водінню в деяких аспектах, але це складніше, оскільки ці системи повинні розуміти складні людські взаємодії. AI сьогодні не розуміє, як він це робить. Він просто автоматизує завдання, грунтуючись на великій кількості історичних даних.

Роль тестувальника – автоматизувати, а не бути автоматизованим. Команди QA мають можливість використати нові методи автоматизації, що стимулюватимуть нові способи та алгоритми роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Трет’якова М.С. Застосування штучного інтелекту в тестуванні програмного забезпечення // Матеріали 23-й Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» (Харків, 16-18 квітня. 2019 р.). Харків, 2019. С. 37-38.

2. Шилдт Г. Java. Полное руководство. 8-е изд. : Пер. с англ. Москва:ООО «И.Д. Вильямс», 2012. 1104 с.

3. Шилдт Г. Java: The Complete Reference. Москва : ООО «И.Д. Вильямс», 2009. 1040 с.

4. Java Code Convention. URL : http://www.oracle.com/technetwork/ java/codeconventions-150003.pdf (Дата звернення 02.12.2019).

5. The Java® Language Specification Java SE 7 Edition. URL : http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf (Дата звернення 02.12.2019).

6. Канер С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. К. : ДиаСофт, 2001. 544 с.

7. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. Москва : Дело, 2007. 312 с.

8. Бейзер Б. Тестирование черного ящика. Москва : «Питер», 2004 г. 318 с.

9. Тамре Л. Введение в тестирование программного обеспечения. Москва : «Вильямс», 2003. 368 с.

10. Marsland S. Machine Learning: An Algorithmic Perspective. Chapman & Hall/CRC, 1st edition, 2009.

11. Scholkopf B., Chapelle O., Zien A. Semi-Supervised Learning. The MIT Press, 1st edition, 2006.

12. Caruana R. A. Multitask learning: A knowledge-based source of inductive bias // In Proceedings of the Tenth International Conference on

Machine Learning. 1993. P. 41-48.

13. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice Hall, second edition, 2008.

14. Rethinking chinese word segmentation: tokenization, character classification, or wordbreak identification / Huang C., Simon P., Hsieh S., Prevot L. // In Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, ACL '07. Association for Computational Linguistics, 2007. P. 69-72.

15. Smirnov I. Overview of Stemming Algorithms. Technical report, DePaul University, 2008.

16. Porter M. F. An algorithm for su-x stripping. Program: Electronic Library & Information Systems: 1980. P. 211-218.

17. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. New York: Cambridge University Press, 2008.

18. Yang Y., Liu X. A Re-Examination of Text Categorization Methods, 1999.

19. Tackling the Poor Assumptions of Naive Bayes Text Classifiers / Rennie J. D., Shih L., Teevan J., Karger D. R.. // In Proceedings of the Twentieth International Conference on Machine Learning, 2003. P. 616-623.

20. Sci-kit learn. Naive Bayes. URL : http://scikit-learn.org/ dev/modules/naive_bayes.html (Дата звернення 02.12.2019).

21. McCallum A., Nigam K. A comparison of event models for Naive Bayes text classification: AAAI-98 Workshop on Learning for Text Categorization. AAAI Press, 1998. P. 41-48.

22. Multinomial naive bayes for text categorization revisited / Kibriya A. M., Frank E., Pfahringer B., Holmes G. // In Proceedings of the 17th Australian joint conference on Advances in Artificial Intelligence, AI'04. Berlin, Heidelberg, 2004. P 488-499.

23. Yang L. Distance Metric Learning: A Comprehensive Survey, 2006.

24. An improved k-nearest-neighbor algorithm for text categorization / Jiang S., Pang G., Wu M., Kuang L. // Expert Syst. January 2012. P. 1503-1509.

25. Vapnik V. N. The nature of statistical learning theory. Springer Verlag New York, Inc., New York, NY, USA, 1995. 334 p.

26. Kwok J. Automated Text Categorization Using Support Vector Machine // In Proceedings of the International Conference on Neural Information Processing ICONIP, 1998. P. 347-351.

27. Duan K., Keerthi S.S. Which is the best multiclass SVM method? An empirical study // In Proceedings of the Sixth International Workshop on Multiple Classifier Systems, 2005. P. 278-285.

28. Hsu C., Lin C. A Comparison of Methods for Multiclass Support Vector Machines, 2002.

29. Oxford Dictionaries Online. The OEC: Facts about the language. URL : http://oxforddictionaries.com/words/the-oec-facts-about-the-language (Дата звернення 25.11.2019).

30. World Wide Web Consortium (W3C). HTML. URL : http://www.w3.org/html/wg/drafts/html/master/ (Дата звернення 28.11.2019).

31. Web Hypertext Application Technology Working Group (WHATWG).HTML. URL : http://www.whatwg.org/html (Дата звернення 28.11.2019).

32. Machine Learning Group at the University of Waikato. Weka 3: Data Mining Software in Java. URL : http://www.cs.waikato.ac.nz/ml/weka/ (Дата звернення 15.11.2019).

33. The Apache Software Foundation. Apache Mahout. URL : http://mahout.apache.org/ (Дата звернення 02.12.2019).

34. The Apache Software Foundation. Apache Lucene Core. URL : http://lucene.apache.org/core/ (Дата звернення 09.12.2019).

35. When to do Manual Testing and when to do Automated Testing. URL : https://www.testing-whiz.com/blog/when-to-do-manual-testing-and-when-to (Дата звернення 07.12.2019).

36. 6 levels of AI-based testing: Have no fear, QA pros. URL : https://techbeacon.com/6-levels-ai-based-testing-have-no-fear-qa-pros (Дата звернення 08.12.2019).

37. 5 ways AI will change software testing. URL : https://techbeacon.com/5-ways-ai-will-change-software-testing (Дата звернення 09.12.2019).

38. How AI is changing test automation. URL : https://techbeacon.com/how-ai-changing-test-automation-5-examples (Дата звернення 04.12.2019).