Перегудов С.В.
студент,
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

МІНІМІЗАЦІЯ ХИБНИХ ПОВІДОМЛЕНЬ, ЗГЕНЕРОВАНИХ ЗАСОБАМИ СТАТИЧНОГО ТЕСТУВАННЯ

Основною задачею, що встановлена перед засобами статичного тестування, є виявлення дефектів або вразливостей у програмному коді на ранніх етапах розробки. Це спрощує та пришвидшує процес розробки, а також значно покращує якість кодової бази.

Сучасні засоби вдало виконують встановлену перед ними задачу. Вихідні звіти з тестування містять майже всі присутні у коді недоліки, а разом з ними і доволі велику кількість хибних повідомлень [1]. В залежності від засобу статичного тестування, складності кодової бази та мови програмування кількість хибних повідомлень про дефекти може складати до 60 відсотків загальної кількості всіх повідомлень [2].

У результаті проведеного порівняння існуючих методів обробки повідомлень про дефекти [3–8] було зроблено висновок про перспективність використання методів машинного навчання, а саме методів класифікації.

Для навчання моделей було використано датасет, згенерований на основі результатів тестування спеціалізованого набору даних «Juliet» для мови програмування C++ від 2017 року, за допомогою засобу статичного тестування «SonarQube Community Edition». Результати тестування наведенні у таблиці 1. Кожен запис в датасеті містить наступні атрибути: кількість строк коду у файлі з недоліком; розмір вкладеності керуючих структур у файлі; кількість можливих різних варіантів виконання функції; кількість циклів у межах функції; кількість зовнішніх викликів функції; кількість внутрішніх викликів функції; кількість різних методів у межах файлу; кількість функцій, що викликаються під час роботи функції, що містить недолік; кількість зовнішніх файлів, що викликаються або задіяні під час роботи функції; умовне позначення, що вказує чи містить функція дійсну вразливість.

Таблиця 1

Результати тестування Juliet

Загальна кількість дефектів у коді	Кількість хибних повідомлень	Кількість дійсних повідомлень	Кількість не виявлених дефектів
2537	931	1362	238

Для побудови тестових моделей було обрано наступні найбільш поширеніші методи класифікації:

– метод опорних векторів;

– метод випадкового лісу;

– метод KNN;

– метод XGBoost tree;

– метод C5;

– метод випадкових дерев;

– нейрона мережа.

Результати побудови моделей для різних методів класифікації наведені у таблиці 2.

Таблиця 2

Результати моделей класифікації

Назва методу	Accuracy, %	Recall, %	Precision, %	AUC
Метод опорних векторів	89,214	88,946	89,053	0,867
Метод випадкового лісу	87,775	87,846	87,623	0,88
Метод KNN	87,147	87,147	86,974	0,811
Метод XGBoost tree	83,535	82,119	83,273	0,844
Метод C5	87,414	87,52	87,463	0,83
Метод випадкових дерев	79,834	78,547	78,834	0,793
Нейрона мережа	84,633	84,834	83,953	0,808

На основі даних наведених у таблиці 2, метод опорних векторів продемонстрував найкращі результати за всіма критеріями. Тому цей метод був розглянутий більш детально: було розглянуто використання методу з різними функціями ядра та їх параметрами.

Було проведено більше 1000 дослідів, з різним значенням параметрів, поліноміальної, сигмоїдної та радіально базисної функції. Результати найкращих представників, для кожної функції, наведено у таблиці 3.

Таблиця 3

Найкращі представники для кожної функції ядра

Назва функції	C	Degree	σ	Accuracy	Recall	Precision	AUC
Поліноміальна	2	10	2	87,538	87,646	88,142	0,834
Сигмоїдна	10	-	10	91,244	91,245	91,342	0,908
Радіально базисна	15	-	0,5	89,914	90,572	90,637	0,872

На підставі отриманих даних із тестування моделей класифікації, метод опорних векторів із сигмоїдним ядром, з параметрами C = 10, σ = 10, демонструє найкращі результати.

Таким чином, в результаті проведених досліджень зроблено наступні висновки:

– методи класифікації є найперспективнішим способом оброки повідомлень, згенерованих засобами статичного аналізу коду;

– метод опорних векторів демонструє найкращі результати, серед інших методів класифікації;

– метод опорних векторів, в основі якого лежить сигмоїдна функція, з наступними параметрами C = 10, σ = 10, найкраще справляється з задачею класифікації повідомлень про дефекти.

Отримані результати в подальшому можливо використовувати для розробки програмного додатку до засобів статичного тестування, з метою зменшення кількості хибно позитивних повідомлень про дефекти у кінцевому звіті з тестування.

Список використаних джерел:

1. Muske T. Survey of Approaches for HandlingStatic Analysis Alarms / T. Muske, A. Serebrenik., 2017. – 153 с.

2. Kremenek T. Z-Ranking: Using Statistical Analysis to Counter the Impact of Static Analysis Approximations / T. Kremenek, D. Engler., 2003. – 78 с.

3. Chou A. False Positives Over Time: A Problem in Deploying Static Analysis Tools [Електронний ресурс] / Andy Chou. – 2015. – Режим доступу: до ресурсу: https://www.cs.umd.edu/~pugh/BugWorkshop05/papers/34-chou.pdf

4. Woosuk L. ound Non-Statistical Clustering of Static Analysis Alarms [Електронний ресурс] / L. Woosuk, L. Wonchan, Y. Kwangkeun. – Режим доступу: до ресурсу: https://docplayer.net/14058062-Sound-non-statistical-clustering-of-static-analysis-alarms.html

5. Taming False Alarms from a Domain-Unaware C Analyzer by a Bayesian Statistical Post Analysis / J.Yungbum, S. Jaeho, Y. Kwangkeun, K. Jaehwang. – Seoul, 2005. – 127 с.

6. Csallner C. DSD-Crasher: A hybrid analysis tool for bug finding [Електронний ресурс] / C. Csallner, Y. Smaragdakis, X. Tao. – 2008. – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/1348250.1348254

7. Alikhashashneh E. Using Machine Learning Techniques to Classifyand Predict Static Code Analysis Tool Warnings / E. Alikhashashneh, R. Rajeev. – Indianapolis, 2019. – 52 с.

8. Kathleen G. Feature Set Selection for Improved Classification of Static Analysis Alerts / Goeschel Kathleen., 2019. – 120 с.

Світенко Г.М.

студент,

Харківський національний університет радіоелектроніки

ПАРАЛЕЛІЗМ У PYTHON

Паралельні обчислення – це форма обчислень, в якої два або більше обчислення виконуються за один і той же проміжок часу. Паралельні обчислення ґрунтуються на тому, що великі задачі можна розділити на кілька менших завдань, кожне з яких можна вирішити незалежно від інших. Застосування паралельних обчислень покликано зменшити загальний час виконання програми.

Паралелізм є окремим випадком організації паралельних обчислень, коли в один момент часу розв’язуються декілька задач. Для виконання паралельних програм необхідним є комп’ютер з багатоядерним процесором або з кількома процесорами.

Паралельні обчислення у Python [1; 2] можуть бути реалізовані з використанням трьох модулів:

– multiprocessing – для виконання операції, які є ресурсномісткими;

– asyncio – для виконання операції, що пов’язані з введенням та виведенням даних;

– threading – універсальний модуль, що підходить для усіх інших варіантів операцій.