Diaconescu Teodora-Cosmina

Grupa 352

Proiect Testarea Sistemelor Software

Îmbunătățirea testării unitare cu ajutorul inteligenței artificiale

Partea 1

# Despre

Echipa A9

Link Repo: <https://github.com/Teo0o0/TssProject>

Scop: Obținerea unor rapoarte referitoare la cod sursă

Limbaj redactare: Python

Limbajele pe care se pot realiza rapoartele: Python (.py) și Java (.java)

# Metrici definite

## Lizibilitatea codului (readability) :

Ce spune ChatGPT despre lizibilitatea unui cod sursă? “Lizibilitatea unui cod se referă la cât de ușor poate fi înțeles și interpretat de către oameni. Un cod lizibil este scris într-un mod clar și concis, folosind convenții de denumire intuitive, structuri logice evidente și comentarii explicative acolo unde este necesar.”

Atât din această definiție, cât și din experiența personală, concluzionăm: cu cât o secvență de cod este mai greu de înțeles, probabilitatea de a ignora anumite cazuri în cadrul procesului de realizare a testelor crește. Acest lucru poate avea un impact semnificativ asupra robusteții produsului rezultat.

Pentru a valorifica aspecte legate de “good practice” ale codului sursă, multe limbaje de programare au definite librării specifice pentru aceasta. În python, există pylint, ce analizează codul fără a-l rula, identifică erori și impune un standard de codare. [2] În Java, echivalentul este Checkstyle, iar în C++ există Clang-Tidy și Cppcheck.

Ceea ce am dorit să valorificăm în cadrul proiectului este o măsură universală a lizibilității, ce ține de diferite trăsături vizibile în mai multe limbaje de programare și ce pot fi transferate. Modelul de tipul Arbori de regresie (Random Forest Regressor) folosit a fost antrenat cu ajutorul unui dataset de pe platforma Kaggle: “Code Snippets: Insights and Readability” [3].

Trăsăturile considerate în procesul de antrenare:

- Numărul de linii

- Lungimea totală a codului

- Numărul de comentarii

- Complexitatea ciclomatică

- Numărul de indentări

- Numărul de bucle/bifurcații

- Numărul de identificatori/operații

Despre modelul folosit : Random Forest Regressor ( Arbori de regresie )

Este un algoritm de învățare supervizată, pentru probleme de regresie, pe un domeniu continuu, ce face predicții bazându-se pe trăsăturile introduse. Folosește mai mulți arbori de regresie, fiecare antrenat pe o parte aleasă random din datele de antrenare folosite și o parte random din trăsăturile folosite. Acest model tinde să nu intre în supraadaptare (overfit).

“Random Forest Regressor este o implementare din biblioteca scikit-learn pentru regresia cu ansamblu de arbori de decizie. Iată o descriere mai detaliată a acestui algoritm:

* Ensemble of Decision Trees: Random Forest Regressor construiește un ansamblu de arbori de decizie, unde fiecare arbore este antrenat pe o submulțime aleatoare a setului de date de antrenare și a caracteristicilor. Această abordare de ansamblu ajută la reducerea riscului de overfitting și la îmbunătățirea performanței generale.
* Bootstrapping: Random Forest folosește bootstrapping pentru a crea diferite seturi de date de antrenare. Acest lucru implică selectarea aleatoare a eșantioanelor cu înlocuire din setul de date inițial pentru a crea subseturi multiple de date, fiecare folosit pentru antrenarea unui arbore de decizie diferit.
* Random Feature Selection: La fiecare divizare a arborelui, algoritmul selectează un subset aleatoriu de caracteristici. Acest lucru ajută la diversificarea arborilor din forestă și la reducerea corelării între ei.
* Predicție prin Votare: În cazul regresiei, predicția finală este obținută prin media predicțiilor tuturor arborilor din forestă. Cu cât sunt mai mulți arbori în forestă, cu atât este mai robustă predicția finală.
* Hyperparameter Tuning: Random Forest Regressor are anumite hyperparametri care pot fi ajustați pentru a optimiza performanța modelului. Acești parametri includ numărul de arbori din forestă, adâncimea maximă a fiecărui arbore, numărul minim de eșantioane necesare pentru a face o divizare etc.
* Performanță și Scalabilitate: Random Forest Regressor este cunoscut pentru performanța și scalabilitatea sa bune. Este capabil să lucreze cu seturi de date mari și cu un număr mare de caracteristici, iar antrenarea și evaluarea sa pot fi paralelizate pentru a accelera procesul.”

(Generat de ChatGPT)

## Complexitatea ciclomatică (cyclomatic complexity) :

Ce spune ChatGPT despre complexitatea ciclomatică?

“Complexitatea ciclomatică este o măsură a complexității structurale a unui program de calculator. A fost introdusă de către Thomas J. McCabe Sr. în 1976 și este folosită în special în ingineria software pentru a evalua complexitatea unui program și pentru a identifica părțile critice care ar putea fi dificil de înțeles, testat sau întreținut.

În esență, complexitatea ciclomatică numără numărul de căi independente prin care poate trece fluxul de control al unui program. Se calculează prin numărarea numărului de noduri de decizie (de exemplu, instrucțiuni if, switch, while, for) și adăugând 1 la rezultat. Acest lucru înseamnă că complexitatea ciclomatică este egală cu numărul minim de căi care trebuie parcurse pentru a acoperi toate posibilitățile de execuție a programului.

O valoare mare a complexității ciclomatice indică adesea o structură mai complexă a programului, ceea ce poate face dificilă înțelegerea, testarea și întreținerea sa. De obicei, se recomandă ca valoarea complexității ciclomatice să fie menținută la un nivel scăzut pentru a asigura o dezvoltare și o întreținere mai ușoară a codului.”

De ce ne raportăm la aceasta? Complexitatea ciclomatică ne oferă un minim de posibilități ce ar trebui luate în calcul, în cazul în care codul este scris corect, pentru a avea o acoperire totală în cadrul procesului de testare al codului. O valoare mare a complexității ciclomatice indică o structură mai complexă a programului, ceea ce ar putea face testarea sa dificilă.

În cazul limbajului Python, definim această măsură ca fiind: numărul de puncte de decizie (if, while, for, elif) + 1 – numărul de puncte de ieșire (return, brake, continue). În cazul limbajului Java, calculul este 1 + numărul de puncte de decizie (if, while, for, switch).

## Complexitatea problemei rezolvate :

Măsura propusă se referă la dificultatea problemei și nivelul de înțelegere necesar pentru a-l realiza. În cazul sarcinilor de complexitate scăzută, testarea este relativ simplă, cu fie mai directe, fie mai puține cazuri de acoperit. Odată cu creșterea complexității, apar mai multe cazuri de tratat, din ce în ce mai marginale și greu de găsit.

Pentru aceasta folosim o rețea neurală preantrenată, CodeBERT. Trecem problema de la una de clasificare în una de regresie. Label-urile deja date se încadrează în trei clase: ușor, mediu, greu. Le valorificăm drept 0.0, 1.0, 2.0 și considerăm domeniul continuu 0-2 pentru estimarea complexității sarcinii.

Acest model este compilat și ar putea fi introdus în rapoarte după mici ajustări.

CodeBERT : A Pre-Trained Model for Programming and Natural Languages. Este un model preantrenat pentru limbaje de programare (Python, Java, JavaScript, PHP, Ruby, Go). [1]

# Conținut

Obiectivul proiectului este de a genera rapoarte cu diferite metrici pentru fișiere ce conțin cod Python sau Java.

În cazul limbajului Python, fișierul source.py parcurge folderul curent, generând câte un raport pentru fiecare fișier de tip .py găsit.

În cazul limbajului Java, fișierul java\_worker/get\_files.py parcurge folderul Tss\_Test, ce conține două fișiere de tip .java, generate de ChatGPT, și scrie rapoarte pentru fiecare.

Rapoartele conțin primele două metrici discutate pentru fiecare funcție găsită în parte.

# De Viitor

- Un prim lucru de îmbunătățit este calitatea prezicerilor, prin adăugarea de date pentru antrenarea modelului folosit. Din rezultate, se poate observa că numărul de comentarii nu contribuie destul de mult în scorul de lizibilitate acordat unei secvențe de cod.

- O a doua modificare ar putea fi valorificarea unei metrici legate de redactarea testelor în sine și corelarea metricilor anterior definite cu numărul minim de teste necesare pentru a acoperi cât mai multe cazuri, inclusiv unora limită.

# Bibliografie

[1] microsoft. “GitHub - Microsoft/CodeBERT: CodeBERT.” GitHub, <https://github.com/microsoft/CodeBERT>. Last accessed: 12.5.2024

[2] PyPI. “Pylint,” February 25, 2024. https://pypi.org/project/pylint/. Last accessed: 12.5.2024

[3] Mark Otto, Andrew Fong. . Code Snippets: Insights and Readability. Retrieved from Kaggle https://www.kaggle.com/datasets/paakhim10/code-snippets-insights-and-readability. Licensed under MIT License. Last accessed: 12.5.2024