IV127

Závěrečná zpráva

Mykola Kaplenko

Duplicated Code Detector (AST + GPT2)

Účel

Detekce úseků duplikovaného kódu.

Algoritmus detekce duplikovaného kódu

1. Sestrojíme abstraktní syntaktický strom uživatelského kódu.

2. Rekurzivně extrahujeme zajímavé podstromy.

   Podstrom se považuje za zajímavý v případě, že kořen podstromu (*root node*) obsahuje atribut *body*. Libovolný *for* nebo *while* cyklus, *function definition*, *with statemen*t a libovolné další uzly, které obsahují vnořený kód, se tak považují za zajímavé a jsou extrahovány.

   Dale, zajímavým je každý podstrom, jehož kořen je dítětem (*child node*) již extrahovaného (obsahujícího atribut *body*) podstromu. Každý řádek extrahovaného *for* cyklu (nebo dalších již extrahovaných podstromu) se tak považuje za zajímavý podstrom.

   Příklad:

```
1        for i in range(10):
2            my_func()
3
4        for j in range(0, 20, 2):
5            my_func()
6
7        print(“Done!”)
```

   Uvedený kód obsahuje 5 zajímavých podstromů:

   - *for* cyklus (řádky 1-2)
   - *function call* (řádek 2)
   - *for* cyklus (řádky 4-5)
   - *function call* (řádek 5)
   - *function call* (řádek 7)

3. Dekódujeme každý extrahovaný zajímavý podstrom do *string* formátu.

4. Spustíme *tokenizer*. Úsek uživatelského kódu je rozdělen na jednotlivé tokeny, každý token je zakódován (embedded) jako 1024-dimenzionalní vektor.

5. Sada vektorů (vektor odpovídá zakódovanému tokenu) se dostává na vstup GPT-2.

   GPT-2 je neuronová síť (transformer) iniciálně nacvičená s účelem modelování variabilních textových datasetů (predikce následujícího tokenu v sekvenci tokenů). Použita verze GPT-2 byla "docvičená" (fine tuned) na *python* datasetech, které obsahují implementace kanonických algoritmů (quick sort, backtracking, generování číselných řad, atd.).

   GPT-2 transformuje poskytnuté vektory (tokeny). Ze sekvence transformovaných vektorů zvolíme poslední vektor - tento vektor reprezentuje význam celého zajímavého podstromu (úseku kódu).

6. Použijeme GPT-2 pro kódováníani každého extrahovaného podstromu (příklad v bodě 1). Výsledkem je 5 vektorů (každý vektor reprezentuje jeden podstrom).

7. Počítáme *pairwise cosine similarity* mezi získanými vektory. Vysledkem je *similarity matrix* rozměrnosti 5x5. Každý prvek matice udává míru podobnosti dvou příslušných podstromu.

8. Zvolíme největší a nejmenší prvek v *similarity matrix*. Normalizujeme *similarity matrix* tak, že prvky matice jsou v rozmezí 0-1.

9. Porovnáváme prvky *similarity matrix* se stanovenou prahovou hodnotou. Pokud prvek matice (číslo v rozmezí 0-1) je větší než prahova hodnota, pak odpovídající dvojice zajímavých podstromů odpovídá úsekům duplikovaného kódu.

10. Pokud duplikované úseky následují za sebou v uživatelském kódu - provedeme sloučení následujících po sobě úseků do většího úseku.

11. Výsledkem detekce je seznam (*list*) duplikovaných úseků a textova nápověda (*hint*) pro uživatele, v niž řádky duplikovaného kódu jsou označeny indexem duplikovaného úseků.

Úspěšnost detektoru

Celkový počet detekovaných chyb: **365**

Počet úloh, v nichž je detekováno alespoň 10 chyb: **6**

Precision: záleží na konkrétním nastavení prahové hodnoty - velice konzervativní hodnota 0.95 zaručuje, že v detekovaných případech se nevyskytují *false positive* výsledky (precision 1.0)

Redundant ifelse (AST)

Účel

Detekce použití *return true/false* místo *return condition*.

Příklad:

Uživatelský kód

```
1        if a > b:
2            return True
3        else:
4            return False
```

Korektní kód

```
1        return a > b:
```

Algoritmus detekce duplikovaného kódu

1. Sestrojime abstraktní syntaktický strom uživatelského kódu.
2. Chyba v uživatelském kódu je detekována na základě splnění následující podmínky, ktera genericky definuje šablonu hledané konstrukce:

```
1        if (
2            len(body_if) == 1
3            and isinstance(body_if[0], ast.Return)
4            and isinstance(body_if[0].value, ast.Constant)
5            and body_if[0].value.value is True
6
7            and len(orelse) == 1
8            and isinstance(orelse[0], ast.Return)
9            and isinstance(orelse[0].value, ast.Constant)
10           and orelse[0].value.value is False
11       ):
12           self.detected = True
```

Úspěšnost detektoru

Celkový počet detekovaných chyb: **3168**

Počet úloh, v nichž je detekováno alespoň 10 chyb: **13**

Precision: **1.0**

Redundant elif (AST)

Účel

Detekce použití redundantní *elif* podmínky v případě, kde by stačilo použít *else* podmínku.

Příklad:

Uživatelský kód

```
1        if cond_1:
2            ...
3        elif cond_2:
4            ...
5        elif cond_3:
6            ...
```

Korektní kód

```
1        if cond_1:
2            ...
3        elif cond_2:
4            ...
5        else:
6            ...
```

V případě, že `cond_1`, `cond_2 a cond_3` pokrývají všechna možná řešení (`cond_1 or cond_2 or cond_3` je vždy `True`) můžeme poslední `elif` podmínku bezpečně nahradit `else` podminkou (a neztratime při tom žádné řešení).

Algoritmus detekce duplikovaného kódu

1. Sestrojíme abstraktní syntaktický strom uživatelského kódu.

2. V abstraktním syntaktickém stromě postupně procházíme *if* a *elif* podminky, kontrolujeme při tom jaké operandy a jaké operátory v dané podmínce se používají. Pokud detekujeme použití operandů a operátorů, které pokrývají množinu všech možných řešení - můžeme bezpečně nahradit poslední `elif` podmínku `else` podmínkou.

   Příklad:

   Uživatelský kód

   ```
   1        if a > b:
   2            ...
   3        elif a < b:
   4            ...
   5        elif a == b:
   6            ...
   ```

Můžeme bezpečně nahradit kódem

```
1        if a > b:
2            ...
3        elif a < b:
4            ...
5        else:
6            ...
```

3. Sestrojíme nový modifikovaný abstraktní syntaktický strom a vrátíme uživateli jako nápovědu (*hint*)

Úspěšnost detektoru

Celkový počet detekovaných chyb: **309**

Počet úloh, v nichž je detekováno alespoň 10 chyb: **6**

Precision: **1.0**

Zdrojový kód:

Odevzdávárna

https://github.com/xkaple01/AdaptiveLearning