**פרוייקט סופי בקורס ישומי בינה מלאכותית לפתירת תקלות – דו"ח תוצאות**

1) מטרת הפרוייקט:

אלגוריתם בארינל מחשב דיאגנוזות אפשריות לקומפוננטות בהן יכולים להימצא באגים בקוד. בארינל לא מתחשב בתלות בין הקומפוננטות השונות, ומתייחס רק לקומפוננטות עצמן. מטרת הפרוייקט היא לבדוק כיצד התלות בין הקומפונטות משפיעה על אלגוריתם בארינל, והאם יש חשיבות לסדר הקומפוננטות. כלומר, האם כאשר קומפוננטה A מופיעה לפני קומפוננטה B, התוצאה תהיה זהה למצב בו כאשר קומפוננטה B מופיעה לפני קומפוננטה A. והאם ניתן לדעת באיזה מהסדרים הללו יופיע הבאג.

2) האלגוריתם:

**קלט**: תיקיית המסלולים שהטסטים עברו בהם עם כל קומפוננטות שכל טסט עבר (פלט של JAVA\_TRACER).

1. ניקח את רשימות כל הקומפוננטות שטסטים עברו בהן.

1.1. נשמור את הקומפוננטות כבודדים

1.2. נחבר ביחד כל שתי קומפוננטות צמודות בטסט לזוג

1.3. נחבר ביחד כל שלוש קומפוננטות צמודות בטסט לשלשה

1.4. נשמור את כל הקבוצות שמתקבלות ב (1.1) עם (1.2) ו (1.3) וקבוצה אחת.

* **נשים לב**: חיבור לזוגות/שלשות מבוצע עם חפיפה, וכאשר יש חשיבות לסדר בינהם. (למשל: עבור הרשימה A,B,C,D נקבל את הקבוצה: A,B,C,D,A\_B,B\_C,C\_D,A\_B\_C,B\_C\_D). צריכה להיות חפיפה בין הקבוצות כי יכול להיות באג בכל מעבר בין קומפוננטות, וכן יש חשיבות לסדר (A\_B לעומת B\_A) מכיוון שבאג בכיוון ריצה אחד, לא בהכרח מחייב באג בכיוון ריצה שני.

2. ניצור קובץ לפי הקלט של ברינל:

2.1. שמות כל הקומפוננטות (כאן מתבצעת גם בדיקת כפילות – אם יש שתי קומפוננטות שמופיעות כזוג/שלשה במקומות שונים, לפי הסימון כאן נדע שמדובר באותם הקומפוננטות).

2.2. הסתברות – ניתן עדיפות שווה לכל הקומפוננטות (0.1)

2.3. באגים – בקובץ החדש נגדיר קומפוננטה כבאג אם התת קומפוננטה שלה מופיעה כבאג בקובץ המקורי. (למשל – הקומפוננטה A\_B תופיעה כבאג אם A או B הופיעו כבאג במקור).

2.4. ניצור את רשימת כל הטסטים

2.5. ניצור רשימה של טסטים שעברו או לא לפי השמות שלהם

2.6. ניצור את רשימת כל הטסטים ועבור כל אחד מהם נרשום באיזה קומפוננטות הוא עבר (הבודדים, הזוגות, והשלשות) ונוסיף גם עבור כל טסט אם הוא עבר או לא.

3. נריץ על הקובץ שנוצר בסעיף את (2) את אלגוריתם ברינל המקורי.

**פלט**: קובץ עם דיאגנוזות אפשריות ונתונים נוספים, לפי אגוריתם בארינל המקורי.

3) הסבר על הניסויים:

לקחנו מספר רב של קבצי הרצה לבארינל מחולקים לשתי קבוצות (קבצים עם יחסית הרבה באגים - Xrefs, וקבצים עם יחסית מעט באגיםDominators - ) – כאשר אלו הם קבצים עם מידע מעודכן ואמיתי. עבור כל אחד מהקבצים הללו - הרצנו עליו את אלגוריתם SFL על מנת לקבל גם את הדיאגנוזות המקוריות לקבצים אלו.

על כל קובץ הרצנו את האלגוריתם שלנו. יצרנו קובץ חדש – בו הפכנו את כל הקומפוננטות המקוריות לזוגות ושלשות עם חפיפה ועם חשיבות לסדר, כפי שמפורט באלגוריתם. גם את הבאגים שינינו בהתאם לפי איך שכתוב באלגוריתם. קיבלנו עבור כל קובץ מקורי - קובץ הרצה חדש לאלגוריתם SFL עם מידע מעובד שיצרנו. הרצנו על כל הקבצים הללו את אלגוריתם SFL המקורי וקיבלנו דיאגנוזות חדשות.

חישבנו את הממוצעים של ה precision וה recall עבור כל אחת מהקבוצות (קצת באגים/הרבה באגים) בכל אחד מהמקרים (האלגוריתם המקורי/אלגוריתם זוגות-שלשות), והשוונו בין התוצאות.

על מנת לקבל את גודל כל הרכיבים, הוספנו הדפסות ל- console של גדלי המערכים עם הנתונים.

4) התוצאות שקיבלנו:

**תוצאות:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File name** | **Single count**  **(=old sum of all)** | **Double count** | **Triplet Count** | **New sum of all** | **Old bug count** | **New bug count** | **Old size of diagnosis** | **New size of diagnosis** |
| dominators | | | | | | | | |
| CVE-2017-7623 | 38 | 49 | 58 | 145 | 2 | 16 | 38 | 65 |
| CVE-2017-7939 | 37 | 48 | 56 | 141 | 5 | 29 | 36 | 60 |
| CVE-2017-9204 | 36 | 46 | 55 | 137 | 2 | 13 | 35 | 74 |
| CVE-2017-9205 | 37 | 48 | 58 | 143 | 2 | 12 | 37 | 85 |
| CVE-2017-9206 | 39 | 52 | 65 | 156 | 2 | 14 | 38 | 99 |
| CVE-2017-9207 | 37 | 49 | 60 | 146 | 2 | 12 | 37 | 64 |
| **average** | 37.33333 | 48.66667 | 58.66667 | 144.6667 | 2.5 | 16 | 36.83333 | 74.5 |
| xrefs | | | | | | | | |
| CVE-2015-8784 | 707 | 1420 | 2123 | 4250 | 55 | 387 | 573 | 2442 |
| CVE-2016-8691 | 349 | 569 | 704 | 1622 | 49 | 508 | 229 | 1045 |
| CVE-2016-8692 | 246 | 368 | 432 | 1046 | 51 | 421 | 224 | 628 |
| CVE-2016-8887 | 416 | 583 | 719 | 1718 | 49 | 383 | 380 | 910 |
| CVE-2016-10250 | 400 | 557 | 698 | 1655 | 50 | 432 | 355 | 911 |
| CVE-2016-10251 | 471 | 574 | 672 | 1717 | 192 | 1107 | 514 | 2192 |
| CVE-2016-10271 | 647 | 893 | 1086 | 2626 | 17 | 142 | 29 | 29 |
| CVE-2016-10272 | 687 | 1146 | 1486 | 3319 | 17 | 184 | 43 | 46 |
| CVE-2017-7452 | 514 | 693 | 842 | 2049 | 28 | 162 | 418 | 819 |
| CVE-2017-7453 | 840 | 1360 | 1866 | 4066 | 15 | 108 | 700 | 1352 |
| CVE-2017-7454 | 1025 | 1746 | 2456 | 5227 | 34 | 276 | 891 | 7038 |
| CVE-2017-7623 | 650 | 952 | 1211 | 2813 | 49 | 330 | 583 | 1118 |
| CVE-2017-7939 | 578 | 801 | 986 | 2365 | 61 | 396 | 507 | 922 |
| CVE-2017-7962 | 1001 | 1676 | 2331 | 5008 | 53 | 436 | 681 | 1868 |
| CVE-2017-9204 | 547 | 761 | 967 | 2275 | 34 | 222 | 454 | 976 |
| CVE-2017-9205 | 565 | 788 | 997 | 2350 | 44 | 283 | 483 | 1287 |
| CVE-2017-9206 | 545 | 754 | 953 | 2252 | 28 | 179 | 460 | 1269 |
| CVE-2017-9207 | 591 | 849 | 1110 | 2550 | 41 | 280 | 499 | 908 |
| **average** | 598.8333 | 916.1111 | 1202.167 | 2717.111 | 48.16667 | 346.4444 | 445.7222 | 1431.111 |

**החישובים**: (הסדר בין התוצאות תואם לקבצים)

**Only-one:**

|  |  |  |
| --- | --- | --- |
|  | **Dominators** | **Xrefs** |
| Precision | 0.0526315789473684  0.11111111111111106  0.05714285714285712  0.05405405405405403  0.05844535359438923  0.05405405405405403 | 0.07275292220878533  0.20087336244541415  0.21428571428571364  0.1143423959601419  0.12268188302425041  0.4880528723945074  0.0  0.0  0.014354066985645822  0.01310315458446613  0.022509531987514052  0.07433628318584011  0.09270216962524573  0.04867330851028112  0.03744493392070454  0.060374149659863464  0.03761894224385896  0.07615230460921778 |
| **average** | **0.0645731681506389** | **0.0939032219795249** |
| Recall | 0.0263157894736842  0.022222222222222213  0.02857142857142856  0.027027027027027015  0.029222676797194615  0.027027027027027015 | 0.0026455608075921996  0.004099456376437024  0.004201680672268897  0.0023491609547917494  0.0024536376604850055  0.002541942043721406  0.0  0.0  0.0005126452494873507  0.0008735436389644085  0.0009759128692791455  0.0015170670037926543  0.0015197076987745197  0.0009196301479193071  0.0011013215859030743  0.0014397804576375894  0.0013435336515663918  0.0018573732831516545 |
| **average** | **0.0267310285197639** | **0.00168621967232068** |

**Triplet + Couple + Only-one:**

|  |  |  |
| --- | --- | --- |
|  | **Dominators** | **Xrefs** |
| Precision | 0.030769230769230806  0.06666666666666674  0.027359781121751067  0.10588235294117665  0.10285227756492138  0.031250000000000035 | 0.2019741837509597  0.3180322966507086  0.3421933863541202  0.19630295390170352  0.21260767423649285  0.6317651956702613  0.0  0.0  0.007326007326007361  0.007356998344675395  0.04924906194621428  0.07231968074727411  0.05506116973027441  0.05663393676255452  0.017418032786885477  0.10774485657724028  0.07121212121212239  0.04225778802388167 |
| **average** | **0.0607967181772909** | **0.132747519112298** |
| Recall | 0.0019230769230769254  0.002298850574712646  0.002104598547827005  0.008823529411764721  0.007398284984491892  0.0026041666666666696 | 0.0010437942312710971  0.0006487115246957488  0.0008166523688477523  0.0005136171790790175  0.0004922768724648144  0.0006318131457750188  0.0  0.0  4.5222267444489875e-05  6.812035504329068e-05  0.00029345792220175437  0.00022127729701799565  0.00013963171453647185  0.00013135266705704886  7.845960714813274e-05  0.0003947144415218592  0.0004006545905987308  0.0001515276264788604 |
| **average** | **0.00419208451808997** | **0.000337293545065671** |

5) סיכום:

קל לראות, כי אכן יש הבדל בין התוצאות המתקבלות באלגוריתם בארינל המקורי לעומת התוצאות המתקבלות באלגוריתם שלנו. עבור קבוצת ה Dominators גם ה Precision וגם ה Recall הממוצעים קטנים יותר. עבור קבוצת ה Xrefs ה Precision הממוצע גדול יותר וה Recall הממוצע קטן יותר. תוצאות אלו חשובות, שכן אנו רוצים לדעת עד כמה האלגוריתם מחזיר דיאגנוזות אמינות. נסיק כי האלגוריתם אכן משפיע על תוצאת הדיאגנוזה (ועל האמינות שלה), ואכן יש חשיבות לסדר בין הקומפוננטות בזמן הרצת בארינל.

**נספחים:**

1) איך מריצים את הקוד ומשחזרים את הניסויים:

עבור הרצה עם קובץ הכולל מידע מלא ואמיתי: (כמו בניסויים לפרוייקט זה)

1. בקובץ chenge\_data\_file\_to\_triplet\_data\_file.py שנה את שם הקובץ שברצונך להריץ עליו את האלגוריתם ואת מיקום התיקייה שבה הוא נמצא, וכן את מיקום תיקיית הפלט (שורות 11,17,102).
2. הרץ את הקובץ chenge\_data\_file\_to\_triplet\_data\_file.py.
3. שנה את שם הקובץ עליו תרצה להריץ בקובץ run\_sfl\_code.py וכן את שמות התיקיות פלט/קלט (שורות 6,9,14).
4. הרץ את הקובץ run\_sfl\_code.py.

עבור "הרצה מאפס" של האלגוריתם:

1. הרץ את פרוייקט JAVA\_TRACER

2. התאם את מיקום קבצי הפלט של 1 (תיקיית DebuggerTests) לתיקיות בקוד הפרוייקט (בקובץ – makeTableFromTestsTraces).

\*אם רוצים ליצור קבצים עבור האלגוריתם המקורי יש להוסיף את המיקום בקובץ makeTableFromTestsTracesOnlyOne ואז לשנות ב Main לקובץ הנכון (זה נמצא בהערה – צריך להוריד אותה ולשים את השורה שאחריה בהערה)

3. שנה את מיקום תיקיית surefire-reports למיקום הנכון במחשבך בקובץ readTestsResults.

4. הרץ את הפרוייקט על הקבצים המתקבלים ב - 1 ועל קבצי הפלט של הטסטים (סעיפים 2 ו-3) – בשלב זה מריצים את הקוד שלנו, ונוצר קובץ קלט חדש לאלגוריתם בארינל.

5. הרץ את SFL-diagnoser על הפלט של סעיף 4.

2) קבצי תוצאות:

ההרצות החדשות:

בתיקיית final results files on real data ב- git נמצאים כל קבצי הקלט והפלט. תוספת המילה single מציינת כי מדובר בקבצי קלט-פלט לאלגוריתם המקורי הרץ על קומפוננטות בודדות, שאר הקבצים הם לאלגוריתם שלנו שרץ על גבי יחידים-זוגות-שלשות (והמילה transformed מצורפת לשם הקובץ שעבר שינוי לאלגוריתם החדש).

ההרצות הקודמות שביצענו לפרוייקטים:

כל קבצי הפלט של האלגוריתם שלנו (קלט לבארינל עבור התוספת שלנו ועבור המקורי) – נמצאים ב- git בתיקייה out-files. כל קבצי התוצאות של הרצת בארינל – נמצאים ב- git בתיקייה results.

3) הקוד עצמו:

קישור ל git: <https://github.com/elior19/double_tracer>