**Dynamic Analysis Features Extraction For Fault Prediction**

**פרויקט בקורס יישומי בינה מלאכותית באיתור תקלות**

**מגישים: נבו משיח וקרן פוטש**

1. **מוטיבציה**

כחלק מהשאיפה לבנות מודל סיווג שיוכל לחזות קבצים שיש בהם שגיאות , ניתן לקבל על כל קובץ מידע סטטי כגון – מספר שורות בקובץ, מספר פונקציות בקובץ וכו' וכן מידע דינמי – הרצת חבילת הקוד שבודקת את הקבצים (tests) ולחזות האם הקובץ תקול או לא.

בפרויקט שלנו התמקדנו בניתוח המידע הדינמי שניתן לחלץ ע"י הרצת חבילת בדיקות התוכנה ובבניית המאפיינים הדינמיים עבור כל קובץ.

1. **תיאור המודל**

כפי שניתן לראות בתמונה הנ"ל המודל בנוי מחמישה שלבים עיקריים:

**שלב ראשון -** הרצת חבילות הבדיקה של הקבצים באמצעות Jcov Hiveוקבלת קובץ xml ו-html המכילים את כל הנתונים שנאספו במהלך הריצה.

**שלב שני -** קבלת קובץ ה-xml (או ה-html) והוצאת כל המידע הרלוונטי על הקבצים מחולק לפי הבלוקים והפונקציות הקיימים בכל קובץ (הסבר על המידע שחולץ מהקבצים מצורף בנספח א')

**שלב שלישי –** ניתוח המידע שחולץ מקובץ ה-xml, כך שעבור כל קובץ ננתח את המידע בהסתכלות הן לפי בלוקים והן לפי פונקציות.

**שלב רביעי –** יצירת פיטצ'רים (מאפיינים) שונים לכל קובץ על סמך המידע שנותח בשלב הקודם.

**שלב חמישי –** בניית מודל מסוג Random Forest באמצעות המאפיינים שנוצרו, והתאמת ה-class לכל קובץ לפי נתונים שהתקבלו על הקבצים – האם הקובץ תקול או תקין ("תקול" יקבל סיווג של "1" ו"תקין" יקבל סיווג של "0".

1. **הסבר על הפיט'צרים**

חילקנו את הפיט'צרים למספר קבוצות עיקריות בחלוקה לפי בלוקים ופונקציות (רשימת הפיטצ'רים עבור כל קבוצה מתוארת בנספח ב'):

בניתוח לפי בלוקים:

1. **Basic aggregate-functions features for block hit count list –**

פונקציות aggregation שמקבלות רשימה של hit count עבור כל בלוק ומחזירות:

ממוצע (mean), חציון (median), שונות (variance) ,סטיית תקן(std), הערך המינימאלי (min, יכול להצביע על מיעוט בבדיקות), הערך המקסימלי (max, יכול להצביע על לולאות או בדיקות יתרות) , מספר הערכים השונים (distinct\_values, יכול להצביע על מסלולי ריצה שונים) ואחוז המופעים של 0 (zero\_value\_percent, קטע קוד ללא שום בדיקה).

**סה"כ – 8 פיטצרים.**

1. **‘Weight’ for block hit count list –**

נחשב את פונקציות ה-aggregate ע"י נתינת משקל לכל בלוק. ניצור את משקלים לפי הכפלת כל ערך של block hit count באורך הבלוק, וחלוקה באורך הבלוק המקסימאלי בקובץ.

**סה"כ – 8 פיטצרים.**

1. **’Normalize’ for block hit count list –**

נחשב את פונקציות ה-aggregate ע"י נירמול הבלוקים. ניצור את הנירמול ע"י חלוקת כל ערך של block hit count ב-hit count methenter block (ערך ה- block hit count של הבלוק הראשון בפונקציה).

**סה"כ – 8 פיטצרים.**

1. **‘Filter’ for block hit count list –**

נוכל לפלטר את הבלוקים שלנו ע"פ סוגים שונים. נתבונן ב- cond, goto, methenter, exit ו-inner (הבלוקים בפונקציה שהם לא methenter או exit). נחשב את פונקציות ה-aggregate לפי ה- filters השונים.

**סה"כ 40 פיטצ'רים.**

1. **Loop for block hit count list –**

כל בלוק שערך ה- hit count שלו גדול מערך ה- hit count של הבלוק הראשון בפונקציה – משתתף בלפחות לולאה אחת. נשלב מסקנה זו ונבדוק את אחוז הבלוקים שמשתתפים בלולאה בשילוב עם הפילטרים.

**סה"כ 5 פיטצ'רים.**

בניתוח לפי פונקציות:

1. **Basic aggregate-functions features for function hit count list –**

נקבע את ערך ה- function hit count להיות ערך ה- hit count של הבלוק הראשון בפונקציה.

בשילוב עם פונקציות האגריגציה הבסיסיות (עבור בלוקים) נקבל את הפיצ'רים הבאים עבור ממוצע, חציון, שונות, סטיית תקן, ערך מינימום ומקסימום, מספר ערכים שונים, ואחוז המופעים של הספרה אפס. ערכים אלו יהוו לנו את פונקציות האגריגציה הבסיסיות של פיצ'רים מוכווני פונקציה.

**סה"כ 8 פיטצ'רים.**

1. **‘Weight’ for function hit count list –**

נגדיר שני סוגי משקולות:

1. נכפיל כל ערך function hit count באורך הפונקציה, ונחלק באורך הפונקציה המקסימאלי בקובץ.

2. נכפיל כל ערך function hit count במספר הבלוקים שיש בפונקציה, ונחלק במספר הבלוקים המקסימאלי של פונקציה בקובץ**.**

כך נחשב את פונקציות ה-aggregate (מוכווני פונקציות) ע"י נתינת משקל לכל פונקציה.

1. **‘Filter’ for function hit count list –**

נוכל לפלטר את הפונקציות שלנו ע"פ סוגים שונים. נתבונן ב- private, public, protected ו- constructor functions. נחשב את פונקציות ה-aggregate לפי הפילטרים של הפונקציות.

1. **ניתוח ה-class - נתונים חסרים**

המודל נבנה עבור 4 גרסאות של קבצים:

* קבוצת האימון: גרסה 1.7, גרסה 1.7-rc2, גרסה 1.8
* קבוצת המבחן: גרסה 1.8-rc2

לאחר קבלת הקבצים שאכן תקולים, התאמנו עבור כל גרסה את הקבצים שעשינו עליהם את שלב ה- Feature Extraction לבין רשימת הקבצים התקולים וקיבלנו את התוצאות הבאות:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Percentage of bugged files** | **Total files** | **Total bugged files** | **#Bugged filed with feature extraction** | **#Bugged files** | **Version** |  |
|  | 2845 | 96 | 74 | 104 | 1.7 | **Train** |
| 3 | 4 | 1.7-rc2 |
| 19 | 92 | 1.8 |
|  | 572 | 4 | 4 | 6 | 1.8-rc2 | **Test** |

ניתן לראות כי רק כ-3% מקובץ האימון וכ-1% מקובץ המבחן מסווגים כתקולים, דבר אשר מקשה על בניית מודל חיזוי שייתן תוצאות טובות.

1. **הסבר על המחלקות והתיקיות המרכיבות את הפרויקט:**

**תיקיות:**

* **buggedFiles** - תיקייה שמכילה טבלה ובה יש לשים את רשימת הקבצים התקולים עבור כל גרסה.
* **jcov\_hive –** תיקייה ובה שמורים כל הקבצים שנאספו בשלב הראשון ע"י הרצת jcov hive (קבצי xml ו-html)
* **temporaryFiles –** תיקייה בה נשמרים כל הקבצים הזמניים הנדרשים עבור הרצת המודל (טבלאות csv):
  + הקבצים שנוצרו לאחר שלב ה-parsing: parsing\_table+version\_name
  + הקבצים שנוצרו לאחר שלב ה-features extraction (עדיין ללא ה-class): features\_table+version\_name
  + הקבצים המעובדים עבור הרצת המודל ובהם כל הנתונים על קבוצת האימון וקבוצת המבחן – data\_train, data\_test

**מחלקות:**

* **xml\_parse.py –** מחלקה זו אחראית על שלב 2 – היא מקבלת קובץ xml המכיל את הנתונים מהרצת חבילות הבדיקה על הקבצים ומייצרת טבלת csv שבה נשמרים כל הנתונים שישמשו לשלב ה-feature extraction. המחלקה כוללת 2 פונקציות:

1. **def parsing\_xml(version\_name, perform\_new\_parse=True):**

פונקציה זו אחראית על סידור הנתונים בצורה קריאה ונוחה למשתמש, לאחר עיבוד כל הנתונים, הם נשמרים בטבלה בשם: 'parsing\_table\_' + version\_name + '.csv' בתוך תיקיית - Temporary Files.

הפונקציה מקבלת שני פרמטרים:

1) version\_name – string שמתאר מהי הגרסה הנוכחית שיש לפרסר.

2) perform\_new\_parse – משתנה Boolean, True – במידה ורוצים לבצע את תהליך הפירסור מההתחלה, False- במידה ורוצים לחסוך את ביצוע תהליך הפירסור מההתחלה ורוצים להשתמש בטבלה שנוצרה בריצה הקודמת.

1. **def add\_to\_table(class\_path, method\_name, , …., is\_goto, table) :**

פונקציה המקבלת את כל הנתונים שנאספו עבור בלוק מסויים בתהליך הפירסור ושומרת אותם בצורה מסודרת בטבלה. עבור כל בלוק מתבצעת קריאה אחת לפונקציה זו, ונשמרת שורה אחת בלבד בטבלה. (פירוט על הטבלה בנספח א')

* **feature\_extraction.py** – מחלקה זו אחראית על שלבים 3 ו-4, היא אחראית על עיבוד המידע לאחר שלב הפירסור, ויצירת פיטצ'רים לפי בלוקים ופונקציות עבור הקבצים השונים.

מחלקה זו כוללת שלוש פונקציות:

1. **def create\_block\_features(version\_name, data, perform\_new\_feature\_extraction)**: פונקציה זו מכילה את רוב הקוד עבור יצירת הפיטצ'רים, ומכילה שני חלקים (region)
   1. Collecting data - סידור כל הנתונים בצורה נוחה לחישוב וניתוח
   2. Building features – בניית הפיטצ'רים עבור כל קובץ

לאחר יצירת הפיטצ'רים הם נשמרים בטבלה בשם: ' features\_table\_' + version\_name + '.csv' בתוך תיקיית - Temporary Files.

הפונקציה מקבלת שלושה פרמטרים:

1) version\_name – string שמתאר מהי הגרסה שיוצרים עבורה את הפיטצ'רים.

2) data – במידה ובשלב הקודם בוצע תהליך הפירסור במלואו (parsing=yes), בסוף התהליך נוצר אובייקט מסוג data Frame אשר ניתן לעבוד לפיו.

3) perform\_new\_feature\_extraction – משתנה Boolean, True – במידה ורוצים לבצע את תהליך בניית הפיטצ'רים מההתחלה, False- במידה ורוצים לחסוך את ביצוע תהליך זה מההתחלה ורוצים להשתמש בטבלה שנוצרה בריצה הקודמת.

1. **def create\_inner\_bl\_hc\_list(bl\_hc\_list, bl\_type\_list):** פונקציה שמחזירה רק את ה-hit count של בלוקים פנימיים - inner block (ללא בלוקים שהם התחלה/סוף של פונקציה)
2. **def create\_filtered\_list(filt, source\_list, factor\_list):** פונקציה אשר מסננת רשימה של hit count לפי קריטריון מסויים (למשל רק hit count של בלוקים מסוג cond)

* **parse\_results.py –** מחלקה זו עוברת על כל הקבצים שנמצאו תקולים ובונה מילון שבו עבור כל גרסה רשימה עם כל הקבצים התקולים (בעלי סיומת מסוג .java) של אותה גרסה. במחלקה זו קיימת פונקציה אחת בשם - def get\_bugged\_files() אשר מחזירה את המילון שנבנה.
* **main.py**  - המחלקה בה מריצים את המודל המלא:תחילה מעבדים את המידע שהתקבל ומכינים את קבצי האימון והמבחן להרצת המודל, לאחר כן מריצים את המודל לפי קבוצות הפיטצ'רים שמתוארות בנספח ב' ושומרים את כלל התוצאות בקובץ: **Model result.csv.**

המחלקה מכילה את הפונקציות הבאות:

1. **def prepare\_data(training,versions\_array, bugged\_paths, dic\_versions):**

הפונקציה אחראית על הכנת הנתונים, ומקבלת את הפרמטרים הבאים:

1. training – במידה והוא שווה לTRUE הפונקציה מכינה את נתוני האימון, במידה והוא מקבל FALSE הפונקציה מכינה את נתוני המבחן.
2. version\_array – רשימת הגרסאות שעבורן נכין את הנתונים.
3. bugged\_paths – רשימת כל הקבצים התקולים לפי גרסאות.
4. dic\_version – מילון שממפה בין השמות של הגרסאות שהתקבלו מקבצי ה- xml והשמות של הגרסאות שהתקבלו עבור הקבצים התקולים.
5. **def build\_model(x\_train, y\_train, x\_test, y\_test):** פונקציה שאחראית על בניית מודל הלמידה מסוג Random Forest, אימון המודל לפי קבצי האימון (x\_train, y\_train), בדיקת המודל ע"י קבצי המבחן (x\_test, y\_test) והדפסת התוצאות שהתקבלו לפי המדדים: Accuracy, AUC area, Precision, Recall.

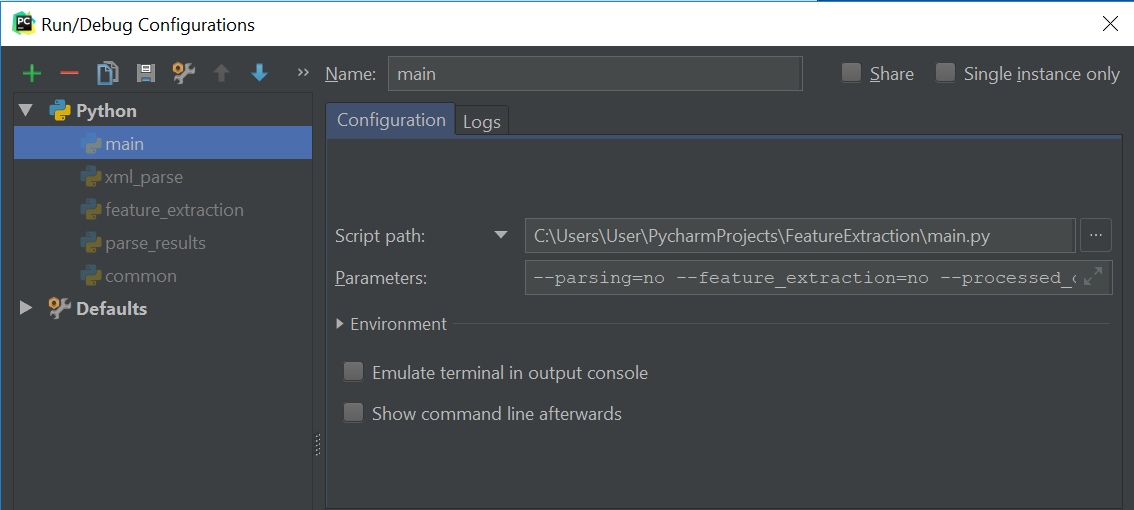
* train\_files – רשימה ובה כל הגרסאות שיכילו את קבוצת האימון, לדוגמא:
  + train\_files = ['1\_7\_jcov', '1\_7\_rc2\_jcov', '1\_8\_jcov']
* test\_files – רשימה ובה כל הגרסאות שיכילו את קבוצת המבחן, לדוגמא:
  + test\_files = ['1\_8\_rc2\_jcov']

1. **תיאור הרצת המודל**

בשביל להריץ את המודל צריך להזין ב-Run/Decug Configuration את הפרמטרים הבאים:

1. parsing –
   1. במידה ורוצים לבצע את שלב ה-parsing מההתחלה יש להזין: parsing=yes .
   2. במידה ורוצים להשתמש בטבלאות המוכנות ) 'parsing\_table\_' + version\_name + '.csv' ( שנוצרו לאחר שלב זה יש להזין: parsing=no.
2. feature\_extraction –
   1. במידה ורוצים לבצע את שלב יצירת הפיט'צרים מההתחלה יש להזין: feature\_extraction=yes .
   2. במידה ורוצים להשתמש בטבלאות המוכנות ) ' feature \_table\_' + version\_name + '.csv' ( שנוצרו לאחר שלב זה יש להזין: feature\_extraction=no .
3. process\_data –
   1. במידה ולא רוצים להשתמש במידע המעובד (train\_data.csv, test\_data.csv) ורוצים לבצע את התהליך המלא או את חלקו מההתחלה (כלומר לעבד את הנתונים) יש להזין: process\_data=yes .
   2. במידה ורוצים להשתמש בטבלאות המוכנות (train\_data.csv, test\_data.csv) שנוצרו בסוף תהליך עיבוד הנתונים יש להזין : process\_data=no.

דוגמא להזנת הנתונים:



לאחר מכן ניתן ללחוץ על Run main ויבנו המודלים הבאים (לפי הסדר הנ"ל):

1. בניית המודל לפי כל הפיטצ'רים שנוצרו (Group name – all features)
2. בניית המודל לפי הפיטצ'רים שמתייחסים לבלוקים (Group name – block features)
3. בניית המודל לפי הפיטצ'רים שמתייחסים לפונקציות Group name – function features))
4. בניית המודל לפי קבוצות הפיטצ'רים כאשר כל פעם:
   1. בונים את המודל רק לפי אותה קבוצה (Group name – with \_\_\_\_ )
   2. בונים את המודל ללא אותה קבוצה (Group name – without \_\_\_ )

קבוצות הפיט'צרים מתוארות בחלק 3 ובנספח ב' (קבוצה 1 – פיטצ'רים 1-8, קבוצה 2 פיטצ'רים 9-16,... קבוצה אחרונה – פיטצ'רים 94-125). ה-Group name מתאר את המודל שנבנה עבור טבלת התוצאות.

לבסוף כלל התוצאות ישמרו בקובץ Model version.csv

1. **תוצאות הניסוי**
   1. **מדדים:**
      1. Accuracy
      2. Auc area
      3. Precision
      4. Recall
   2. **תוצאות**:

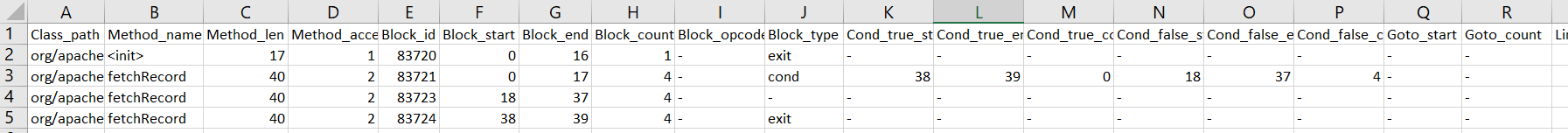
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Group Name** | **Accuracy** | **Auc Area** | **Precision** | **Recall** |
| **all features** | 0.966783 | 0.610915 | 0.058824 | 0.25 |
| **block features** | 0.97028 | 0.612676 | 0.066667 | 0.25 |
| **function features** | 0.97028 | 0.612676 | 0.066667 | 0.25 |
| **with (1, 8)** | 0.965035 | 0.610035 | 0.055556 | 0.25 |
| **without(1, 8)** | 0.97028 | 0.612676 | 0.066667 | 0.25 |
| **with (9, 16)** | 0.972028 | 0.613556 | 0.071429 | 0.25 |
| **without(9, 16)** | 0.965035 | 0.610035 | 0.055556 | 0.25 |
| **with (17, 24)** | 0.973776 | 0.614437 | 0.076923 | 0.25 |
| **without(17, 24)** | 0.97028 | 0.612676 | 0.066667 | 0.25 |
| **with (25, 64)** | 0.963287 | 0.609155 | 0.052632 | 0.25 |
| **without(25, 64)** | 0.966783 | 0.486796 | 0 | 0 |
| **with (65, 69)** | 0.977273 | 0.492077 | 0 | 0 |
| **without(65, 69)** | 0.968531 | 0.611796 | 0.0625 | 0.25 |
| **with (70, 77)** | 0.972028 | 0.489437 | 0 | 0 |
| **without(70, 77)** | 0.972028 | 0.613556 | 0.071429 | 0.25 |
| **with (78, 93)** | 0.975524 | 0.491197 | 0 | 0 |
| **without(78, 93)** | 0.972028 | 0.613556 | 0.071429 | 0.25 |
| **with (94, 125)** | 0.963287 | 0.485035 | 0 | 0 |
| **without(94, 125)** | 0.968531 | 0.611796 | 0.0625 | 0.25 |

* 1. ההשפעה של הפיטצ'רים:

1. **מסקנות**

**נספח א' – חילוץ המידע מקבצי ה-Xml:**

לאחר פירסור המידע מקבצי ה-xml מתקבלת הטבלה הבאה:



הסבר על השדות שבטבלה:

1. Class\_path – הכתובת המלאה של כל קובץ
2. Method\_name – שם הפונקציה
3. Method\_len – אורך הפונקציה (מספר שורות קוד)
4. Method\_flags – פונקציה מסוג Private/protected/publics
5. Block\_id- מזהה חד חד ערכי לכל בלוק
6. Block\_start – באיזה שורה בקוד מתחיל הבלוק
7. Block\_end – באיזה שורה בקוד מסתיים הבלוק
8. Block\_count – כמות הפעמים שחבילת הבדיקה נכנסה לבלוק.
9. Block\_type- הסוג של הבלוק - Cond\_true, Cond\_false, Goto
10. Cond\_true\_star – השורה שבה התחיל תנאי מסוג True
11. Cond\_true\_end – השורה שבה נגמר תנאי מסוג True
12. Cond\_true\_count – כמות התנאים שהם מסוג True בבלוק
13. Cond\_false\_start - השורה שבה התחיל תנאי מסוג False
14. Cond\_false\_end - השורה שבה נגמר תנאי מסוג False
15. Cond\_false\_count – כמות התנאים שהם מסוג False בבלוק
16. Goto\_start – השורה שבה התחיל בלוק מסוג Goto
17. Goto\_count – כמות הפעמים שהופיע Goto בבלוק
18. Line\_in\_file- השורה בקובץ שבה הבלוק נמצא

**נספח ב' – רשימת הפיט'צרים המלאה בחלוקה לפי קבוצות**

* Columns in feature table – מספרי העמודות של הקבוצת פיט'צרים בטבלת ה- feature\_table

**בניתוח לפי בלוקים:**

|  |  |  |
| --- | --- | --- |
| **Columns in feature table** | **List of features** | **Group** |
| **1-8** | Hc\_block\_mean, Hc\_block\_std, Hc\_block\_median, Hc\_block\_var, Hc\_block\_min, Hc\_block\_min, Hc\_block\_max, Hc\_block\_unique, Hc\_block\_per\_zero | **Basic aggregate-functions features for block hit count list:** |
| **9-16** | Weight\_hc\_block\_mean, Weight\_hc\_block\_std, Weight\_hc\_block\_median, Weight\_hc\_block\_var, Weight\_hc\_block\_min, Weight\_hc\_block\_max, Weight\_hc\_block\_unique, Weight\_hc\_block\_per\_zero | **‘Weight’ for block hit count list:** |
| **17-24** | Normalized\_hc\_block\_mean, Normalized\_hc\_block\_std, Normalized\_hc\_block\_mid, Normalized\_hc\_block\_var, Normalized\_hc\_block\_min, Normalized\_hc\_block\_max, Normalized\_hc\_block\_unique, Normalized\_hc\_block\_per\_zero | **‘Normalize’ for block hit count list:** |
| **25-64** | Cond\_hc\_block\_mean, Cond\_hc\_block\_std, Cond\_hc\_block\_median, Cond\_hc\_block\_var, Cond\_hc\_block\_min, Cond\_hc\_block\_max, Cond\_hc\_block\_unique, Cond\_hc\_block\_per\_zero, Goto\_hc\_block\_mean, Goto\_hc\_block\_std, Goto\_hc\_block\_median, Goto\_hc\_block\_var, Goto\_hc\_block\_min, Goto\_hc\_block\_max, Goto\_hc\_block\_unique, Goto\_hc\_block\_per\_zero, Methenter\_hc\_block\_mean, Methenter\_hc\_block\_std, Methenter\_hc\_block\_median, Methenter\_hc\_block\_var, Methenter\_hc\_block\_min, Methenter\_hc\_block\_max, Methenter\_hc\_block\_unique, Methenter\_hc\_block\_per\_zero, Exit\_hc\_block\_mean, Exit\_hc\_block\_std, Exit\_hc\_block\_median, Exit\_hc\_block\_var, Exit\_hc\_block\_min, Exit\_hc\_block\_max, Exit\_hc\_block\_unique, Exit\_hc\_block\_per\_zero, Inner\_hc\_block\_mean, Inner\_hc\_block\_std, Inner\_hc\_block\_median, Inner\_hc\_block\_var, Inner\_hc\_block\_min, Inner\_hc\_block\_max, Inner\_hc\_block\_unique, Inner\_hc\_block\_per\_zero | **‘Filter’ for block hit count list:** |
| **65-69** | Per\_blocks\_in\_loop, Cond\_per\_blocks\_in\_loop, Goto\_per\_blocks\_in\_loop, Methenter\_per\_blocks\_in\_loop, Exit\_per\_blocks\_in\_loop | **Loop for block hit count list:** |

**בניתוח לפי פונקציות:**

|  |  |  |
| --- | --- | --- |
| **Columns in feature table** | **List of features** | **Group** |
| **70-77** | Hc\_fun\_std, Hc\_fun\_median, Hc\_fun\_var, Hc\_fun\_min, Hc\_fun\_max, Hc\_fun\_unique, Hc\_fun\_per\_zero | **Basic aggregate-functions features for function hit count list:** |
| **78-93** | Fun\_len\_weight\_hc\_fun\_mean, Fun\_len\_weight\_hc\_fun\_std, Fun\_len\_weight\_hc\_fun\_mid, Fun\_len\_weight\_hc\_fun\_var, Fun\_len\_weight\_hc\_fun\_min, Fun\_len\_weight\_hc\_fun\_max, Fun\_len\_weight\_hc\_fun\_unique, Fun\_len\_weight\_hc\_fun\_per\_zero, Num\_of\_bl\_weight\_hc\_fun\_mean, Num\_of\_bl\_weight\_hc\_fun\_std, Num\_of\_bl\_weight\_hc\_fun\_mid, Num\_of\_bl\_weight\_hc\_fun\_var, Num\_of\_bl\_weight\_hc\_fun\_min, Num\_of\_bl\_weight\_hc\_fun\_max, Num\_of\_bl\_weight\_hc\_fun\_unique, Num\_of\_bl\_weight\_hc\_fun\_per\_zero | **‘Weight’ for function hit count list:** |
| **94-125** | Public\_hc\_fun\_mean, Public\_hc\_fun\_std, Public\_hc\_fun\_mid, Public\_hc\_fun\_var, Public\_hc\_fun\_min, Public\_hc\_fun\_max, Public\_hc\_fun\_unique, Public\_hc\_fun\_per\_zero, Private\_hc\_fun\_mean, Private\_hc\_fun\_std, Private\_hc\_fun\_mid, Private\_hc\_fun\_var, Private\_hc\_fun\_min, Private\_hc\_fun\_max, Private\_hc\_fun\_unique, Private\_hc\_fun\_per\_zero, Protected\_hc\_fun\_mean, Protected\_hc\_fun\_std, Protected\_hc\_fun\_mid, Protected\_hc\_fun\_var, Protected\_hc\_fun\_min, Protected\_hc\_fun\_max, Protected\_hc\_fun\_unique, Protected\_hc\_fun\_per\_zero, Constructor\_hc\_fun\_mean, Constructor\_hc\_fun\_std, Constructor\_hc\_fun\_mid, Constructor\_hc\_fun\_var, Constructor\_hc\_fun\_min, Constructor\_hc\_fun\_max, Constructor\_hc\_fun\_unique, Constructor\_hc\_fun\_per\_zero | **‘Filter’ for function hit count list:** |