Enhancing Mobile Security through Machine Learning: A Study on Android Malware Detection

Authors:
Moti Dahari308212570
Amit Kubani 204804488

Machine Learning-based Approach for Android Malware Detection: An Evaluation of Performance and Limitations

הקדמה

- הפרויקט שלנו נועד לסווג אפליקציות אנדרואיד(APK) כזדוניות או שפריות. השתמשנו במערך נתונים של 4591 אפליקציות והרצנו עליו מודלים שונים של למידת מכונה.
- בעיה זו תופסת תאוצה ותוספת נפח רחב בעולמות אבטחת המידע מכיוון שרוב חברות הטכנולוגיה מביאות פתרונות וישומים טכנולוגים למשתמשי קצה באמצעות אפליקציות ייעודיות למובייל ואתרים רספונסיבים ל-WEB.
- המאגר מכיל בסה"כ 4591 אפליקציות, מתוכם 1652 מסווגות כידידותיות 2939 זדוניות.
- לכל אפליקציה מכילה 22,383 רשומות של פיצ'רים בינאריים כאשר 1 מייצג שהאפליקציה מכילה את הפיצ'ר הנתון, אחרת 0.
- במהלך בניית הפרויקט התמודדנו עם שלל בעיות בכדי לדעת את האפליקציות כזדוניות או ידידותיות שיפורטו בהמשך.

תיאור המאגר

- הרשומות של הפיצ'רים מחולקות ל10 קטגוריות שונות. כל קטגוריה מכילה פיצ'רים בינאריים שמתארים תכונות שקיימות באפליקציה. הקטגוריות הן:
- **Sha256-** פונקציית גיבוב קריפטוגרפית המג'נרטת מזהה ייחודי עבור כל אפליקציה שאנחנו מחלצים ממנה מידע של פיצ'רים. המודל לא מקבל את השדה הנ"ל לצורך הלמידה.
- **Label- השדה המייצג לנו האם האפליקציה זדונית או לא.**
- **app_permissions** - ההרשאות שאפליקציה מבקשת מהמשתמש כשהיא מותקנת במכשיר שלו. דוגמאות להרשאות אפליקציה כוללות גישה למצלמה, למיקרופון או לרשימת אנשי הקשר.
- **api_permissions** - אלו הן הרשאות שאפליקציה מבקשת לגשת לממשקי API שונים (ממשקי תכנות יישומים) המסופקים על ידי מערכת ההפעלה אנדרואיד. דוגמאות להרשאות API כוללות את היכולת לגשת למיקום המכשיר, לבצע שיחות טלפון או לשלוח הודעות טקסט.
- **api_calls** - אלו הן פונקציות ה- API הספציפיות שאליהן אפליקציה קוראת בזמן ריצה.

תיאור המאגר

- **Activities-** אלו הם המסכים או רכיבי ממשק המשתמש המרכיבים אפליקציה עבור משתמשי הקצה. לדוג':
- **s_and_r** - מייצג לנו את הקלטים והשירותים- שירותים הם תהליכי רקע שהאפליקציה יכולה להפעיל , וקלטים הם רכיבים שיכולים לקלוט שידורים מהמערכת או מאפליקציות אחרות. דוגמא לשירות: קביעת אירוע ביומן לפי שעה דרך הווצאפ, דוגמא לקלט: שיתוף קובץ pdf בין שתי אפליקציות שונות.
- **interesting_calls** - הכוונה הם לקריאות API ספציפיות או פונקציות מערכת שנחשבות ראויות לציון או עלולות להיות חשודות מכל סיבה שהיא.
- **Urls** – כתובות האינטרנט שהאפליקציה מתקשרת כדי לקבל/לעדכן/למחוק/ליצור (insert ,update ,get ,delete) לדוג': *endpoint- >users/getusersbyid*
- **providers -** אלו הם המסכים או רכיבי ממשק המשתמש המרכיבים אפליקציה עבור משתמשי הקצה.

- בסה"כ כמות הפיצ'רים שחולצו כלל האפליקציות היא רשימה של 22,383 פיצ'רים ייחודיים.

תהליך השגת המידע וניתוח המאגר

- השלב הראשון היה להשיג מאגר מידע של אפליקציות זדוניות וידידותיות מהאינטרנט. חיפשנו בגוגל, מאגרי מידע ופרויקטים קיימים באינטרנט ולבסוף עשינו איחוד של כלל המידע שמצאנו ל-2 תיקיות, תיקייה אחת עבור אפליקציות זדוניות ותיקייה שניה עבור אפליקציות שפיריות.
- כתבנו סקריפט שרץ על 2 התיקיות לעיל ומשנה את שמות הקבצים כך שהשם של הקובץ של אפליקציה זדונית יכיל "_M" ועבור אפליקציה שפירית יתווסף "_B" כך שבסופו של דבר יהיה ייצוג לכל אפליקציה האם היא זדונית או לא.
- בשלב השלישי העברנו את כל האפליקציות הקיימות לתיקייה אחת שתכיל את כלל האפליקציות לאחר שינויי השמות.
- רצנו על כל הקבצים בתיקייה:
  עבור כל קובץ נבדוק האם השם שלו מכיל ייצוג עבור זדוני או לא, ע"פ הבדיקה הזו נדע לתת label לכל אפליקציה ואת השדה הזה העברנו לפונקציה שמבצעת חילוץ של הפיצ'רים עבור אותה אפליקציה ושומרת את המידע בקובץ json.
- הפונקציה שמחלצת את המידע מבצעת unzip עבור כל קובץ APK ושומרת אותו בתיקייה שנקראת unpack .
- בכדי לחלץ את פיצ'רים השתמשנו בכלי שנקרא AAPT. הכלי הזה משתנה בין מערכות ההפעלה שמריצות את הקוד(קובץ אחר כל מערכת הפעלה) והוא חלק בלתי נפרד מהחבילה של הandroid sdk (ערכת פיתוח תוכנה). השתמשנו בכלי הזה לחילוץ משאבים מקובץ APK .
- עבור כל קטגוריה יצרנו list שמכיל את כלל הפיצ'רים שהיו באפליקציה לפי קטגוריות.
- ולבסוף יצרנו קובץ json אחד המכיל מערך ריק ואליו הכנסנו אובייקטים כאשר כל אובייקט מייצג את הפיצ'רים של האפליקציות.

סיום העיבוד והכנתו ללמידה

- בשלב הראשון ייבאנו את כל המידע מקובץ הjson וטענו אותו לאובייקט, הכנסו את המידע לDataFrame והשלמנו את הפיצ'רים החסרים בכל אובייקט(לפי ערך 0- אם הפיצ'ר היה קיים היה לו ערך 1) כדי שהפיצ'רים יהיו בפורמט אחיד כלפי כלל האפליקציות.
- בדקנו את האלגוריתמים לפי שלש חלוקות שונות של train וtest 90-10, 80-20, 70-30
- הרצנו את האלגוריתמים על מגוון ערכים שונה של הפרמטרים של כל אלגוריתם.
- לאחר הרצת כלל האלגוריתמים על כלל החלוקות והפרמטרים יצאנו את כלל התוצאות לקבציjson חיצוניים כאשר כל שם קובץ מייצג את האלגוריתם ואת החלוקה שבה הרצנו את האלגוריתם. כלומר, כל קובץ יכיל את ההרצה של האלגוריתם על כלל הערכים הרנדומליים של הפרמטרים לפי ייחס האימון והטסט.

אלגוריתמים

Logistic Regression

- מכיוון שהמידע שלנו מורכב מפיצ׳רים במימד גבוה מה שיכול לגרום ללמידה להיות מסובכת, הרצנו את האלגוריתם עם Ridge Regularization שמנסה לאזן בין המזעור של ה-loss של הסיכון האמפירי וגם של הווקטור. בנוסף קבענו learning rate =0.1 ומגוון ערכי c_value שונים שקטנים מ1 בשביל להגדיל את ההשפעה של Regularizer. פרמטר c_value מקטין את ההשפעה של Regularizer ביחס הפוך לערך שלו.
- הפרמטרים שהביאו את התוצאות המקסימליות:
- C_value: 0.7462138628,
- epsilon: 0.001
- random_state_val: 1

Logistic Regression

כל תמונה מייצגת תוצאות הרצה על train ו- test לפי learning rate שונה. אפשר לראות שקיבלנו תוצאות יפות עבור אלגוריתם זה, וניתן לראות שלא קיים overfitting מובהק.

0.1


```
"LogisticRegression": {
  "modelLogisticRegression": [
    {
      "modelLogisticRegressionTrainAndTest": {
        "accuracy": 0.9717391304347827,
        "precision": 0.9590443686006825,
        "recall": 0.9964539007092199,
        "C": 0.7462138628,
        "epsilon": 0.001,
        "random_state_val": 1
      },
      "modelLogisticRegressionTrain": {
        "accuracy": 0.999515855725006,
        "precision": 1.0,
        "recall": 0.9991912656692277,
        "C": 0.7462138628,
        "epsilon": 0.001,
        "random_state_val": 1
      }
    }
  ]
},
```


0.2


```
"LogisticRegression": {
  "modelLogisticRegression": [
    {
      "modelLogisticRegressionTrainAndTest": {
        "accuracy": 0.9619151251360174,
        "precision": 0.9456521739130435,
        "recall": 0.9905123339658444,
        "C": 0.7462138628,
        "epsilon": 0.001,
        "random_state_val": 1
      },
      "modelLogisticRegressionTrain": {
        "accuracy": 0.9994553376906318,
        "precision": 1.0,
        "recall": 0.9991023339317774,
        "C": 0.7462138628,
        "epsilon": 0.001,
        "random_state_val": 1
      }
    }
  ]
},
```


0.3


```
"LogisticRegression": {
  "modelLogisticRegression": [
    {
      "modelLogisticRegressionTrainAndTest": {
        "accuracy": 0.9637155297532656,
        "precision": 0.9551357733175915,
        "recall": 0.9853836784409257,
        "C": 0.7462138628,
        "epsilon": 0.001,
        "random_state_val": 1
      },
      "modelLogisticRegressionTrain": {
        "accuracy": 0.9996887643946467,
        "precision": 1.0,
        "recall": 0.999482936918304,
        "C": 0.7462138628,
        "epsilon": 0.001,
        "random_state_val": 1
      }
    }
  ]
},
```

K-Nearest Neighbors

אפשר לראות שהאלגוריתם השיג תוצאות פחות טובות מה logistic regression . בדקנו על מגוון ערכים של הפרמטר k וראינו שאלו התוצאות הטובות ביותר לפי פרמטר k. ניתן לראות בנוסף שקיבלנו precision גבוה לעומת recall נמוך מה שמעיד שהאלגוריתם לא טועה כאשר הוא מסווג אפליקציה כ malicious אבל מפספס כמות נכבדה של אפליקציות malicious ומתייג אותם כbenign

0.1


```
"0.1": [
  {
    "KNeighborsClassifier": {
      "modelKNeighborsClassifier": [
        {
          "modelKNeighborsClassifierTrainAndTest": {
            "accuracy": 0.8673913043478261,
            "precision": 0.9637096774193549,
            "recall": 0.8213058419243986,
            "n_neighbors": 5
          },
          "modelKNeighborsClassifierTrain": {
            "accuracy": 0.9293149358508835,
            "precision": 0.980106100795756,
            "recall": 0.8997564935064936,
            "n_neighbors": 5
          }
        }
      ]
    }
  }
]
```


0.2


```
"0.2": [
  {
    "KNeighborsClassifier": {
      "modelKNeighborsClassifier": [
        {
          "modelKNeighborsClassifierTrainAndTest": {
            "accuracy": 0.9009793253536452,
            "precision": 0.9767932489451476,
            "recall": 0.852670349907919,
            "n_neighbors": 5
          },
          "modelKNeighborsClassifierTrain": {
            "accuracy": 0.9251089324618736,
            "precision": 0.9782716049382716,
            "recall": 0.8955696202531646,
            "n_neighbors": 5
          }
        }
      ]
    }
  }
]
```


0.3


```
"0.3": [
  {
    "KNeighborsClassifier": {
      "modelKNeighborsClassifier": [
        {
          "modelKNeighborsClassifierTrainAndTest": {
            "accuracy": 0.8933236574746009,
            "precision": 0.9721059972105998,
            "recall": 0.845873786407767,
            "n_neighbors": 5
          },
          "modelKNeighborsClassifierTrain": {
            "accuracy": 0.91005291005291,
            "precision": 0.9696796338672768,
            "recall": 0.877783531848783,
            "n_neighbors": 5
          }
        }
      ]
    }
  }
]
```

Decision tree

השתמשנו במדד ג'יני למציאת פיצול אופטימלי בצמתי העץ במקום במדד האנטרופיה שנלמד בכיתה. כמו כן עומק העץ המקסימלי שנותן את התוצאות הטובות ביותר. ניתן לראות שהאלגוריתם נותן תוצאות דומות כמו logistic regression מבחינת המדדים הסטטיסטים.


```
"0.1": [
  {
    "DecisionTreeClassifier": {
      "modelDecisionTreeClassifier": [
        {
          "modelDecisionTreeClassifierTrainAndTest": {
            "accuracy": 0.9652173913043478,
            "precision": 0.9795221843003413,
            "recall": 0.9663299663299664,
            "max_depth": 50,
            "random_state_val": 0
          },
          "modelDecisionTreeClassifierTrain": {
            "accuracy": 0.9992737835875091,
            "precision": 0.9991866612444082,
            "recall": 0.999593165174939,
            "max_depth": 50,
            "random_state_val": 0
          }
        }
      ]
    }
  }
]
```

```
"0.2": [
  {
    "DecisionTreeClassifier": {
      "modelDecisionTreeClassifier": [
        {
          "modelDecisionTreeClassifierTrainAndTest": {
            "accuracy": 0.9782372143634385,
            "precision": 0.9858407079646018,
            "recall": 0.9789103690685413,
            "max_depth": 50,
            "random_state_val": 0
          },
          "modelDecisionTreeClassifierTrain": {
            "accuracy": 0.9997276688453159,
            "precision": 0.9995427526291724,
            "recall": 1.0,
            "max_depth": 50,
            "random_state_val": 0
          }
        }
      ]
    }
  }
]
```

```
"0.3": [
  {
    "DecisionTreeClassifier": {
      "modelDecisionTreeClassifier": [
        {
          "modelDecisionTreeClassifierTrainAndTest": {
            "accuracy": 0.9666182873730044,
            "precision": 0.9665127020785219,
            "recall": 0.9800936768149883,
            "max_depth": 50,
            "random_state_val": 0
          },
          "modelDecisionTreeClassifierTrain": {
            "accuracy": 0.9996887643946467,
            "precision": 1.0,
            "recall": 0.9994739610731194,
            "max_depth": 50,
            "random_state_val": 0
          }
        }
      ]
    }
  }
]
```

Linear SVC

- למדנו שהאלגוריתם הזה עובד היטב עם נתונים לא מבניים כמו טקסט, תמונות וכו
- לאחר חיפוש באינטרנט החלטנו להשתמש ב Linear SVC כאשר הkernel שלו ליניארי מכיוון שהוא מתאים לכמות גדולה של מימדים.
- קראנו בנוסף שהסיכוי לoverfitting קטן ביחס לLR. אבל לפי התוצאות שלנו הדבר לא משתקף.


```
"0.2": [
  {
    "LinearSVC": {
      "modelLinearSVC": [
        {
          "modelLinearSVCTrainAndTest": {
            "accuracy": 0.9357997823721437,
            "precision": 0.9543726235741445,
            "recall": 0.9348230912476723,
            "C": 0.3837138946,
            "epsilon": 0.001,
            "random_state_val": 0
          },
          "modelLinearSVCTrain": {
            "accuracy": 0.9997276688453159,
            "precision": 1.0,
            "recall": 0.9995491433724075,
            "C": 0.3837138946,
            "epsilon": 0.001,
            "random_state_val": 0
          }
        }
      ]
    }
  }
]
```

```
"0.1": [
  {
    "LinearSVC": {
      "modelLinearSVC": [
        {
          "modelLinearSVCTrainAndTest": {
            "accuracy": 0.9282608695652174,
            "precision": 0.937007874015748,
            "recall": 0.9333333333333333,
            "C": 0.5869497833,
            "epsilon": 0.001,
            "random_state_val": 0
          },
          "modelLinearSVCTrain": {
            "accuracy": 0.999515855725006,
            "precision": 1.0,
            "recall": 0.9992,
            "C": 0.5869497833,
            "epsilon": 0.001,
            "random_state_val": 0
          }
        }
      ]
    }
  }
]
```

```
"0.3": [
  {
    "LinearSVC": {
      "modelLinearSVC": [
        {
          "modelLinearSVCTrainAndTest": {
            "accuracy": 0.9223512336719883,
            "precision": 0.9617755856966708,
            "recall": 0.9112149532710281,
            "C": 0.7316143185,
            "epsilon": 0.001,
            "random_state_val": 0
          },
          "modelLinearSVCTrain": {
            "accuracy": 0.9996887643946467,
            "precision": 1.0,
            "recall": 0.9994734070563455,
            "C": 0.7316143185,
            "epsilon": 0.001,
            "random_state_val": 0
          }
        }
      ]
    }
  }
]
```

אתגרים

- כפי שניתן לראות, הפרויקט דרש מחקר לקבלת הdata (קבצי APK) והצריך בניה של מגוון מודלים שונים להשלמת רכיבי הפרויקט. האתגרים שעמדו בפנינו הם:

1. אפיון מלא של הפרויקט, בניית סדר תהליכים, למידת קונפיגורציות מתאימות לצורך ביצוע הפרויקט וחיבור של המודלים השונים.
2. למידת ספריות למידת מכונה של Python , נעזרנו באינטרנט ובמדריכים שונים המסבירים כיצד להשתמש בספריית Scikit-learn
3. המידע שלקחנו לא הגיע מאתר ייעודי לdata כגון: Kaggle ולא הגיע בפורמט טבלאי שהיה נוח לעבודה עם אלגוריתמי הלמידה.
4. בנוסף, הdata שנאספה לא הגיעה עם תיוג ולכן נאלצנו להוסיף אותו בכוחות עצמנו למודל חילוץ הפיצ'רים חילוץ הפיצ'רים מהאפליקציות – הפיצ'רים לא היו בפורמט של KEY ו-VALUE והיינו צריכים לחלץ את המידע מקבצים דחוסים(APK)

אתגרים

1. אובייקט הפיצ'רים עבור כל אפליקציה היה צריך להיות בפורמט אחיד עבור כלל האפליקציות מבחינת השדות שהיו בכל אובייקט(מילוי שדות חסרים בערך 0) .

2. היו 22,383 פיצ'רים ולכן הזמן ריצה של האלגוריתמים הצריך מכונה חזקה והמון זמן עיבוד של המידע
בעקבות גודל כמות הפיצ'רים השונים התחלנו לעשות מחקר מעמיק על הפיצ'רים, דבר זה גרע מאיתנו לסיים את המחקר לחלוטין ולסנן פיצ'רים שאינן רלוונטיים עבור המודלים.

משימות עתידיות

- סינון המידע וצמצום כמות הפיצ'רים כך שהפיצ'רים שיופיעו יהיו פיצ'רים שייצגו בצורה הטובה ביותר את התכונות של כל אפליקציה ויהיו רלוונטיות לאלגוריתמי הלמידה. דבר זה יוביל לצמצום את זמני הריצה ואיכות הלמידה של האלגוריתמים.
- הוספת אלגוריתמי למידה נוספים, כדי לבצע למידה מעמיקה על המידע וביצוע השוואות בין האלגוריתמים.
- התחלנו לבצע ניתוח סטטיסטי לטובת הבנה של הפיצ'רים, לכל פיצ'ר השוונו את האחוזים שהפיצ'ר קיים ב-data לפי סוג התיוג. התוצאות שראינו העידו על כך שקיימים פיצ'רים שפחות רלוונטיים לתיוג האפליקציה. כמובן שניתן לבצע מחקר מעמיק יותר.

קישורים

- קישור לפרויקט: https://github.com/motidahari/Mobile-Security-ML-Android-Malware-Detection-Machine-learning-course
- קישור לקובץ הפיצ'רים של המודלים: קישור
- קישור לתוצאות הסיווג של האלגוריתמים: קישור
- קישור לרשימה של קטגוריות ופיצ'רים עבור כל קטגוריה: קישור
- קישור לניתוח סטטיסטי של הפיצ'רים: קישור