ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

каф. ЕОМ

Методи машинного навчання на основі аналізу трафіка для виявлення програм-вимагачів платформи Android

Ст. групи КСМм-23-1
Ступаренко Р.Ю.

Керевник
ас. Кравченко П.О.

Актуальність кваліфікаційної роботи

2

- У сучасну цифрову епоху використання пристроїв Android широко поширене в різних секторах. Кіберзлочинці неминуче пристосовуються до нових технологій безпеки та використовують ці платформи для використання вразливостей у підлих цілях, таких як викрадення конфіденційних і особистих даних користувачів. Це може призвести до фінансових втрат, дискредитації, програм-вимагачів або розповсюдження інфекційного шкідливого програмного забезпечення та інших катастрофічних кібератак. Через те, що програми-вимагачі шифрують дані користувача та вимагають викуп в обмін на ключ дешифрування, це один із найруйнівніших типів шкідливого програмного забезпечення. Наслідки атак програм-вимагачів можуть варіюватися від втрати важливих даних до зриву бізнес-операцій і значної грошової шкоди. Методи на основі штучного інтелекту (ШІ), а саме машинне навчання (ML), довели свою ефективність у виявленні атак програм-вимагачів Android. Проте ансамблевы моделі та моделі глибокого навчання (DL) не були достатньо вивчені.
- Тому в цій кваліфікаційній роботі будуть використовуються методи на основі ML і DL для створення ефективних, точних і надійних моделей для двійкової класифікації. Для навчання та тестування моделей використовувався загальнодоступний набір даних від Kaggle, що складається з 392035 записів із безпечним трафіком і 10 різними типами атак програм-вимагачів Android.

3

Мета та завдання роботи

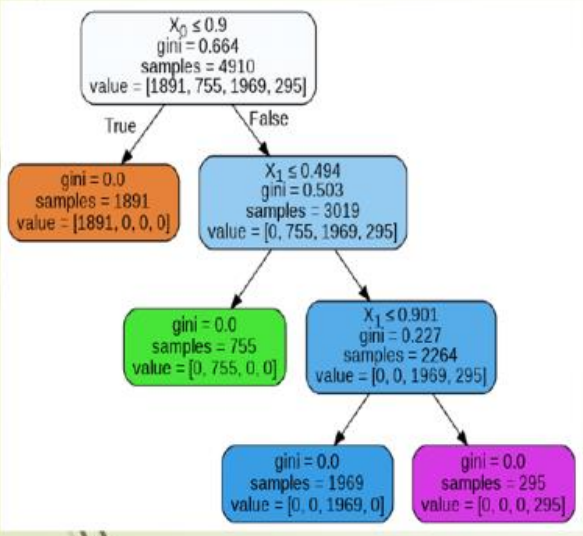
- Мета кваліфікаційної роботи є використання методів машиного навчання на базі аналізу трафіка для виявлення програм-вимагачів на платформі Android.

Завдання:

- Провести аналіз сучасних методів машинного навчання
- Побудувати методологію дослідження
- Провести аналіз та усунути дисбаланс класів у наборі даних
- Провести експериментальні дослідження (програмна реалізація)

4

Метод Дерева рішень (DT)



Дерева рішень є потужним інструментом в галузі машинного навчання та аналізу даних. Основна ідея цього методу полягає в створенні структурованого дерева з рішеннями на кожному рівні, яке допомагає приймати рішення на основі вхідних даних.

Алгоритм дерева рішень розпочинається з кореня дерева і розглядає набір вхідних даних. На кожному кроці він обирає оптимальний поділ для розділення даних на підгрупи. Цей вибір зазвичай базується на показниках, таких як інформаційна ентропія чи критерій Джині, які вимірюють ступінь "чистоти" чи однорідності підгруп.

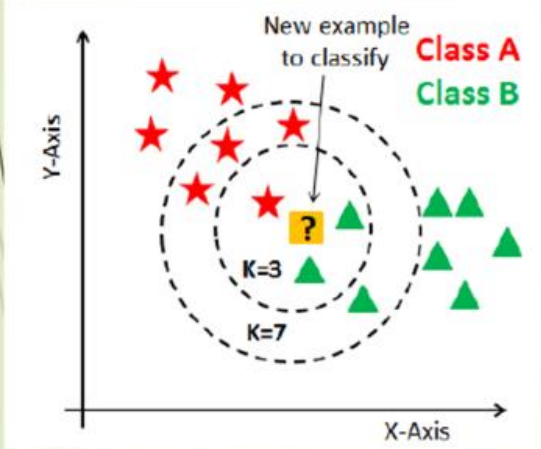
Формула інформаційної ентропії:

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right)$$

де n – число класів у вихідній підмножині, Ni – кількість прикладів i-го класу, N – загальна кількість прикладів у підмножині.

Метод k-NN

5

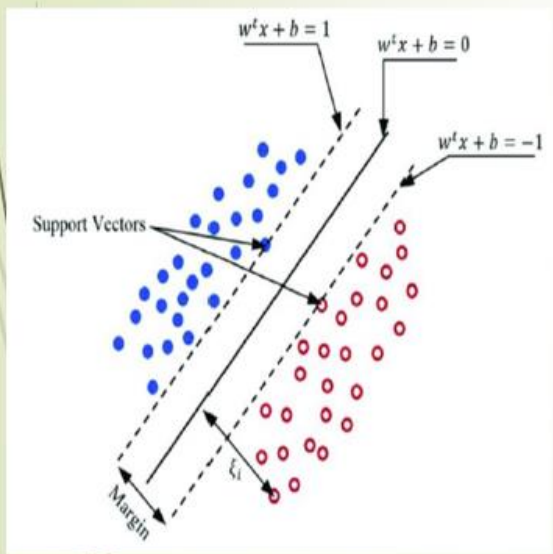


Метод k-найближчих сусідів (k-NN) є одним із найпростіших алгоритмів машинного навчання, який використовується для класифікації та регресії. Ця методика грунтується на припущенні, що схожі дані мають схожі класи чи значення. Основна ідея полягає в тому, що об'єкт класифікується за більшістю його "сусідів" у просторі ознак.

Для виконання класифікації за методом k-NN потрібно визначити кількість сусідів (k) та відстань між ними. Після цього, для нового об'єкта обчислюється відстань до усіх інших об'єктів у наборі даних. Потім вибираються k найближчих об'єктів, і класифікація відбувається шляхом голосування: новий об'єкт отримує клас, який найчастіше зустрічається серед його сусідів.

Метод опорних векторів (SVM)

6



Метод опорних векторів (SVM) є потужним і широко використовуваним алгоритмом у машинному навчанні, особливо в задачах класифікації та регресії.

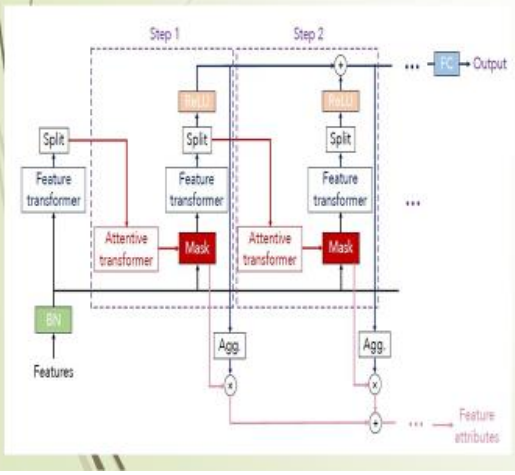
Основна ідея полягає у тому, щоб знайти оптимальну гіперплощину (або набір гіперплощин), яка максимізує розбіжність між двома класами даних. Гіперплощина - це просто (N-1)-мірна площина в N-вимірному просторі, яка розділяє дані на два класи.

Для досягнення цього SVM використовує вектори навчання, які є точками даних в просторі. Вектори навчання, розташовані найближче до границі розділення, називаються опорними векторами. Опорні вектори є ключовими елементами для побудови гіперплощини, оскільки вони визначають її положення та орієнтацію.

Штучна нейронна мережа TabNet

7

У роботі також розглядається архітектура НМ для табличних даних (TabNet), призначена для відображення «дерева рішень». Мета - успадкувати переваги ієрархічних методів (інтерпретованість, розріджений вибір ознак) та методів на основі НМ (покрокове та наскрізне навчання). Зокрема, у TabNet розглядаються дві ключові потреби - висока продуктивність та інтерпретованість. Високу продуктивність часто недостатньо - НМ повинні інтерпретувати, замінювати деревоподібні методи.



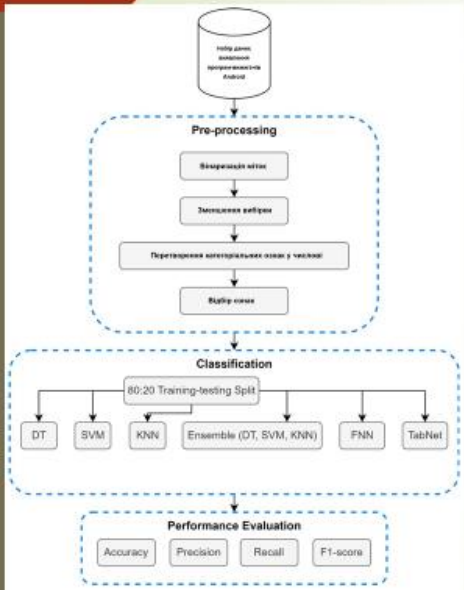
TabNet - нейромережа з повнозв'язних шарів з послідовним механізмом уваги, яка:

- використовує розріджений вибір об'єктів за примірниками, отриманий на основі навчального набору даних;
- створює послідовну багатоступінчасту архітектуру, в якій кожен крок прийняття рішення може зробити свій внесок у ту частину рішення, яка базується на вибраних функціях;
- покращує здатність до навчання шляхом нелінійних перетворень вибраних функцій;
- імітує ансамбль, залучаючи більш точні виміри та більше кроків покращення рішення.

Кожен шар даної архітектури - це крок рішення, що містить у собі блок з повнозв'язними шарами для перетворення характеристик - Feature Transformer і механізм уваги визначення важливості вхідних оригінальних характеристик.

Методологія дослідження

8



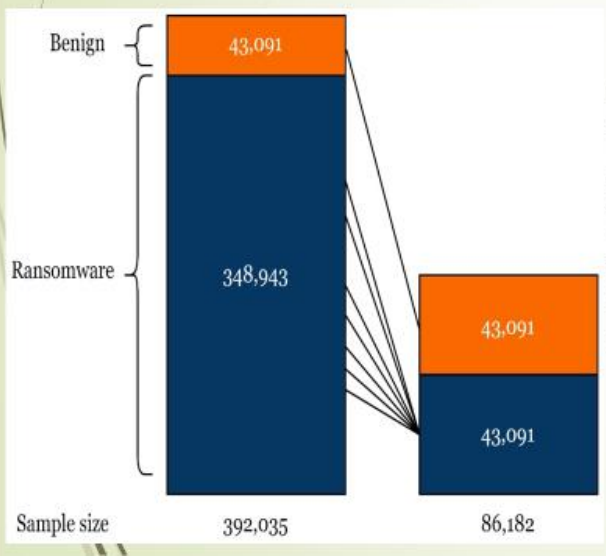
Це дослідження було проведено з використанням узгодженої методології, яка включала наступні основні етапи: збір даних, попередню обробку, класифікацію та оцінку. Коли ми отримали набір даних, ми спочатку візуалізували та оцінили дані, щоб зрозуміти наявні функції. Після цього етапи попередньої обробки для покращення якості даних і підвищення продуктивності та надійності моделей були зрозумілі.

Етапи попередньої обробки включали бінарізацію міток, зменшення вибірки, перетворення категоріальних ознак у числові та вибір ознак за допомогою методів прямого вибору ознак і важливості ознак. Потім набір даних було розділено на набори для навчання та тестування 80:20 відповідно, щоб підготувати його до класифікації за допомогою моделей DT, SVM, KNN, ансамблю (DT, SVM, KNN), FNN і TabNet. Нарешті результати були отримані та проаналізовані.

На рисунку представлено візуальне представлення методологічних кроків, прийнятих для нашого дослідження.

Зменшення вибірки

9



Після бінарізаціх міток, щоб усунути дисбаланс класів у наборі даних, ми застосували метод рандомізованої недостатньої вибірки.

Цей метод зазвичай використовується для вирівнювання міток класів у наборі даних шляхом зменшення кількості значно більш поширеного класу (програм-вимагачів) у нашому випадку до рівня меншості класу (доброякісного). Таким чином, вибірка обох міток класу була недостатньою, щоб мати однаковий розмір (43 091: програми-вимагачі; 43 091: доброякісні), як показано на рисунку

Експериментальні дослідження

10

- Вибір обладнання: Використовувався ноутбук на операційній системі Windows 11 Home з 8 ГБ оперативної пам'яті та процесором i7.
- В якості мови програмування було обрано Python 3.10. Також використовувалося Tensorflow
- Набір даних Android Ransomware Detection, загальнодоступний на Kaggle, використовувався після недостатньої вибірки. Набір даних складався з 43091 програми-вимагача та 43091 доброякісного зразка.
- Після застосування методів вибору ознак (прямий вибір ознак і важливість ознак з k = 10) було відібрано 19 найкращих спільних ознак для навчання та тестування моделей.
- Для навчання та тестування було виконано розподіл даних 80:20. Загалом було проведено два експерименти з використанням DT, SVM, KNN, моделі ансамблю (DT, SVM, KNN) із random_state = 42, FNN і TabNet.
- У експерименті №1 були використані всі 70 функцій.
- У експерименті № 2 було використано 19 найкращих характеристик. Модель DT оптимізували за допомогою налаштування гіперпараметрів, а потім застосували пошук у сітці з перехресною перевіркою.

11

Налаштування параметрів, що застосовуються до моделей

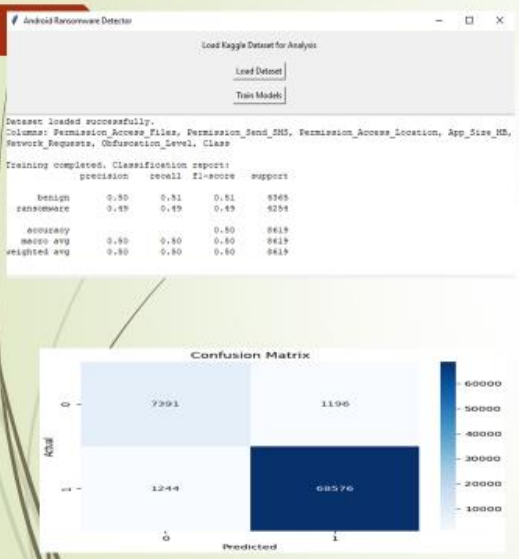
Модель	Параметри	Вибір оптимальних значень
DT	'max_depth'	[None, 5, 10, 15]
	'min_samples_split'	[2, 5, 10]
	'min_samples_leaf'	[1, 2, 4]
SVM	C	[0.1, 1, 10]
	Kernel	['linear', 'rbf', 'poly']
	gamma	['scale', 'auto']
FNN	Функція активації в прихованих шарах	ReLU with 32 units
	Кількість нейронів у вихідному шарі	1
	Функція активації на вихідному рівні	Sigmoid
	Розмір партії	16
	Кількість епох	10
	Кількість шарів	3
TabNet	Кількість кроків прийняття рішення в мережі	64
	Розмір залучення уваги	32
	Кількість кроків у блоках уваги та агрегації	5
	Гамма	1.3
	Кількість самостійно навчених трансформаторів характеристик	2
	Кількість трансформаторів загальної функції	2
	Імпульс для нормалізації партії	0.02
	Максимальне абсолютне значення	2.0

12

Результати, отримані після експерименту 1 з використанням усіх 70 функцій і експерименту 2 з використанням 19 найкращих функцій.

Методи	Функції	Accuracy	Precision	Recall	F1-score
DT	Всі	96.89%	98.29%	98.22%	98.25%
	19 функцій	97.24%	98.50%	98.40%	98.45%
SVM	Всі	89,05%	89,05%	100%	94,21%
	19 функцій	89,05%	89,05%	100%	94,21%
KNN	Всі	88.79%	90.49%	97.68%	93.95%
	19 функцій	88.43%	90.10%	97.74%	93.77%
Ensemble (DT, SVM, KNN)	Всі	90.44%	90.37%	99.91%	94.90%
	19 функцій	90.24%	90.17%	99.93%	94.80%
FNN	Всі	89.09%	89.12%	99.95%	94.22%
	19 функцій	89.10%	89.13%	99.95%	94.23%
TabNet	Всі	89.04%	89.05%	99.99%	94.20%
	19 функцій	86.84%	88.96%	97.28%	92.94%

Реалізація дослідження



Експеримент 1 – матриця плутанини DT.

Було проаналізовано матриці плутанини для DT, показані на рисунках. Правильна ідентифікація безпечного мережевого трафіку Android вважається TN; однак правильна ідентифікація трафіку програм-вимагачів Android є TP.

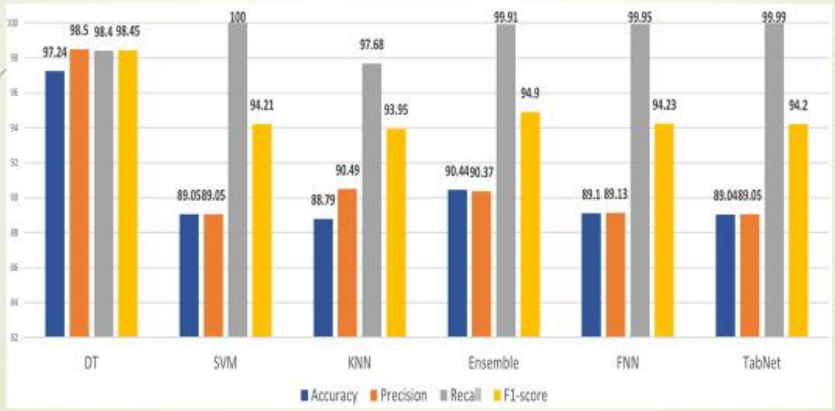
Неправильна ідентифікація безпечного мережевого трафіку як програми-вимагача є FN, а неправильна ідентифікація програми-вимагача як безпечного мережевого трафіку є FP. Якщо модель має більше TP і TN (або менше FN і FP), вона вважається більш точною. На рисунках для результатів класифікатора DT ми бачимо, що кількість екземплярів TP та TN більше. Він досяг найвищої точності, точності та оцінки F1.

Експеримент 2 – матриця плутанини DT.

14

Порівняння результатів

- У двох проведених експериментах було зареєстровано чотири метрики оцінювання — Accuracy, Precision, Recall, та F1-score— і всі моделі оцінювали за ними.



Висновки

15

В рамках кваліфікаційної роботи було проведено дослідження методів штучного інтелекту (машинного навчання) для виявлення програм вимагачів на платформі Android на базі аналізу трафіка.

- Для проведення двох експериментів було використано останній набір даних виявлення програм-вимагачів Android, 2023 від Kaggle. У рамках попередньої обробки даних для усунення дисбалансу набору даних була прийнята техніка рандомізованої недостатньої вибірки. Після попередньої обробки даних було застосовано вибір ознак за допомогою прямого вибору ознак і важливості ознак. Всього 19 функцій були визнані вирішальними для аналізу та ідентифікації атак. Після вибору функції для навчання та тестування було виконано розділення набору даних 80:20.
- Було проведено два експерименти з використанням DT, SVM, KNN, моделі ансамблю (DT, SVM, KNN), FNN і TabNet. У експерименті 1 були використані всі 70 функцій. У експерименті 2 було використано 19 найкращих характеристик. Ефективність цих моделей була розрахована з точки зору точності, точності, запам'ятовування та оцінки F1. У результаті DT перевершив результати з точністю 97,24%, точністю 98,50% і результатом F1 98,45%. Найвище відкликання 100% було отримано за допомогою моделі SVM.
- Апробація результатів дослідження: Ляшенко О.С., Ступаренко Р.Ю., Знайдюк В.Г. Методи машинного навчання для аналізу трафіку для виявлення програм-вимагачів платформи Android. Проблеми інформатизації. Тези доповідей дванадцятої міжнародної НТК. 2024. – 21-22 листопада 2024. – Том 2. – С. 95.

ДОДАТОК Б

Android ransomware detector

```
import tkinter as tk
from tkinter import filedialog, messagebox
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report

class RansomwareDetectorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Android Ransomware Detector")

        # GUI elements
        self.label = tk.Label(root, text="Load Kaggle Dataset for Analysis")
        self.label.pack(pady=10)

        self.load_button = tk.Button(root, text="Load Dataset", command=self.load_dataset)
        self.load_button.pack(pady=5)

        self.train_button = tk.Button(root, text="Train Models", command=self.train_models, state=tk.DISABLED)
        self.train_button.pack(pady=5)
```

```
        self.result_text = tk.Text(root, height=15, width=80)
        self.result_text.pack(pady=10)

        self.dataset = None

    def load_dataset(self):
        file_path = filedialog.askopenfilename(title="Select Dataset", filetypes=[("CSV files", "*.csv")])
        if file_path:
            try:
                self.dataset = pd.read_csv(file_path)
                self.result_text.insert(tk.END, "Dataset loaded successfully.\n")
                self.result_text.insert(tk.END, f"Columns: {', '.join(self.dataset.columns)}\n")

                self.train_button.config(state=tk.NORMAL)
            except Exception as e:
                messagebox.showerror("Error", f"Failed to load dataset: {e}")

    def train_models(self):
        if self.dataset is None:
            messagebox.showwarning("Warning", "Please load a dataset first.")
            return
        try:
            # Preprocess dataset
            X = self.dataset.iloc[:, :-1]
            y = self.dataset.iloc[:, -1]
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
                # Initialize models
                dt = RandomForestClassifier(random_state=42)
                svc = SVC(probability=True, random_state=42)
                knn = KNeighborsClassifier()

                # Ensemble model
                ensemble = VotingClassifier(estimators=[
                    ('Random Forest', dt),
                    ('SVM', svc),
                    ('KNN', knn)
                ], voting='soft')

                # Train ensemble model
                ensemble.fit(X_train, y_train)
                predictions = ensemble.predict(X_test)

                # Show results
                report = classification_report(y_test, predictions)
                self.result_text.insert(tk.END, "\nTraining completed. Classification report:\n")
                self.result_text.insert(tk.END, report)

            except Exception as e:
                messagebox.showerror("Error", f"An error occurred during training: {e}")

        if __name__ == "__main__":
            root = tk.Tk()
            app = RansomwareDetectorApp(root)
            root.mainloop()
```