Analýza ML metód a algoritmov pre detekciu malware (typ malware - trojské kone - detegovanie zo záznamov prevádzky o tokoch v počítačovej sieti)

Projekt INNOvative SOC/INNOSOC

Obsah

Úvod ... 3
Experimenty a výsledky ... 8
Datasety ... 8
Dataset: Trojan_Detection ... 8
Použité algoritmy a metódy ... 10
Náhodný les (Random Forest) ... 10
AdaBoost a Gradient Boosting Classifier ... 12
Gaussian Naive Bayes ... 13
Deep Learning with Feedforward Neural Network ... 14
Príprava a postup výpočtov ... 15
Nastavenia a parametre zvolených metód ... 16
Náhodný les (Random Forest) ... 16
AdaBoost ... 17
Gaussian Naive Bayes ... 18
Deep Learning with Feedforward Neural Network ... 18
Vyhodnocovanie experimentov ... 20
Konfúzna matica ... 20
Accuracy ... 21
Precision ... 21
Recall ... 22
F1-Score ... 22
Výsledky ... 22
Dataset: TrojanDetec (výpočty na Google Colab - Processing time) ... 22
2025 12 05: výpočty - obrazovky ... 23
Zhrnutie ... 34

Úvod

Malvér

Názov malvér (malicious software, škodlivý softvér) sa používa pre označenie akéhokoľvek programu alebo kódu, ktorý bol vytvorený so cieľom poškodiť počítačový systém, napríklad bez súhlasu používateľa získať prístup k údajom, k počítačovým systémom alebo sieťam, tirž získať pre tvorcu softvéru finančný prospech, narušiť prevádzku, urobiť sabotáž, a podobne.

Malvér môžeme klasifikovať na základe jeho

- správania sa
- spôsobu šírenia
- základe cieľov, ktoré sleduje

Existuje veľa druhov malvéru, k hlavným typom patria nasledujúce.

Vírusy

Sú to malé programy, ktoré sa pripájajú k iným legitímnym súborom alebo programom. Na to aby sa mohli aktivovať a šíriť ďalej vyžadujú interakciu používateľa, napr. otvorenie infikovaného súboru (faktúry, obrázka, ...) v príloha e-mailom, alebo kliknutie na odkaz v e-mailovej správe, otvorením multimediálnych súborov a programov stiahnutých z nedôveryhodných webových stránok, pripojením infikovaného USB kľúča, kliknutím na škodlivú online reklamu, ktorá sa môže objaviť aj na legitímnych webových stránkach, kliknutím na odkaz na sociálnych sieťach, a ďalšie.

Spustením programu sa bez vedomia používateľa spúšťa aj vírus, ktorý môže napadnúť ďalšie programy. Účinky vírusov môžu byť rôzne závažné - od nepríjemných správ až po vážne poškodenie súborov alebo systému. Pre ochranu pred vírusmi je potrebný obozretný postup online aktivitách, aktualizovanie antivírusu, operačného systému a aplikácií.

Počítačové červy (worms)

Sú to samostatné programy. Na rozdiel od vírusov sú navrhnuté tak, aby sa replikovali a šírili autonómne a nevyžadovali priamu interakciu používateľa. Môžu zneužívať bezpečnostné zraniteľnosti softvéru, operačného systému alebo siete (v sieťových protokoloch, aplikáciách a službách, napríklad zdieľanie súborov SMB. Príkladom zneužitia zraniteľnosť v systéme Microsoft Windows bol červ WannaCry. Ďalším príkladom sú aj e-mailové červy, ktoré po infikovaní jedného systému automaticky rozošlú svoje kópie kontaktom v adresári používateľa. Pre šírenie červov ich tvorcovia používajú často sociálne inžinierstvo, pomocou ktorého sa snažia naviesť príjemcu, aby správu otvoril.

Rýchle šírenie červov medzi systémami môže spôsobiť zahltenie sieťovej kapacity, a viesť k vyčerpania systémových zdrojov a spomaleniu alebo zlyhaniu systémov. Červy môžu slúžiť aj ako "nosič" pre iný malvér. Predchádzame im inštalovaním bezpečnostných záplat pre systémy.

Trójske kone (trojans)

Trojské hone predstierajú, že ide o legitímny a užitočný softvér (napr. hru, nástroj, alebo inštaláciu), no v skutočnosti obsahujú skrytý škodlivý kód, ktorý sa dá aktivizovať po splnení určitej podmienky (nastane určitý dátum, napr. Nový rok alebo Vianoce).

Po spustení táto ukrytá časť programu obyčajne vykonáva škodlivé aktivity, napríklad

- otvorenie "zadných vrátok" (backdoor) do systému
- krádež alebo zmenu údajov
- mazanie súborov
- formátovanie disku
- inštalácia ďalšieho malvéru

Ak je bežný program nakazený vírusom, stáva sa jeho rozširovateľom, a je aj potenciálnym trójskym koňom.

Ransomware

Šifruje súbory na napadnutom počítači a robí ich nedostupnými. Páchateľ následne žiada výkupné (ransom) za dešifrovací kľúč. Ide o jednu z najnebezpečnejších a najčastejších hrozieb.

Spyware

Je určený na špehovanie aktivít používateľa bez jeho vedomia. Zbiera citlivé informácie, ako sú prihlasovacie údaje, heslá, históriu prehliadania alebo finančné údaje, a odosielajú ich útočníkovi.

Adware (advertising-supported software, reklamný softvér)

Je to softvér, ktorý automaticky zobrazuje, prehráva alebo sťahuje reklamný materiál, ako sú vyskakovacie okná, bannery či ikony, alebo presmeruje prehliadač. Niekedy slúži na financovanie vývoja programu, aby bol zadarmo dostupný používateľom. Často sa spája so spyware, lebo mnoho takýchto programov sleduje činnosť používateľa bez jeho vedomia a zbiera jeho dáta. Hoci zvyčajne nie je taký deštruktívny ako iné typy malvéru, môže ohroziť súkromie, špehovať používateľa, a spomaliť systém, znepríjemňovať prácu s internetom a softvérom.

Rootkity (program skrývajúci škodlivé aplikácie v PC)

Ide o sadu nástrojov, ktoré útočníkovi umožňujú získať úplnú kontrolu nad systémom a zároveň maskovať svoju prítomnosť a aktivity pred operačným systémom a bezpečnostným softvérom.

Keylogger (Keystroke Logger)

Je to software alebo hardware zaznamenávajúci stlačenie klávesov na napadnutom zariadení. Cieľom je zachytiť prihlasovacie údaje, čísla kreditných kariet a iné dôverné informácie.

Detekcia malvéru typu Trójsky kôň

Pre účely nášho inovatívneho SOC riešenia sme sa v rámci pilotného testovania zamerali len na malvér typu Trójsky kôň (angl. trojan), na metódy jeho detekcie, ktoré pre analýzu využívajú monitorovanie prevádzky v počítačovej sieti. Najprv opíšeme spôsoby šírenia trojských koní a ich vlastnosti, ktoré sa prejavia na prevádzke v sieti a môžu slúžiť na detekciu trojských koní.

Spôsoby šírenia trojských koní

Trojské kone sa môžu šíriť v počítačovej sieti rôznymi spôsobmi, často ich šíreniu pomáha sociálne inžinierstvo – zamerané na oklamanie používateľa alebo využívajú zraniteľností systému.

Pri sociálnom inžinierstve sa trojský kôň vydáva za legitímny softvér a presviedča používateľa, aby ho dobrovoľne stiahol a spustil. Môže to byť:

- **E-mailové prílohy a phishing** - jeden z najbežnejších spôsobov. Útočníci posielajú e-maily, ktoré vyzerajú dôveryhodne (napr. ako faktúra, oznámenie banky alebo správa od kolegu) a nabádajú používateľa, aby otvoril škodlivú prílohu alebo klikol na odkaz.
- **Stiahnutie softvéru z nedôveryhodných zdrojov** - používatelia si nevedomky stiahnu trójskeho koňa, keď sťahujú pirátsky softvér, cracky, hry alebo iné programy z neoficiálnych alebo pochybných webových stránok.
- **Sociálne siete a služby na zdieľanie súborov** – tu šírenie môže prebiehať aj cez škodlivé odkazy alebo súbory zdieľané cez chatovacie aplikácie, sociálne médiá alebo peer-to-peer siete.

Príklady využitia **zraniteľností systému**:

- **Zneužitie zraniteľností prehliadača (Exploity)** - návšteva infikovanej webovej stránky môže automaticky spustiť stiahnutie a inštaláciu trójskeho koňa bez vedomia používateľa, a to využitím bezpečnostných chýb v internetovom prehliadači alebo jeho doplnkoch.

- **Jednotky USB a vymeniteľné médiá** - infikované USB kľúče môžu preniesť trójskeho koňa do iného počítača, keď sú pripojené a automatické spúšťanie (autorun) je povolené.

Takže na rozdiel od vírusov a červov sa trójske kone nešíria samy od seba. Vyžadujú si interakciu používateľa (napr. otvorenie súboru, inštaláciu programu) alebo zneužitie konkrétnej zraniteľnosti na systéme, aby sa nainštalovali a spustili.

Ako sa dá zo sledovania komunikácie v počítačovej sieti odhaliť trojský kôň?

Na detekciu trójskych koní v počítačovej sieti sa používajú rôzne metódy a nástroje. Zvyčajne ide o kombináciu monitorovania sieťového prenosu, analýzy koncových zariadení a dodržiavania osvedčených bezpečnostných postupov:

1. Použitie bezpečnostného softvéru

- **Antivírusové a antimalvérové programy** Tieto programy sú základom ochrany. Skenujú súbory a procesy na koncových zariadeniach a hľadajú známe signatúry (digitálne odtlačky) trójskych koní. Moderné programy využívajú aj heuristickú analýzu, ktorá dokáže identifikovať podozrivé správanie neznámeho malvéru.
- **Systémy na detekciu a prevenciu prienikov (IDS/IPS)** Tieto systémy monitorujú **sieťovú prevádzku** a hľadajú vzory, ktoré by mohli naznačovať aktivitu trójskeho koňa alebo útočníka, ako napríklad **neobvyklé porty, pokusy o neoprávnený prístup alebo komunikáciu so známymi škodlivými servermi** (C&C servery - Command & Control servery - koordinuje akcie infikovaných počítačov, poskytuje pokyny, aktualizácie a môže slúžiť na nahrávanie ukradnutých dát).

2. Monitorovanie sieťovej prevádzky

- **Analýza tokov (NetFlow, sFlow)** Správcovia siete môžu monitorovať objem a typ prenášaných dát. Neobvyklé **zvýšenie odchádzajúcej prevádzky do neznámych destinácií** môže byť znakom exfiltrácie dát trójskym koňom.
- **Monitorovanie DNS požiadaviek** Mnohé trójske kone komunikujú **so svojimi riadiacimi servermi cez DNS**. Analýza neobvyklých alebo častých požiadaviek na podozrivé domény môže pomôcť pri ich detekcii.

3. Analýza správania a anomálií

- **Behaviorálna analýza** Nástroje na analýzu správania (napr. EDR - Endpoint Detection and Response) sa učia normálne vzorce správania používateľov a aplikácií v sieti. Akákoľvek významná odchýlka – napríklad program, ktorý zrazu začne pristupovať k veľkému množstvu citlivých súborov alebo sa pokúša spustiť systémové príkazy – je označená ako podozrivá aktivita.

- **Pravidelné audity a logovanie** Pravidelná kontrola systémových logov (záznamov udalostí) na serveroch a koncových staniciach môže odhaliť neoprávnené zmeny konfigurácie, vytvorenie nových používateľských účtov alebo inštaláciu neautorizovaného softvéru.

4. Osvedčené postupy

- **Firewall** Správne nakonfigurovaný firewall obmedzuje neoprávnený prístup do siete
  a blokuje komunikáciu na neštandardných portoch, čím sťažuje trójskym koňom komunikáciu s vonkajškom.
- **Vzdelávanie používateľov** Sociálne inžinierstvo je bežná metóda šírenia trójskych koní. Školenie zamestnancov o rozpoznávaní phishingových e-mailov a nebezpečných webových stránok výrazne znižuje riziko infekcie.
- **Pravidelné aktualizácie** Udržiavanie operačných systémov a softvéru aktuálnych pomáha zaplátať bezpečnostné zraniteľnosti, ktoré by trójske kone mohli zneužiť na infiltráciu siete.

Kombináciou týchto techník a nástrojov možno výrazne zvýšiť šance na včasnú detekciu a neutralizáciu trójskych koní v sieti.

Trojské kone môžu generovať sieťovú prevádzku, podľa nej ich chceme odhaliť

Ako je uvedené vyššie, trojské kone sa môžu prejaviť aj generovaním podozrivej sieťovej aktivity alebo prevádzky, ktorá môže indikovať kompromitované zariadenie.

Príklady takej komunikácie generovanej trojskými koňmi sú:

Neautorizovaná komunikácia - trojský kôň môže komunikovať so vzdialeným riadiacim serverom, ktorý je ovládaný útočníkom a od ktorého preberá inštrukcie alebo naň odosiela ukradnuté dáta.

DDoS útoky - niektoré trojany zmenia napadnuté zariadenie na „zombie počítač“, ktorý je súčasťou siete botnetov. Táto sieť infikovaných počítačov potom môže byť použitá na zaplavenie servera veľkým objemom prevádzky – na DDoS (Distributed Denial of Service) útok.

Stiahnutie ďalšieho malvéru – bežný typ trojana (downloader trojan) použije internetové pripojenie na stiahnutie a inštaláciu ďalších škodlivých (malicious) programov do infikovaného systému.

Increased bandwidth usage:

Aktivity trojanu, ako je posielanie veľkého objemu ukradnutých dát, alebo účasť na sieti botnet, môžeš viesť k neočakávaným špičkám v sieťovej aktivite a spomaliť sieťové pripojenie pre používateľa.

Keďže sa aktivita trojských koní môže prejaviť na nezvyčajnej sieťovej prevádzke, a skúmali sme modely strojového učenia, ktoré detegujú trojany na základne údajov o prevádzke v sieti.

Experimenty a výsledky

Datasety

Dataset: Trojan_Detection

Pre trénovanie modelov na detekciu malvéru typu „trojský kôň“ (trojan horse) na základe údajov o sieťovej prevádzke bol použitý dataset Trojan_Detection:

github.com/Shreekavinmr/Trojan-horse-detection/blob/main/dataset.pdf, www.kaggle.com/datasets/subhajournal/trojan-detection/data – Download.

Tieto dáta pochádzajú pôvodne z web portálu Kanadského inštitútu pre kybernetickú bezpečnosť (CIC) www.unb.ca/cic/datasets/index.html.

Dataset Trojan_Detection má veľkosť po rozbalení 93.89 MB a obsahuje záznamy o tokoch v počítačovej sieti. Paketový tok tu označuje sériu paketov, pri komunikácii dvoch aplikácii - tie majú rovnakú päticu údajov – zdrojovú a cieľovú IP adresu, zdrojový a cieľová port, a typ protokolu.

Dataset obsahuje záznamy dvoch typov (škodlivé – Trojan a neškodné - Benign):

- Trojan Data - 90683 pozorovaní – toky škodlivé (malígne) generované trojskými koňmi
- Benign Data - 86799 pozorovaní – toky neškodnej (benígnej) sieťovej prevádzky

V datasete je pre každý tok zaznamenaných 86 parametrov:

Flow ID	Source IP	Source Port
Destination IP	Destination Port	Protocol
Timestamp	Flow Duration	Total Fwd Packets
Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets
Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean
Fwd Packet Length Std	Bwd Packet Length Max	Bwd Packet Length Min
Bwd Packet Length Mean	Bwd Packet Length Std	Flow Bytes/s
Flow Packets/s	Flow IAT Mean	Flow IAT Std
Flow IAT Max	Flow IAT Min	Fwd IAT Total
Fwd IAT Mean	Fwd IAT Std	Fwd IAT Max
Fwd IAT Min	Bwd IAT Total	Bwd IAT Mean
Bwd IAT Std	Bwd IAT Max	Bwd IAT Min

Fwd PSH Flags	Bwd PSH Flags	Fwd URG Flags
Bwd URG Flags	Fwd Header Length	Bwd Header Length
Fwd Packets/s	Bwd Packets/s	Min Packet Length
Max Packet Length	Packet Length Mean	Packet Length Std
Packet Length Variance	FIN Flag Count	SYN Flag Count
RST Flag Count	PSH Flag Count	ACK Flag Count
URG Flag Count	CWE Flag Count	ECE Flag Count
Down/Up Ratio	Average Packet Size	Avg Fwd Segment Size
Avg Bwd Segment Size	Fwd Header Length.1	Fwd Avg Bytes/Bulk
Fwd Avg Packets/Bulk	Fwd Avg Bulk Rate	Bwd Avg Bytes/Bulk
Bwd Avg Packets/Bulk	Bwd Avg Bulk Rate	Subflow Fwd Packets
Subflow Fwd Bytes	Subflow Bwd Packets	Subflow Bwd Bytes
Init_Win_bytes_forward	Init_Win_bytes_backward	act_data_pkt_fwd
min_seg_size_forward	Active Mean	Active Std
Active Max	Active Min	Idle Mean
Idle Std	Idle Max	Idle Min
Class – Trojan/Benign		

Na stránke www.kaggle.com/code/proloypal/trojan-detection-cnn-97-accuracy/data si možno dataset prezerať (na obrázok bolo vybraných 11 z 86 stĺpcov):

Trojan_Detection.csv (93.89 MB)

Detail Compact Column 11 of 86 columns

#	Flow ID	Source IP	Source Port	Destinatio...	Destinat
96676	10.42.0.1-10.42.0.42-53-37749-17	10.42.0.42	37749	10.42.0.1	53
42891	10.42.0.1-10.42.0.42-53-41352-17	10.42.0.42	41352	10.42.0.1	53
169326	10.42.0.151-107.22.241.77-44353-443-6	10.42.0.151	44353	107.22.241.77	443
34510	10.42.0.211-10.42.0.1-6021-53-17	10.42.0.211	6021	10.42.0.1	53

# Protocol	Timestamp	# Flow Durat...	# Total Fwd ...	Class
17	30/06/2017 07:16:12	1023244	1	Benign
17	13/07/2017 03:48:44	286483	1	Trojan
6	05/07/2017 10:47:35	65633087	12	Benign
17	11/07/2017 05:01:34	251336	1	Trojan

Použité algoritmy a metódy

Náhodný les (Random Forest)

Algoritmus „náhodný les“ sa používa pre klasifikáciu neznámych objektov do tried. Napríklad má rozhodnúť, či klientov banky bude alebo nebude schopný splácať pôžičku, alebo či paketový tok v sieti patrí neškodnej bežnej prevádzke, alebo je to nezvyčajný - anomálny tok, ktorý môže byť spôsobený malvérom.

Náhodný les nechá metóde CART zostaviť zvolený počet rozhodovacích stromov a klasifikáciu neznámych objektov robí tak, že nechá tieto stromy hlasovať o tom, do ktorej triedy jednotlivé objekty patria.

Algoritmus vychádza z trénovacích dát, ktoré obsahujú informácie o objektoch - parametre (prediktory) a príslušnosť k triede (predikovaná veličina). Parametrami môžu

byť napr. vek klienta, príjem, počet detí, národnosť, alebo v prípade sieťovej prevádzky – zdrojová IP adresa, zdrojový port, protokolu, a ďalšie.

Model „náhodný les“ trénujeme tak, že zvolíme počet stromov (n_estimators), ktoré má les obsahovať, a zvolíme počet prediktorov, ktoré sa majú pri každom pestovaní stromu použiť. Potom sa pre vygenerovanie každého stromu - náhodne pripraví výber záznamov z trénovacej množiny (metódou bootstrap - výber s opakovaním) a opakujú sa vetvenia stromu krokmi:

- z množiny prediktorov náhodne vyberieme požadovaný počet prediktorov
- použijeme metódu CART (pozri nižšie) na nájdenie najlepšieho rozhodovacieho pravidla pre rozdelenie vybraných záznamov
- konštrukcia rozhodovacieho stromu skončí, keď bude splnené ukončovacie kritérium, (maximálna hĺbka stromu, minimálny počet prípadov v koncovom liste stromu, alebo list obsahuje len prvky rovnakej triedy)

Použitú metódu CART charakterizujeme nižšie.

CART (Classifi-cation And Regression Trees, Klasifikačné a regresné stromy)

Algoritmus CART sa používa na riešenie klasifikačnej úlohy. Patrí medzi rozhodovacie stromy opísané v kapitole Analýza ML metód a algoritmov pre detekciu spam emailových správ. Hlavnou myšlienkou CART je natrénovanie binárneho stromu – ktorý v každom kroku delenia vyberá zo všetkých možných rozhodovacích klasifikačných podmienok tú klasifikačnú podmienku, ktorá je „najlepšia“ z pohľadu tzv. „čistoty“ množín, ktoré získame delením.

Zostavovanie klasifikačných podmienok metódy CART algoritmu vysvetlíme na príklade klientov banky, o ktorých vieme tzv. prediktory - počet detí (0, 1, 2, 3) a národnosť (Slovák, Čech a Maďar). A ďalej , vieme, či boli alebo neboli schopní splácať úver, takže máme dve triedy, ktoré chceme predpovedať pre nových uchádzačov o úver.

Máme tu dva druhy prediktorov, z ktorých každý umožňuje zostaviť viacero vetviacich podmienok:

- ordinárne premenné – počet detí (0, 1, 2, 3) – tam môžeme zostaviť vetviace podmienky „počet detí“ < 1, < 2, < 3, 3
- kategorické premenné – Slovák, Čech, Maďar – tam sú možné vetvenia
    - Slovák – (Čech, Maďar), Čech – (Maďar, Slovák), Maďar – (Slovák, Čech)

Potom v každom kroku, kde sa v metóde CART rozhoduje o vetvení, sa preskúmajú všetky vetviace podmienky a vyberie sa “najlepšia” podmienka.

Pre určenie “najlepšej” podmienky vetvenia sa najčastejšie používajú funkcie “Gini index” alebo “entropia”, tieto kritéria merajú “homogénnosť“ prvkov jednotlivých tried

v množinách, ktoré vzniknú vetvením. Pre túto vlastnosť sa používa aj pojem „nečistota“ (impurity). Cieľom je aby sa vybralo také vetvenie, ktoré čo najviac zmenší „nečistotu“.

Viac informácií možno nájsť v podkladovom dokumente:

- KREDITNÝ SKÓRING POMOCOU NÁHODNÝCH LESOV.pdf
- DECISION TREES - CLASSIFICATION AND REGRESSION TREES – CART.pdf

AdaBoost a Gradient Boosting Classifier

AdaBoost (Adaptive Boosting) je populárna metóda strojového učenia, ktorá patrí do kategórie ensemble metód (metódy súborov modelov). Jej hlavným cieľom je vytvoriť silný klasifikátor kombináciou viacerých slabých klasifikátorov (tzv. "weak learners"), čo sú zvyčajne jednoduché modely, napríklad rozhodovacie stromy s hĺbkou 1.

Princíp metódy AdaBoost spočíva v adaptívnom prístupe, kde sa modely trénujú postupne v sérii krokov, pričom každý ďalší model sa zameriava na chyby, ktorých sa dopustili modely predchádzajúce.

Metóda AdaBoost má nasledujúce kroky:

1. Inicializácia váh údajov (záznamov)

Každá trénovacia vzorka v datasete má na začiatku pridelenú rovnakú váhu. To znamená, že všetky vzorky sú považované za rovnako dôležité pre prvý tréning.

2. Iteratívny tréning slabých klasifikátorov

Tréning prebieha v cykloch. V každej iterácii (kroku) obsahuje:

Tréning: Je natrénovaný nový slabý klasifikátor s cieľom minimalizovať chybu na aktuálne prevážených trénovacích dátach.

Výpočet chyby: Vypočíta sa miera chyby (vážená chyba) tohto nového modelu.

Výpočet váhy klasifikátora: Model, ktorý dosiahol nižšiu chybovosť, získa vyššiu váhu vo finálnom súbore modelov (tzn. jeho hlas bude pri klasifikácií neznámych objektov dôležitejší). Model s vysokou chybou dostane nízku váhu.

3. Adaptívne preváženie údajov

Vzorkám, ktoré aktuálny klasifikátor nesprávne zaradil, zvýšime váhy. Vzorkám, ktoré boli správne zaradené, budú mať váhu znížený. Toto váženie vzoriek zabezpečí, že nasledujúci slabý klasifikátor sa sústredí predovšetkým na vzorky, ktoré boli nesprávne klasifikované.

4. Kombinácia (Agregácia)

Po dokončení všetkých iterácií je vytvorený finálny silný klasifikátor, ktorý je váženou lineárnou kombináciou všetkých natrénovaných slabých klasifikátorov. Pri predikcii sa

sčítajú predpovede jednotlivých modelov, pričom každý hlas je vážený podľa jeho presnosti (váhy klasifikátora z kroku 2).

Výhod AdaBoost:

- jednoduchosť - ľahko sa implementuje a často používa jednoduché slabé klasifikátory
- vysoká presnosť – vie dosahovať veľmi vysokú presnosť klasifikácie
- odolnosť voči pretrénovaniu (overfittingu) - ak sa používa so slabými klasifikátormi, obyčajne sa nemá taký problém s pretrénovaním ako iné metódy.

Metóda AdaBoost je robustným a efektívnym algoritmom, ktorý tvorí základ mnohých pokročilejších boostingových techník (ako sú Gradient Boosting Machines či XGBoost) a je často používaný v praxi.

Gradient Boosting Classifier je výkonný algoritmus strojového učenia, ktorý kombinuje veľa "slabých" modelov (zvyčajne rozhodovacích stromov) do jedného "silného", čím postupne opravuje chyby predchádzajúcich modelov, aby dosiahol presné predikcie v klasifikačných úlohách. Každý nový strom sa učí predpovedať chyby kombinovaného modelu z predchádzajúcich iterácií. Konečný predikčný model je súčet predikcií všetkých týchto slabých modelov. Používa sa gradient na optimalizáciu a "krokovanie" smerom k minimalizácii chybovej funkcie, podobne ako v neurónových sieťach.

Viac informácií možno nájsť v podkladovom dokumente:

- Ensamble metody obecně Bagging - náhodné lesy - Boost.pdf.

Gaussian Naive Bayes

Metóda Gaussian Naive Bayes je založená na Bayesovej vete a dvoch kľúčových predpokladoch: "naivnej" nezávislosti atribútov a tom, že spojité dáta majú Gaussovo (normálne) rozdelenie.

Algoritmus využíva Bayesovu vetu na výpočet podmienenej pravdepodobnosti, že dátový bod patrí do určitej triedy, vzhľadom na jeho vlastnosti (príznaky) - snaží sa určiť, ktorá trieda najpravdepodobnejšie generuje body s takýmito atribútmi.

Metóda predpokladá, že všetky vysvetľujúce premenné (príznaky) sú navzájom nezávislé. To znamená, že prítomnosť jedného príznaku neovplyvňuje prítomnosť žiadneho iného príznaku v rámci danej triedy. Tento predpoklad je v reálnych dátach často zjednodušujúci (a preto "naivný"), ale v praxi algoritmus napriek tomu dosahuje dobré výsledky.

Pre Gaussian Naive Bayes sa tiež predpokladá, že hodnoty každého spojitého príznaku sú rozdelené podľa Gaussovho (normálneho) rozdelenia.

- algoritmus odhadne parametre tohto rozdelenia (priemer a rozptyl) pre každý príznak v rámci každej triedy
- potom pri predikcii sa na základe týchto parametrov a hustoty pravdepodobnosti vypočíta pravdepodobnosť pozorovania konkrétnej hodnoty príznaku pre danú triedu.

Klasifikácia sa vykonáva v nasledujúcich krokoch:

1. Odhad parametrov: Počas tréningu sa pre každú triedu a každý príznak vypočítajú priemer a rozptyl.
2. Výpočet pravdepodobností: Pre nový, neznámy dátový bod (záznam) sa pomocou hustoty Gaussovho rozdelenia vypočíta pravdepodobnosť jeho príznakov pre každú z možných tried (pravdepodobnosť, že ak objekt patrí do danej triedy, tak jeho príznak nadobúda takúto hodnotu).
3. Aplikácia Bayesovej vety: Tieto pravdepodobnosti (hodnôt atribútov) sa skombinujú pomocou Bayesovej vety (vďaka predpokladu nezávislosti sa vynásobia a výsledok sa násobí aj pravdepodobnosťou výskytu danej triedy) - vypočíta sa celková posteriorna pravdepodobnosť, že dátový bod patrí do každej triedy.
4. Predikcia: Trieda s najvyššou vypočítanou pravdepodobnosťou sa určí ako výsledná predikcia pre daný dátový bod.

Deep Learning with Feedforward Neural Network

Deep Learning with Feedforward Neural Network (hlboké učenie s doprednou neurónovou sieťou) patrí medzi metódy strojového učenia, ktorá sa považuje za ďalší krok v oblasti neurónových sietí. Hlboké neurónové siete majú viacvrstvové architektúry neurónov.

Princíp hlbokého učenia s doprednými (feedforward) neurónovými sieťami spočíva v tom, že vstupné dáta prechádzajú cez viaceré vrstvy neurónov, kde každá vrstva spracúva informácie a výstup z jednej vrstvy slúži ako vstup pre nasledujúcu. Tieto siete sú zvyčajne zložené z vstupnej vrstvy, jednej alebo viacerých skrytých vrstiev a výstupnej vrstvy. Každý neurón v jednej vrstve je spojený s neurónmi v nasledujúcej vrstve, a každé spojenie má priradenú váhu, ktorá sa počas trénovania upravuje. Trénovanie siete sa uskutočňuje tak, že vstupné vzorky sú prechádzané sieťou a výstup je porovnaný s očakávaným výstupom. Rozdiel medzi týmito hodnotami sa používa na úpravu váh pomocou algoritmu spätného šírenia chyby (backpropagation), ktorý vypočíta gradient chyby vzhľadom na každú váhu. Tento gradient ukazuje smer, v ktorom sa chyba zvyšuje najviac, a preto sa váhy upravujú v opačnom smere, čo vedie k poklesu chyby. Tento proces učenia sa opakuje iteratívne pre rôzne dáta, čo sa nazýva stochastický pokles gradientu (SGD - Stochastic Gradient Descent).

V prípade, že vstupné dáta obsahujú šum, môže sa sieť pokúsiť naučiť aj nežiaduce nekorektné vzory. Aby sa zabránilo pretrénovaniu (overfitting), kde sieť sa naučí šum namiesto skutočných vzorov, používa sa regulácia (regularization), napríklad L2 regulácia, ktorá obmedzuje veľkosť váh a výsledná rozhodovacia hranica je hladšia a reálnejšia. Táto metóda pomáha sieti učiť sa „pekné“ rozhodovacie hranice, ktoré sú menej citlivé na šum v dátach.

Výhodou hlbokých sietí

je ich schopnosť automaticky sa učiť vlastné vzory z dát, čo je výhodné pri zložitých úlohách ako je klasifikácia obrázkov.

Výhodou hlbokých neurónových sietí s feedforward architektúrou je, že sú schopné automaticky sa učiť z dát zložité nelineárne vzory a sú efektívne v mnohých oblastiach, vrátane klasifikácie obrázkov, detekcie malwaru alebo analýzy lekárskych snímok.

Príprava a postup výpočtov

Pre detekciu sme použili modely zapísané v skripte „Trojan-horse-detection“ z portálu: https://github.com/Shreekavinmr/Trojan-horse-detection/blob/main/README.md .

Práca s modelmi pozostávala z nasledujúcich krokov.

1. Predspracovanie dát (data preprocessing)

- Nahratie datasetu pre detekciu trojanov pomocou knižnice pandas.
- Čistenie dát (data cleaning): vynechanie nerelevantných a duplicitných záznamov, podľa potreby doplnenie chýbajúcich dát.
- Kódovanie kategorických dát (label encoding): prevod kategorických dát ako sú identifikátor toku (Flow ID), zdrojová a cieľová IP adresa (Source IP, Destination IP), časová pečiatka (Timestamp) – na numerické hodnoty.
- Kódovanie návestí tried (Class Label Encoding): Prevod označenia triedy ("Trojan" a "Benign") na binárny tvar (1 resp 0).
- Výber príznakov (Feature Selection): Použiť vzájomné informácie na výber príznakov, ktoré sú najdôležitejšie pre klasifikáciu

2. Rozdelenie dát na trénovacie a testovacie

Rozdelenie dát na trénovaciu a testovaciu množinu (typicky býva 80% trénovacích a 20% testovacích dát). Na rozdelenie sa použila funkcia using train_test_split z knižnice scikit-learn.

3. Výpočty algoritmami strojového učenia (Machine Learning): Náhodný les (Random Forest), AdaBoost (**a**daptive **b**oosting), Gaussian Naive Bayes a Hlboké učenie s doprednou neurónovou sieťou. Nasleduje popis použitých algoritmov strojového učenia.

Náhodný les (Random Forest)

Trénovanie modelu Random Forest: Inicializácia a trénovanie klasifikátora Random Forest Classifier s vhodným počtom estimátorov (50 stromov). Vyhodnotenie presnosti (accuracy) modelu na testovacej množine.

Ladenie hyperparametrov: Použiť rôzne počty estimátorpv – nájdi optimálnu konfiguráciu.

AdaBoost

Trénovanie modelu AdaBoost: Inicializovať a trénovať klasifikátor AdaBoost (napr. so 100 odhadcami). Vyhodnoť presnosť modelu na testovacej sade.

Gaussian Naive Bayes

Trénovanie modelu: Inicializuje sa a trénuje gaussovský naivný Bayesovský klasifikátor.

Krížová validácia: Aplikuje sa krížová validácia pre zistenie robustnejšieho odhadu úspešnosti modelov.

Deep Learning with Feedforward Neural Network

Inštalácia knižníc: Nainštalujte knižnice TensorFlow, scikit-learn a pandas.

Predspracovanie dát pre neurónovú sieť: Štandardizujte vstupné dáta pomocou StandardScaler. Definujte architektúru doprednej neurónovej siete s vhodnými aktivačnými funkciami a vrstvami.

Trénovanie neurónovej siete: Kompilovanie a trénovanie neurónovej siete na predspracovaných dátach počas zadaného počtu epoch.

Nastavenia a parametre zvolených metód

Náhodný les (Random Forest)

počtom estimátorov: 50 stromov

ladenie hyperparametrov: 10, 20, 30, 40, 50, 60, 70, 80, 90 stromov

Nastavenie parametrov:

```
clf.get_params(deep=True)
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
```

```
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 50,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Defoltné nastavenie parametrov podľa

Nastavenie parametrov:

scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

AdaBoost

počtom estimátorov: 100 stromov

Nastavenie parametrov:

```
Aclf = AdaBoostClassifier(n_estimators=100)
Aclf.get_params(deep=True)
{'algorithm': 'deprecated',
 'estimator': None,
 'learning_rate': 1.0,
 'n_estimators': 100,
 'random_state': None}

Gclf = GradientBoostingClassifier(n_estimators=100,
learning_rate=1.0,max_depth=1, random_state=0).fit(X_train, y_train)
Gclf.get_params(deep=True)
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 1.0,
 'loss': 'log_loss',
 'max_depth': 1,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 0,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Gaussian Naive Bayes

Trénovanie modelu: Inicializuje sa a trénuje gaussovský naivný Bayesovský klasifikátor.

Krížová validácia: Aplikuje sa krížová validácia pre zistenie robustnejšieho odhadu úspešnosti modelov.

Nastavenie parametrov:

```
gnb = GaussianNB()
gnb.get_params(deep=True)
{'priors': None, 'var_smoothing': 1e-09}
```

https://scikit-learn.org/0.22/modules/generated/sklearn.naive_bayes.GaussianNB.html

Deep Learning with Feedforward Neural Network

Inštalácia knižníc: Nainštalujte knižnice TensorFlow, scikit-learn a pandas.

Predspracovanie dát pre neurónovú sieť: Štandardizujte vstupné dáta pomocou StandardScaler. Definujte architektúru doprednej neurónovej siete s vhodnými aktivačnými funkciami a vrstvami.

Trénovanie neurónovej siete: Kompilovanie a trénovanie neurónovej siete na predspracovaných dátach počas zadaného počtu epoch.

Použitá neurónová sieť má nasledujúcu architektúru:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer with
sigmoid activation for binary classification
])

model.get_config()
{'name': 'sequential',
 'trainable': True,
 'dtype': {'module': 'keras',
  'class_name': 'DTypePolicy',
  'config': {'name': 'float32'},
  'registered_name': None},
 'layers': [{'module': 'keras.layers',
   'class_name': 'InputLayer',
   'config': {'batch_shape': (None, 84),
    'dtype': 'float32',
    'sparse': False,
    'ragged': False,
    'name': 'input_layer'},
   'registered_name': None},
```

```
{'module': 'keras.layers',
 'class_name': 'Dense',
 'config': {'name': 'dense',
  'trainable': True,
  'dtype': {'module': 'keras',
   'class_name': 'DTypePolicy',
   'config': {'name': 'float32'},
   'registered_name': None},
  'units': 64,
  'activation': 'relu',
  'use_bias': True,
  'kernel_initializer': {'module': 'keras.initializers',
   'class_name': 'GlorotUniform',
   'config': {'seed': None},
   'registered_name': None},
  'bias_initializer': {'module': 'keras.initializers',
   'class_name': 'Zeros',
   'config': {},
   'registered_name': None},
  'kernel_regularizer': None,
  'bias_regularizer': None,
  'kernel_constraint': None,
  'bias_constraint': None},
 'registered_name': None,
 'build_config': {'input_shape': (None, 84)}},
{'module': 'keras.layers',
 'class_name': 'Dense',
 'config': {'name': 'dense_1',
  'trainable': True,
  'dtype': {'module': 'keras',
   'class_name': 'DTypePolicy',
   'config': {'name': 'float32'},
   'registered_name': None},
  'units': 32,
  'activation': 'relu',
  'use_bias': True,
  'kernel_initializer': {'module': 'keras.initializers',
   'class_name': 'GlorotUniform',
   'config': {'seed': None},
   'registered_name': None},
  'bias_initializer': {'module': 'keras.initializers',
   'class_name': 'Zeros',
   'config': {},
   'registered_name': None},
  'kernel_regularizer': None,
  'bias_regularizer': None,
  'kernel_constraint': None,
  'bias_constraint': None},
 'registered_name': None,
 'build_config': {'input_shape': (None, 64)}},
{'module': 'keras.layers',
 'class_name': 'Dense',
 'config': {'name': 'dense_2',
  'trainable': True,
  'dtype': {'module': 'keras',
   'class_name': 'DTypePolicy',
   'config': {'name': 'float32'},
   'registered_name': None},
  'units': 1,
  'activation': 'sigmoid',
  'use_bias': True,
```

```
    'kernel_initializer': {'module': 'keras.initializers',
     'class_name': 'GlorotUniform',
     'config': {'seed': None},
     'registered_name': None},
    'bias_initializer': {'module': 'keras.initializers',
     'class_name': 'Zeros',
     'config': {},
     'registered_name': None},
    'kernel_regularizer': None,
    'bias_regularizer': None,
    'kernel_constraint': None,
    'bias_constraint': None},
   'registered_name': None,
   'build_config': {'input_shape': (None, 32)}}],
 'build_input_shape': (None, 84)}
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Trénovanie prebehlo počas 10 epoch, s veľkosťou batch = 64:

```
model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_split=0.2)
```

Vyhodnocovanie experimentov

Pre vyhodnocovanie experimentov boli použité nasledovné metriky.

Konfúzna matica

Konfúzna matica je jedným z najpopulárnejších spôsobov vyhodnocovania akéhokoľvek klasifikačného modelu. Hoci matica sama o sebe nie je metrikou, maticovú reprezentáciu možno použiť na definovanie rôznych metrík, z ktorých všetky sa stanú dôležitými v nejakom konkrétnom prípade alebo scenári. Konfúznu maticu je možné vytvoriť pre binárnu klasifikáciu, rovnako ako aj pre klasifikačný model s viacerými triedami.

Konfúzna matica sa vytvorí porovnaním predikovaného označenia triedy dátového bodu s jeho skutočným označením triedy. Toto porovnanie sa zopakuje pre celý súbor dát (dataset) a výsledky tohto porovnania sa zostavia v maticovom alebo tabuľkovom formáte. Táto výsledná matica predstavuje konfúznu maticu (obr. nižšie).

		Skutočná trieda	
		Kladná	Záporná
Predikovaná trieda	Kladná	Počet správne kladných (TP)	Počet nesprávne kladných (FP)
	Záporná	Počet nesprávne záporných (FN)	Počet správne záporných (TN)

Obr. 3 Konfúzna matica

Vo všeobecnosti máme zvyčajne 2 triedy – kladnú/pozitívnu a zápornú/negatívnu triedu. Na základe tejto štruktúry môžeme jasne vidieť štyri pojmy dôležitosti.

• True Positive (TP) – Počet správne kladných: Toto je suma celkového počtu príkladov (dátových bodov z datasetu) z pozitívnej triedy, kde sa skutočné označenie triedy rovnalo predpovedanému označeniu triedy, t. j. celkovým počtom prípadov, v ktorých sme správne predpovedali označenie pozitívnej triedy pomocou nášho modelu.

• False positive (FP) – Počet nesprávne kladných: Toto je počet celkových prípadov z negatívnej triedy, kde ich model nesprávne klasifikoval tým, že ich predpovedal ako pozitívne. Odtiaľ pochádza názov, falošne pozitívny.

• True Negative (FN) – Počet nesprávne záporných: Toto je počet celkového počtu príkladov z negatívnej triedy, kde sa skutočné označenie triedy rovnalo predpovedanému označeniu triedy, t. j. celkový počet prípadov, v ktorých sme správne predpovedali označenie negatívnej triedy pomocou nášho modelu.

• False negatívne (FN) – Počet nesprávne záporných: Toto je počet celkového počtu prípadov z pozitívnej triedy, kde ich náš model nesprávne klasifikoval tým, že ich predpovedal ako negatívne. Odtiaľ pochádza názov, falošne negatívny.

Accuracy

Táto metrika je jednou z najpopulárnejších spôsobov merania resp. vyhodnocovania efektívnosti ľubovoľného klasifikátora. Je definovaná ako celková presnosť alebo podiel správnych predpovedí modelu. Vzťah pre výpočet z konfúznej matice je nasledovný:

$$A = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision

Precision (presnosť), známa aj ako kladná prediktívna hodnota, je ďalšou metrikou, ktorú možno odvodiť z konfúznej matice. Táto metrika je definovaná ako počet predikcií, ktoré sú skutočne správne alebo relevantné zo všetkých predikcií založených na kladnej triede. Vzorec pre túto metriku je nasledovný:

$$P = \frac{TP}{TP + FP}$$

Model s vysokou hodnotou presnosti identifikuje vyšší podiel kladnej triedy v porovnaní s modelom s nižšou presnosťou. Táto metrika sa stáva dôležitou v prípadoch, keď sa viac zaujímame o nájdenie maximálneho počtu kladných tried, aj keď sa celková presnosť zníži.

Recall

Recall (úplnosť) je metrika modelu na identifikáciu percenta relevantných dátových bodov. Je definovaná ako počet inštancií kladnej triedy, ktoré boli správne predikované. Označuje sa aj ako miera pokrytia alebo citlivosť. Vzťah pre vypočítanie tejto metriky je nasledovný:

$$R = \frac{TP}{TP + FN}$$

Táto metrika sa stáva dôležitým ukazovateľom efektívnosti klasifikátora v scenároch, v ktorých chceme zachytiť čo najväčší počet prípadov konkrétnej triedy, aj keď to spôsobí zvýšenie falošných detekcií/predikcií. Ako príklad možno uviesť prípad bankového podvodu, model s vysokou hodnotou metriky recall nám poskytne vyšší počet prípadov potenciálnych podvodov. Rovnako to však zvýši vyvolanie poplachov pre väčšinu podozrivých prípadov.

F1-Score

Existujú prípady, v ktorých chceme dosiahnuť vyváženú optimalizáciu presnosti a úplnosti. F1-Score je metrika, ktorá je harmonickým priemerom metrík precision a recall a pomáha nám optimalizovať klasifikátor pre vyváženú presnosť a úplnosť predikcie. Vzorec pre výpočet F1-Score je nasledovný:

$$F = \frac{2.P.R}{P + R}$$

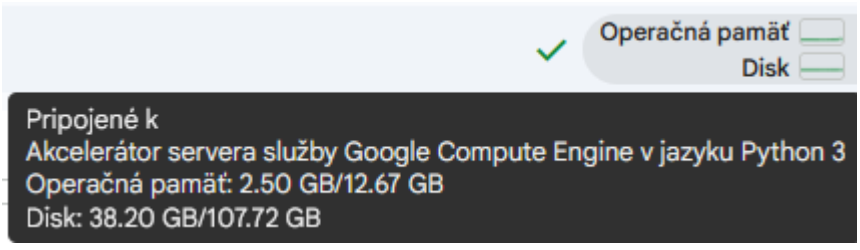
Výsledky

Dataset: TrojanDetec (výpočty na Google Colab - Processing time)

Metóda	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Processing time (sec.)
Random Forest classifier (50 estimators)	0.999967	0.999967	0.999967	0.999967	9 sek
skúšať rôzne počty stromov: 10	99.990172%				ďalšie spolu 60 sek.
20	99.993448%				
30	99.996724%				
40	100.000000%				
50	99.993448%				
60	99.996724%				
70	99.996724%				

80	99.996724%				
90	99.993448%				
AdaBoostClassifier (100 estimators)	0.9996396278338356				20 sek.
cross-validation to obtain a more robust estimate	0.9996265210301141				60 sek.
GaussianNB Classifier					20
gnb.score	91.498493%				
cross_val_score	90.108052%				
using reciver operating characteristic score	92.183155%				
weighted avg		0.921035	0.914985	0.914487	
Deep learning (Feed forward neural network) – 10 epoch	0.983947				55 sek. samotný trénning
weighted avg		0.983953	0.983947	0.983947	

2025 12 05: výpočty - obrazovky



Random Forest classifier

[29] ✓ 0 s

```
import numpy as np  # Import NumPy
```

[30] ✓ 9 s

```
from sklearn.ensemble import RandomForestClassifier
# Setup random seed
np.random.seed(42)
clf = RandomForestClassifier(n_estimators = 50) #50 trees (estimators). The n_estimators parameter controls how many t
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
```

```
0.9999672388939851
```

[31] ✓ 0 s

```
y_preds = clf.predict(X_test)
y_preds
```

```
array([0, 1, 1, ..., 1, 0, 1])
```

[32] ✓ 0 s

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_recall_fscore_support
prec_rec = precision_recall_fscore_support(y_test, y_preds, average='weighted')
# print(f"Accuracy:{accuracy_score(y_test, y_preds):.4f}")
print(f"Accuracy:{accuracy_score(y_test, y_preds):.6f}")
# print(f"Precision:{prec_rec[0]:.4f} Recall:{prec_rec[1]:.4f} F1-score:{prec_rec[2]:.4f}")
print(f"Precision:{prec_rec[0]:.6f} Recall:{prec_rec[1]:.6f} F1-score:{prec_rec[2]:.6f}")
print(classification_report(y_test, y_preds, digits=6))
```

```
Accuracy:0.999967
Precision:0.999967 Recall:0.999967 F1-score:0.999967
              precision    recall  f1-score   support

           0   0.999932  1.000000  0.999966     14754
           1   1.000000  0.999937  0.999968     15770

    accuracy                       0.999967     30524
   macro avg   0.999966  0.999968  0.999967     30524
weighted avg   0.999967  0.999967  0.999967     30524
```

[33] ✓ 0 s

```
print(f"Accuracy:{accuracy_score(y_test, y_preds):.6f}")
```

```
Accuracy:0.999967
```

[34]
✓ 1 min

```python
# 1 min.
# Imporve model.
# Try different amount on n_estimators
import numpy as np
np.random.seed(42)
for i in range(10,100,10):
    print(f"Trying model with {i} estimators...")
    clf = RandomForestClassifier(n_estimators=i).fit(X_train, y_train)
    #print(f"Model accuracy on test set : {clf.score(X_test,y_test) * 100:.2f}%")
    print(f"Model accuracy on test set : {clf.score(X_test,y_test) * 100:.6f}%")
    print(" ")
```

```
Trying model with 10 estimators...
Model accuracy on test set : 99.990172%

Trying model with 20 estimators...
Model accuracy on test set : 99.993448%

Trying model with 30 estimators...
Model accuracy on test set : 99.996724%

Trying model with 40 estimators...
Model accuracy on test set : 100.000000%

Trying model with 50 estimators...
Model accuracy on test set : 99.993448%

Trying model with 60 estimators...
Model accuracy on test set : 99.996724%

Trying model with 70 estimators...
Model accuracy on test set : 99.996724%

Trying model with 80 estimators...
Model accuracy on test set : 99.996724%

Trying model with 90 estimators...
Model accuracy on test set : 99.993448%
```

[35]
✓ 1 min

```python
# 1 min.
from sklearn.model_selection import cross_val_score
cross_val_score = cross_val_score(clf, X, y)
np.mean(cross_val_score)
```

```
np.float64(0.9998427458324681)
```

AdaBoost classifier

[36] ✓ 15 s

```
# 20 sek.
#This code is following a conventional train-test split approach. It fits the model to
#a portion of the data (training set) and then assesses its performance on a separate, previously unseen
#portion of the data (test set).
from sklearn.ensemble import AdaBoostClassifier

# Setup random seed
np.random.seed(42)
Aclf = AdaBoostClassifier(n_estimators=100)
Aclf.fit(X_train,y_train)
Aclf.score(X_test,y_test)
```

```
0.9996396278338356
```

[37] ✓ 1 min

```
# 1 min.
#using cross-validation to obtain a more robust estimate of the model's
#performance by systematically testing it on different subsets of the data.
from sklearn.model_selection import cross_val_score
cross_val_score = cross_val_score(Aclf, X, y)
np.mean(cross_val_score)
```

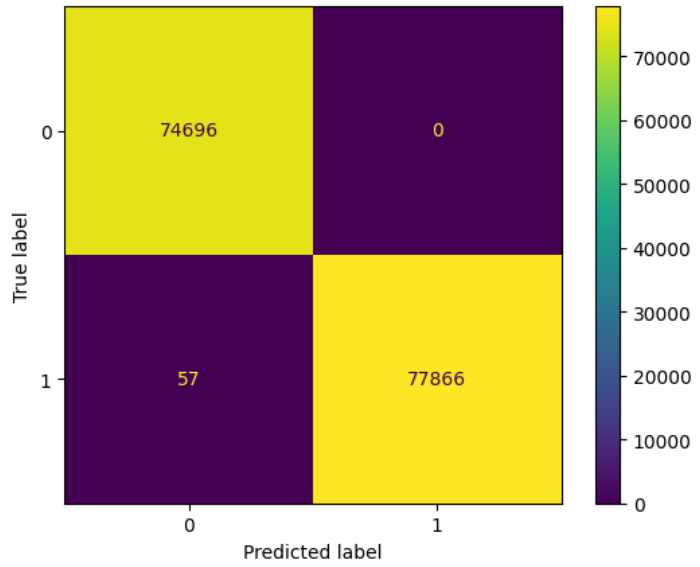
```
np.float64(0.9996265210301141)
```

[38] ✓ 1 s

```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(estimator=Aclf, X=X,y=y)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x781430cc32c0>
```



[39] ✓ 15 s

```
# 19 sek.
from sklearn.ensemble import GradientBoostingClassifier
Gclf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0).fit(X_train, y_train)
Gclf.score(X_test,y_test)
```

```
1.0
```

[40] ✓ 0 s

```
print(f"Model accuracy on test set : {Gclf.score(X_test,y_test) * 100:.6f}%")
```

```
Model accuracy on test set : 100.000000%
```

GaussianNB Classifier

Fitting the data into the model.

[52] ✓ 0 s

```
from sklearn.naive_bayes import GaussianNB
# Setup random seed
np.random.seed(42)
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB
GaussianNB()
```

Evaluating the model using `score()` method

[54] ✓ 0 s

```
print(f"Model accuracy on test set using score method : {gnb.score(X_test,y_test) * 100:.6f}%")
```

```
Model accuracy on test set using score method : 91.498493%
```

[55] ✓ 0 s

```
y_preds = gnb.predict(X_test)
y_preds
```

```
array([1, 1, 1, ..., 1, 0, 1])
```

Evlauating the model using Cross-validation method to avoid overfitting

[56] ✓ 0 s

```
from sklearn.model_selection import cross_val_score
cross_val_score = cross_val_score(gnb, X, y)
print(f"Model accuracy on test set using Cross-validation method : {np.mean(cross_val_score) * 100:.6f}%")
```

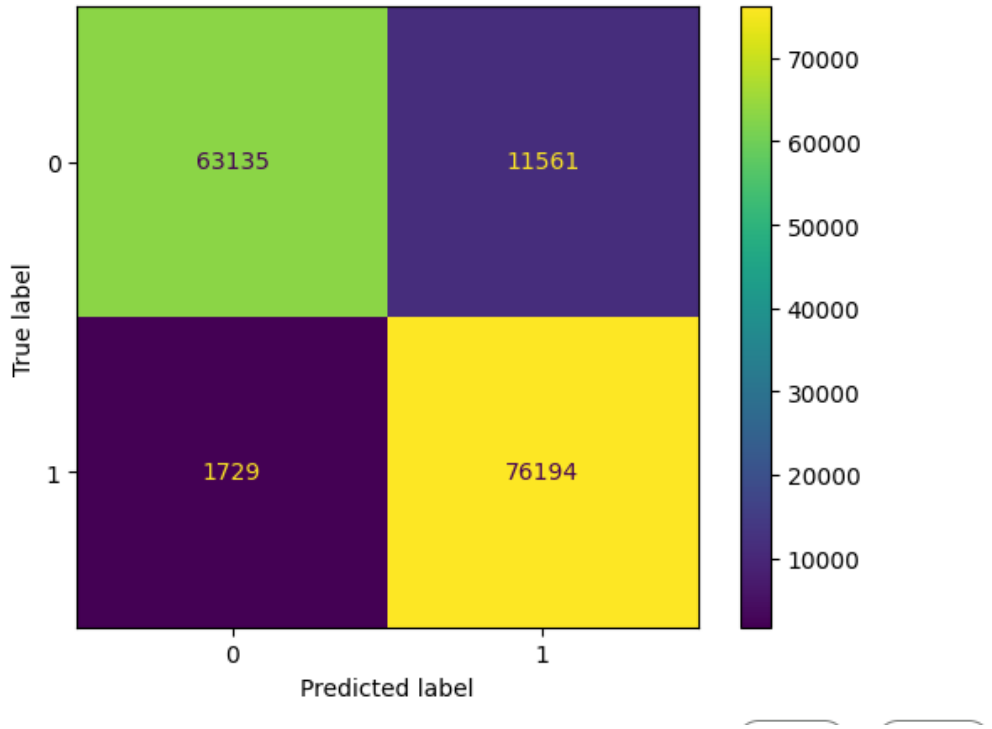
```
Model accuracy on test set using Cross-validation method : 90.108052%
```

Confusion matrix

[57]
✓ 0 s

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(estimator=gnb, X=X,y=y)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x781430570a40>
```



Model Evaluation: Area under the reciever operating characteristic curve (AUC/ROC)

- Area under curve (AUC)
- ROC curve
- ROC curves are a comparison of a model's true positive rate (tpr) versuss a models false positive rate (fpr).

- True possitve = model predicts 1 when truth is 1.
- False possitive = model predicts 1 when truth is 0.
- True negative = model predicts 0 when truth is 0.
- False negative = model predicts 0 when truth is 1

[58] ✓ 0 s

```
# Make predictions with probabilities
y_probs = gnb.predict_proba(X_test)
y_probs[:10]
```

```
array([[0.00846286, 0.99153714],
       [0.35646839, 0.64353161],
       [0.02987006, 0.97012994],
       [0.57359444, 0.42640556],
       [0.58218822, 0.41781178],
       [0.0158397 , 0.9841603 ],
       [0.00721634, 0.99278366],
       [0.30745915, 0.69254085],
       [0.02352071, 0.97647929],
       [0.77689046, 0.22310954]])
```

[59] ✓ 0 s

```
y_probs_positive = y_probs[:,1]
y_probs_positive[:10]
```

```
array([0.99153714, 0.64353161, 0.97012994, 0.42640556, 0.41781178,
       0.9841603 , 0.99278366, 0.69254085, 0.97647929, 0.22310954])
```

[60] ✓ 0 s

```
# Claculate fpr, tpr and thresholds.
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)

# Check the false positive rates
fpr
```

```
array([0.00000000e+00, 6.77782296e-05, 2.71112919e-04, ...,
       9.99051105e-01, 9.99186661e-01, 1.00000000e+00])
```

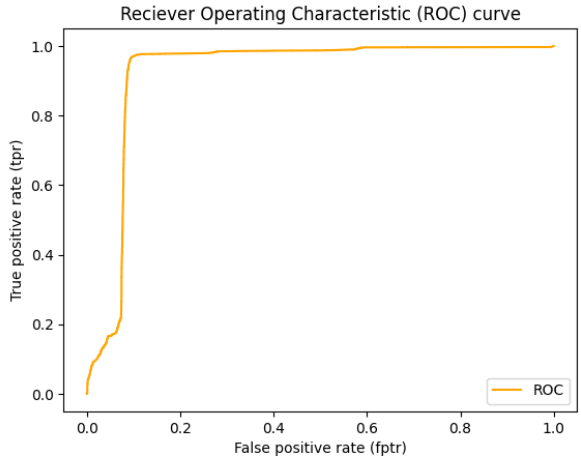
[61]
✓ 0 s

```
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):
    """
    Plots a ROC curve given the false positive rate (fpr)
    and true positive rate (tpr) of a model
    """
    # Plot ROC curve
    plt.plot(fpr, tpr, color = "orange", label="ROC")

    #Customize the plot.
    plt.xlabel("False positive rate (fptr)")
    plt.ylabel("True positive rate (tpr)")
    plt.title("Reciever Operating Characteristic (ROC) curve")
    plt.legend()
    plt.show()

plot_roc_curve(fpr, tpr)
```



[50]
✓ 0 s

```
from sklearn.metrics import roc_auc_score
print(f"Model accuracy on test set using reciver operating characteristic score method : {roc_auc_score(y_test, y_probs_positive) * 100:.6f
```

```
Model accuracy on test set using reciver operating characteristic score method : 92.183155%
```

Classification Report for **GaussianNB Classifier**

[62]
✓ 0 s

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_preds, digits=6))
```

```
              precision    recall  f1-score   support

           0   0.971974  0.848583  0.906097     14754
           1   0.873378  0.977108  0.922336     15770

    accuracy                       0.914985     30524
   macro avg   0.922676  0.912846  0.914216     30524
weighted avg   0.921035  0.914985  0.914487     30524
```

Deep learning (Feed forward neural network)

+ Kód + Text

[63]
✓ 6 s

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.12
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.76.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.15.1)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.18.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.11
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.10)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->t
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tens
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (4.0.0
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.1
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensor
```

[64] ✓ 18 s

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
```

[65] ✓ 17 s

```
pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

[66] ✓ 28 s

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
```

[67] ✓ 3 s

```
dataset_path = Trojan_Detection_csv
df = pd.read_csv(dataset_path, sep=',', skipinitialspace=True)
df.drop(columns=df.columns[0], axis=1, inplace=True)
```

[68] ✓ 0 s

```
df = df.dropna()
df = df.replace("Trojan", 1)
df = df.replace("Benign", 0)
```

```
/tmp/ipython-input-1200159886.py:3: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version.
  df = df.replace("Benign", 0)
```

[69] ✓ 0 s

```
from sklearn.preprocessing import LabelEncoder
number = LabelEncoder()
df["Flow ID"] = number.fit_transform(df["Flow ID"])
df["Source IP"] = number.fit_transform(df["Source IP"])
df["Destination IP"] = number.fit_transform(df["Destination IP"])
df["Timestamp"] = number.fit_transform(df["Timestamp"])
```

[70] ✓ 0 s

```
X = df.drop("Class", axis=1)
y = df["Class"]
```

[71] ✓ 0 s

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

[72] ✓ 0 s

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[73] ✓ 0 s

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer with sigmoid activation for binary classification
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[74] ✓ 0 s

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

[75] ✓ 55 s

```
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

```
Epoch 1/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8827 - loss: 0.3230 - val_accuracy: 0.9656 - val_loss: 0.1243
Epoch 2/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9663 - loss: 0.1177 - val_accuracy: 0.9740 - val_loss: 0.0933
Epoch 3/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9728 - loss: 0.0919 - val_accuracy: 0.9764 - val_loss: 0.0828
Epoch 4/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9755 - loss: 0.0805 - val_accuracy: 0.9776 - val_loss: 0.0723
Epoch 5/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9777 - loss: 0.0704 - val_accuracy: 0.9760 - val_loss: 0.0739
Epoch 6/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9792 - loss: 0.0636 - val_accuracy: 0.9815 - val_loss: 0.0584
Epoch 7/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9818 - loss: 0.0554 - val_accuracy: 0.9823 - val_loss: 0.0562
Epoch 8/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9831 - loss: 0.0528 - val_accuracy: 0.9817 - val_loss: 0.0565
Epoch 9/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9840 - loss: 0.0505 - val_accuracy: 0.9819 - val_loss: 0.0540
Epoch 10/10
1527/1527 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9843 - loss: 0.0478 - val_accuracy: 0.9834 - val_loss: 0.0496
<keras.src.callbacks.history.History at 0x7813bf6fb650>
```

[76] ✓ 1 s

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test loss: {loss:.6f}, Test accuracy: {accuracy:.6f}")
```

```
954/954 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9842 - loss: 0.0493
Test loss: 0.050193, Test accuracy: 0.983947
```

[77] ✓ 1 s

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

```
954/954 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```

[78] ✓ 0 s

```
# Print classification report
#print(classification_report(y_test, y_pred))
print(classification_report(y_test, y_pred, digits = 6))
```

```
                precision    recall  f1-score   support

           0   0.982201  0.985449  0.983822     15119
           1   0.985672  0.982473  0.984070     15405

    accuracy                       0.983947     30524
   macro avg   0.983936  0.983961  0.983946     30524
weighted avg   0.983953  0.983947  0.983947     30524
```

Zhrnutie

Na základe vykonaných experimentov možno zhodnotiť, že najvhodnejšie metódy pre detekciu malvéru typu „trojský kôň“ z prevádzky v sieti sú metódy

- Random Forest: úspešnosť 99,99%, čas vyhodnocovania ~9 sek.
- AdaBoost: úspešnosť 99,96%, čas vyhodnocovania ~19 sek.
- DL FFNN: úspešnosť 98,39%, čas vyhodnocovania ~15 sek.

Z nich Random Forest vykonala výpočty dvakrát rýchlejšie. Ďalšia v poradí Deep learning (Feed forward neural network) má ešte vysokú úspešnosť (98,39%) ale čas trénovania siete bol dlhší (55 sek.). Najrýchlejšou metódou bola GaussianNB Classifier – avšak dosiahla najhoršie výsledky (91.49%). Za najvhodnejšou metódou na detekciu malvéru typu „trojský kôň“ analyzovaním prevádzky považujeme algoritmus **Random Forest**.