FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Semi-supervised learning pro detekci malware
Student:	Bc. Michal Buchovecký
Vedoucí:	Mgr. Martin Jureček
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Antivirovým společnostem denně přichází velké množství nových neoznačených vzorků. Trénovací sada obsahuje relativně malé množství označených a velké množství neoznačených vzorků. Detekční systémy jsou obvykle založeny na supervised learning algoritmech pracujících s označenými vzorky. Cílem práce je vytvořit detekční systém, který dokáže využít dodatečnou informaci z neoznačených vzorků k zvýšení přesnosti detekcí.
Pokyny:
1) nastudovat semi-supervised learning algoritmy vhodné pro detekci malware
2) navrhnout nový, nebo modifikovat existující semi-supervised learning algoritmus
3) naimplementovat uvedené algoritmy a vyhodnotit jejich přesnosti detekcí
4) porovnat výsledky s výsledky supervised learning algoritmů

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 1. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Semi-supervised learning pre detekciu malwaru

Bc. Michal Buchovecký

Katedra informačnej bezpečnosti
Vedúci práce: Mgr. Martin Jureček

7. mája 2019

Poďakovanie

Chcel by som sa poďakovať všetkým, ktorí mi akokoľvek pomohli pri tvorbe tejto diplomovej práce, no predovšetkým vedúcemu práce, Mgr. Martinovi Jurečkovi, za vedenie práce a cenné pripomienky. Takisto by som sa chcel poďakovať svojej rodine za to, že mi umožnili študovať na tejto škole, a zároveň za ich podporu, bez ktorej by bolo štúdium o dosť zložitejšie.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 7. mája 2019 .....

Odkaz na túto prácu

Buchovecký, Michal . *Semi-supervised learning pre detekciu malwaru.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Využívanie strojového učenia v oblasti detekcie malwaru nie je v súčasnosti až tak veľmi populárne. Jedným z dôvodov je aj skutočnosť, že označovanie malwaru a legitímnych súborov, čo je pre strojové učenie nevyhnutné, je veľmi drahý proces. Táto práca sa zaoberá detekciou malwaru pomocou semi-supervised learningu. Tento typ učenia je jednou z kategórií strojového učenia, kedy k trénovaniu modelu využívame ako označené, tak aj neoznačené vzorky. K trénovaniu sme využívali informácie získane zo súborov v PE formáte. V tejto práci je ukázané, že využitím semi - supervised learningu je možné dosiahnuť lepšiu presnosť, než použitím len samotného supervised learningu.

Klíčová slova detekcia malwaru, semi - supervised learning, strojové učenie

Abstract

Nowadays, the use of machine learning for malware detection is not very popular. One of the reasons is that labelling of malware and benign files necessary for machine learning is very expensive process. This thesis is focused on malware detection by semi - supervised learning. Semi-supervised learning is a machine learning technique that makes use of labelled as well as unlabelled samples for training. Information obtained from executable files in PE format

was used for training. In the thesis it is showed that it is possible to reach better accuracy using semi - supervised learning, compared to purely supervised approach.

Keywords malware detection, semi - supervised learning, machine learning

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V dnešnej digitálnej dobe je veľkým trendom spracovávať a uchovávať akékoľvek informácie v digitálnej forme. Súčasné firmy nielen digitalizujú papierové informácie, ale vytvárajú aj biznis modely, ktoré sú založené na digitálnych aktívach. Ako príklad môžeme uviesť rôzne sociálne siete alebo aj v súčasnosti veľmi populárne streamovacie služby. S touto dobou sú však spojené aj hrozby, ktoré prichádzajú vo forme kybernetických útokov. Tie sú často spájané s využívaním malwaru, ktorého hlavným cieľom je obohatiť autora samotného malwaru, či už získaním cenných informácií, alebo napr. žiadaním výkupného od obete. Je preto veľmi dôležité včas tieto hrozby detegovať a vhodne na nich reagovať.

Do popredia sa čoraz viac dostáva strojové učenie alebo tiež machine learning, ktoré sa najčastejšie používa na rozpoznávanie, či detekciu rôznych objektov v rôznych odvetviach. Možno ani mnohí netušia, že sa s touto metódou stretávajú každý deň napr. pri čítaní emailov, kedy nám strojové učenie odfiltruje spam od relevantných emailov alebo na sociálnych sieťach, kde sú pre nás reklamy ušité na mieru práve na základe strojového učenia. Na to, aby sme boli schopný detegovať malware pomocou strojového učenia, nám postačuje dostatočne veľká dátová sada, v ktorej sa nachádzajú vzorky malwaru a legitímnych súborov, a sú istým spôsobom od seba odlíšené resp. označené. Následne potrebujeme zvoliť vhodný algoritmus na klasifikáciu, ktorý týmito vzorkami natrénujeme. Najväčším problémom však je získať označené vzorky malwaru a legitímnych súborov. Označovanie takejto dátovej sady si vyžaduje ľudí s potrebnou dávkou znalostí a zároveň aj veľa času, čo sa odzrkadľuje na tom, že je tento proces finančne náročný a celkovo nie veľmi efektívny.

Táto práca sa zaoberá algoritmami strojového učenia, ktorým na to, aby dosiahli pomerne vysokú presnosť detekcie, postačuje malá množina označených vzoriek v kombinácií s väčšou množinou neoznačených vzoriek, čím je možné ušetriť čas aj financie.

V prvej časti tejto práce sa budeme venovať malwaru. Povieme si o tom,

ako malware detegovať, analyzovať a uvedieme si aj jeho klasifikáciu. V druhej kapitole sa zoznámime so strojovým učením, kde sa dozvieme ako prebieha celý tento proces od získavania dát až po vyhodnotenie. Na túto kapitolu nadviaže kapitola tretia, kde sa oboznámime s niekoľkými najznámejšími semi-supervised algoritmami. Záverečná kapitola sa bude venovať implementácií niektorých zo spomenutých algoritmov, v ktorej si prejdeme celým procesom strojového učenia.

Kapitola 1

Malware

V súčasnosti je najväčšou hrozbou IT sveta škodlivý software. Tento software sa zvyčajne označuje výrazom malware, ktorý sa môže vyskytovať v rôznych podobách. V praxi ide o software, ktorý napadne počítač obete, zvyčajne bez jej vedomosti, a následne vykonáva neoprávnene operácie ako napr. poškodenie počítačového systému, krádež dát a pod. Takýto software k následnému šíreniu využíva rôzne zraniteľnosti, či už samotného operačného systému, konkrétneho softwaru, alebo aj sociálne inžinierstvo [1].

Podľa AV-TEST [2] je každý deň zaregistrovaného viac ako 350 000 nového malwaru. Za rok 2018 bolo zaregistrovaného cez 850 miliónov nového malwaru, čo je pri porovnaní s rokom 2017 zhruba 20% nárast.

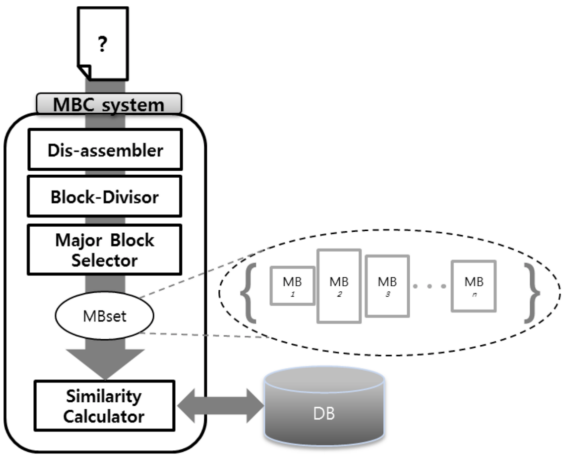
1.1 Detekcia malwaru

Za účelom ochrániť počítačový systém pred infikovaním škodlivým softwarom alebo odstrániť škodlivý software z napadnutého systému, je nutné vykonať korektnú detekciu malwaru. Detekcia malwaru je proces, kedy sa deteguje prítomnosť škodlivého softwaru na počítači resp. rozhoduje sa o tom, či konkrétny súbor je škodlivý alebo nie.

Na detekciu takýchto súborov sa najčastejšie využívajú nasledujúce metódy.

1.1.1 Detekcia signatúr

Najčastejší obranný mechanizmus proti malwaru je detekcia signatúr, ktorú bežne využívajú akékoľvek anti-malware softwary. Každý súbor má svoju špecifickú signatúru, ktorá je tvorená binárnou sekvenciou (niekedy sa za signatúru berie aj hash hodnota nejakého bloku kódu). Aby sa predišlo falošne pozitívnym detekciám, je nutné, aby signatúra bola vygenerovaná z dostatočne dlhej sekvencie kódu. Softwary detegujúce škodlivý software disponujú rozsiahlou databázou, ktorá obsahuje signatúry všetkých doposiaľ známych malwarov. V okamihu výskytu nového súboru v počítači, je tento súbor testovaný,



Obr. 11: Statická analýza - porovnávanie binárneho obsahu [5]

či obsahuje nejakú známu binárnu sekvenciu, resp. signatúru, a následne je vyhodnotený buď ako malware, alebo legitímny súbor. Nevýhodou je, že škodlivé súbory, ktorých signatúry sa nevyskytujú v databáze, sú ťažko detegovateľné. Autor malwaru sa môže detekcii ľahko vyhnúť tým, že pridá časti kódu, ktoré v skutočnosti nemajú žiadny význam, no dosiahne zmenu signatúry súboru [1, 3].

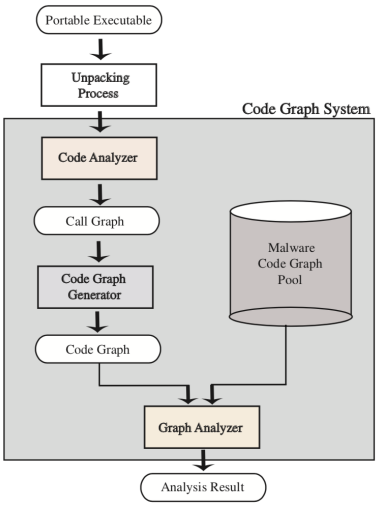
1.1.2 Statická analýza

Statická analýza potencionálne škodlivého súboru je v podstate využitie aproximácie určitej časti kódu, na základe ktorej sa určuje, či analyzovaný súbor môže vykonávať nejaké škodlivé činnosti [1].

V praxi existuje mnoho spôsobov statickej analýzy, uvedieme niekoľko najpoužívanejších z nich:

- **Porovnávanie binárneho obsahu**

  Analýza pozostáva zo 4 častí. Najprv je vstupný binárny súbor dissasemblerom preložený na sekvenciu inštrukcií. Táto sekvencia je rozdelená do blokov podľa funkcií. Následne tzv. major block selector prechádza všetkými blokmi a vyberá tie, ktoré sa budú analyzovať. Tieto bloky sú v poslednej fáze porovnávané Maďarským algoritmom [4] voči známym malwarovým vzorkám za účelom nájdenia podobnosti (viď obr. 11). Analyzovaním len vybraných blokov a nie celých binárnych súborov sa výrazne skracuje čas réžie, a tým pádom je samotná analýza výrazne rýchlejšia [5].



Obr. 12: Statická analýza - detekcia malwaru pomocou grafu kódu [9]

- **Výskyt reťazcov v linkovaných knižniciach a API volaniach**

  Kolter a kol. [6] testovali niekoľko machine learning algoritmov za účelom detegovať malware na základe vlastností vyextrahovaných z reťazcov, ktoré boli použité v dynamicky linkovaných DLL knižniciach a API volaniach. Zvolili 500 n-gramov (n-gram je definovaný ako súvislý sled n po sebe idúcich položiek z určitej sekvencie textu), kde pre n = 4 dostávali najlepšie výsledky. Tento model následne testovali pre niekoľko klasifikačných metód (Naive Bayes, SVM, rozhodovacie stromy, boosting). Víťazom sa stal posilnený rozhodovací strom s plochou 0.996 pod ROC krivkou.

- **Detekcia založená na ťažení informácií z API volaní**

  V roku 2010 Sami a kol. [7] využívali na detekciu malwaru informácie o API volaniach, ktoré získavali z IAT tabuliek súborov. V experimente použili Fisher Scoring algoritmus [8] ohodnocujúci API volania, ktoré sa najčastejšie vyskytujú v škodlivých súboroch. Podarilo sa im dosiahnuť až 99.7% úspešnosť detekcie.

- **Graf kódu**

  Lee a kol. [9] využili k detekcii malwaru grafy kódov. Analyzátor kódu pomocou transformačného algoritmu transformoval binárny súbor do orientovaného grafu API volaní, ktorý predstavuje vlastnosti skúmaného súboru. Porovnávanie grafu volaní je NP-úplný problém, preto je tento graf potrebné kvôli jednoduchšej analýze transformovať do grafu kódu. Grafový analyzátor nakoniec zmeria podobnosť dvoch kódov a vyhodnotí, či vstupný súbor je malware alebo nie (viď obr. 12).

- **Využitie informácií z PE formátu**

  Saini a kol. [10] predstavili rýchlu metódu ako odlíšiť malware od legitímnych súborov na základe informácií získaných z PE súborov. Zamerali sa na dve informácie, a to podozrivý počet sekcií a frekvencia volaní jednotlivých funkcií. Tieto dva príznaky použili na natrénovanie viacerých klasifikátorov. Výsledkom tohto experimentu bola viac ako 98% presnosť klasifikácie.

1.1.3 Dynamická analýza

Statická analýza súborov je často rýchly proces, ktorý dokáže odhaliť škodlivé súbory, no má svoje limity. Nie je príliš zložité sa voči nej brániť. Skúsení autori malwaru často používajú rôzne obfuskácie kódu alebo samorozbaľovacie archívy. Takéto použité techniky čiastočne zabraňujú použitiu statickej analýzy.

Dynamická analýza malwaru je založená na monitorovaní správania potenciálne škodlivého súboru počas jeho spustenia. Sleduje sa interakcia so systémom a skúmajú sa rôzne podozrivé činnosti. Takáto analýza najčastejšie prebieha na nejakom reálnom systéme, v sandboxe alebo virtuálnom počítači [1].

Podľa [11] sa využívajú dva prístupy, ktorých výsledky dosahujú rôznu granularitu a kvalitu:

- vytvoria sa obrazy celého systému pred a po spustení testovaného súboru, ktoré sa následne porovnávajú a analyzujú sa zmeny, ku ktorým počas behu daného súboru došlo
- činnosť testovaného súboru sa analyzuje priamo počas behu

Prvý spomenutý prístup je jednoduchší na realizáciu, no poskytuje menej detailné informácie. Nevieme totiž jednoznačne určiť, ako sa potencionálny malware správal počas behu, a aké akcie vykonával. Druhý spôsob je oproti prvému ťažšie realizovateľnejší, no prináša ďaleko detailnejšie informácie.

Sandbox

V počítačovej bezpečnosti sa tento pojem spája s prostredím, ktoré je oddelené a má obmedzený prístup k zdrojom hostiteľského systému s prísnou kontrolou, a právami. V takomto prostredí môžu nedôveryhodné aplikácie bežať bez toho, aby vzniklo nejaké riziko napadnutia, resp. infekcie hostiteľského systému [12].

Virtuálny počítač

Virtualizácia poskytuje prostredie, ktoré pracuje nezávislé vzhľadom k iným prostrediam v počítači resp. to, čo sa udeje v jednom prostredí nemôže nijak ovplyvniť alebo zmeniť iné prostredie. Rozdiel medzi virtuálnym počítačom a sandboxom je ten, že v prípade spustenia aplikácie v sandboxe je všetko,

čo sa aplikácia pokúša vytvoriť alebo zmeniť, po jej ukončení stratené. Naopak vo virtuálnom počítači sú všetky tieto zmeny zachované [12].

Analýza potencionálne škodlivých súborov vo virtuálnom prostredí je veľmi obľúbená metóda, nakoľko sa od spustenia na reálnom systéme takmer nelíši. Hlavnou hrozbou tejto analýzy je, že spustený súbor dokáže detegovať jeho spustenie vo virtuálnom prostredí, a teda dokáže svoje správanie zmeniť [13].

Existuje niekoľko spôsobov ako detegovať spustenie vo virtuálnom počítači, jeden z nich je tzv. Red Pill viď algoritmus 1.1.

Algoritmus 1.1: Red Pill - detekcia spustenia vo virtuálnom počítači [14]

```
1  int swallow_redpill () {
2    unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\
         xc3";
3     *((unsigned*)&rpill[3]) = (unsigned)m;
4          ((void(*)())&rpill)();
5          return (m[5]>0xd0) ? 1 : 0;
6        }
```

Pomocou algoritmu 1.1 dokážeme rozpoznať, či náš program beží vo virtuálnom prostredí alebo nie. Základom je inštrukcia SIDT kódovaná ako 0F010D, v ktorej je uložený obsah registru IDTR (adresa a veľkosť tabuľky vektorov prerušenia). Existuje však len jeden IDTR register, a keďže spúšťame minimálne dva operačné systémy (hostiteľský a virtuálny), virtuálny počítač musí tento register premiestniť na iné miesto. Pre príklad VMWare Workstation 4 premiestňuje IDT na adresu 0xFFXXXXXX [14].

1.1.4 Hybridná analýza

Okrem statickej a dynamickej analýzy môžeme použiť aj analýzu hybridnú, čo je akákoľvek kombinácia statickej a dynamickej analýzy. Keďže obe metódy majú svoje výhody a nevýhody, ich kombinácia nám môže pomôcť skúmaný súbor lepšie pochopiť.

1.2 Klasifikácia malwaru

Malware je veľmi široký pojem, ktorým sa označuje akýkoľvek škodlivý software. V minulosti bol takýto software vyvíjaný hlavne za účelom žartu, no postupne to prerástlo do stavu, kedy je hlavným cieľom získať od obete peniaze. V súčasnosti existuje mnoho rôznych typov malwaru, medzi základne skupiny radíme vírusy, červy a trójske kone.

1.2.1 Vírus

Snáď najčastejším výrazom, ktorý ľudia nesprávne používajú pre označenie škodlivého softwaru, je vírus. Ako však už bolo spomenuté, nie každý takýto software je vírus. Vírus je škodlivý software, ktorého hlavným cieľom je ukryť

sa v inom softwari bez toho, aby pozmenil jeho pôvodnú funkčnosť. Po spustení napadnutého softwaru sa spustí aj samotný vírus, ktorý sa snaží replikovať napádaním iného softwaru [15].

Vírusy sa ďalej delia na niekoľko kategórií podľa toho, akým spôsobom sa šíria, resp. skrývajú [15].

- **vírusy napádajúce súbory** - tento typ vírusu sa infikuje do spustiteľného súboru, ktorý sa následne spustí spolu s infikovaným súborom

- **vírusy napádajúce zavádzací sektor** - vírus infikuje zavádzací sektor disku. Ak je následne počítač spustený z tohto disku, vírus sa dostane do pamäte, v ktorej zostáva počas celej doby chodu počítača a môže ďalej infikovať iné disky, resp. vymeniteľné média, pomocou ktorých sa môže preniesť aj na iné počítače

- **multilaterálne vírusy** - vírusy, ktoré sa infikujú a šíria viacerými rôznymi spôsobmi. Kombinujú výhody vírusov, ktoré napádajú súbory a vírusov, ktoré napádajú zavádzací sektor. V praxi je takýto vírus veľmi účinný a dokáže sa veľmi rýchlo šíriť. Ich výskyt je však zriedkavý, pretože je ťažké ich navrhnúť a implementovať

- **makro vírusy** - ich výskyt je najčastejší v kancelárskych balíkoch ako napr. MS Office, v ktorých je možné vytvárať tzv. makrá slúžiace na automatizáciu nejakých krokov. Tieto makrá sa zvyčajne nainfikujú do prednastavenej šablóny, odkiaľ sa ďalej šíria do novovytvorených dokumentov

- **vírus operačného systému** - táto skupina vírusov napadá priamo súbory operačného systému a spúšťajú sa spolu so spustením napadnutého súboru. Je zároveň veľmi účinný, keďže súbory patriace priamo operačnému systému sa spúšťajú veľmi často

1.2.2 Červ

Červ je nezávislý program, ktorý sa replikuje a šíri pomocou internetu na iné počítače. K šíreniu využíva rôzne zraniteľnosti systému obete. Hlavným rozdielom medzi červom a vírusom je ten, že vírus potrebuje k prežitiu a šíreniu iné súbory na rozdiel od červu, ktorý je úplne nezávislý [15].

1.2.3 Trójsky kôň

Trójsky kôň alebo tiež aj Trojan je škodlivý software, ktorý sa v skutočnosti javí ako neškodný za účelom presvedčiť užívateľa, aby si ho nainštaloval. Tento pojem pochádza zo starovekého gréckeho príbehu o drevenom koňovi, ktorý

bol použitý ako pomoc gréckym vojskám dobiť mesto Trója. Na rozdiel od spomenutých vírusov a červov, hlavnou úlohou trójskych koňov nie je ich replikovanie, ale poškodzovanie napadnutého systému [15].

Podľa [16] bol jedným z najznámejších trójskych koňov Storm Worm, ktorý napádal operačné systémy spoločnosti Microsoft. Tento trójsky kôň začal útočiť prevažne v Európe a USA 19. januára 2007 tým, že rozosielal e-maily s predmetom „230 dead as storm batters Europe.“ Z napadnutých počítačov následne spravil boty, ktoré slúžili na šírenie tohto malwaru a rozposielanie veľkého množstva spamu. Podľa [17] pochádzal tento trójsky kôň z Ruska.

1.3 Subklasifikácia malwaru

Podľa funkčnosti malwaru je možné ho ďalej deliť do rôznych kategórií. Uvedieme tie, ktoré sú v praxi najčastejšie.

1.3.1 Adware

Slovo adware ja skratkou zo slov advertising-supported software. Tento typ malwaru je charakteristický neustálym zobrazovaním reklám. Často sa môžeme stretnúť aj so softwarom s bezplatnou licenciou, no spolu s ním si nainštalujeme aj nejaký adware, ktorý slúži na generovanie zisku. Väčšinou je tento typ malwaru šírený len za účelom zobrazovania reklám. Nie je však neobvyklé, že spolu s adwarom sa nainštaluje aj spyware, ktorý je pre užívateľa oveľa nebezpečnejší [18].

1.3.2 Backdoor

Backdoor je škodlivý software, ktorý umožňuje poskytnúť neoprávnený prístup do infikovaného systému. Tento typ malwaru je veľmi nebezpečný, najmä kvôli tomu, že útočník môže s daným systémom robiť takpovediac čo chce. Často sa využívajú tzv. keyloggery, snímanie obrazovky, krádež dát a pod. Backdoory sú často integrované už v nejakých programoch, poprípade na infikovanie využívajú zraniteľnosti systému alebo softwaru [19].

1.3.3 Bootkit

Bootkit je typ malwaru, ktorý útočí na tzv. Master Boot Record alebo Volume Boot Record. Týmto útokom je následne možné spustiť iný škodlivý software ešte pred spustením samotného operačného systému. S príchodom technológií ako UEFI a Secure Boot sa tento malware vyskytuje už len zriedkavo [20].

1.3.4 Coinminer

V súčasnej dobe sú veľkým fenoménom kryptomeny, a preto ani táto oblasť neunikla pozornosti medzi tvorcami malwaru. Coiminer je software, ktorý dokáže

jednotlivé meny ťažiť (využitím výpočtového výkonu), a tým pádom generovať zisk. Autori malwaru však infikujú coinminer do cudzieho počítača a dokážu tak využiť na ťažbu výpočtové zdroje obete. Medzi hlavné príznaky infikovania týmto malwarom patrí zvýšená činnosť CPU, resp. GPU, prehrievanie PC, zvýšená sieťová aktivita a celkové spomalenie systému [21].

1.3.5 Downloader

Downloader je typ malwaru, ktorého primárnou úlohou je stiahnuť do infikovaného počítača ďalší nežiadúci obsah z internetu. Môže ísť o rôzne súbory, nechcené aplikácie, ďalší malware, či nejaké príkazy [22].

1.3.6 Exploit

Exploit je typ programu, ktorý k napadnutiu systému využíva nejakú zraniteľnosť, či už samotného systému alebo konkrétneho softwaru za účelom získania práv k systému, spúšťania ďalšieho nežiaduceho kódu a pod [23].

1.3.7 Ransomware

Ransom malware, resp. ransomware, je typ malwaru, ktorý bráni obeti k prístupu jeho systému, resp. k osobným súborom a k opätovnému získaniu prístupu vyžaduje výkupné. Počiatky ransomwaru siahajú až do 80. rokov, kedy bolo výkupné potrebné zaslať poštou. V dnešnej dobe sa však preferujú kryptomeny, ktorých výhodou je, že sú anonymné [24].

Existuje niekoľko druhov ransomwaru [24]:

- **scareware** - tento druh ransomwaru neustále zobrazuje obeti tzv. popup správy o tom, že jeho systém bol infikovaný malwarom a na jeho odstránenie je potrebné zaplatiť
- **screen lockers** - ako už samotný názov napovedá, tento druh ransomwaru zablokuje počítač, čo je často sprevádzané zobrazením veľkého loga polície hovoriace o tom, že na našom PC bola zistená nelegálna činnosť a je potrebné zaplatiť pokutu
- **šifrujúci ransomware** - v súčasnosti najnebezpečnejší a najviac vyskytujúci sa druh ransomwaru, ktorý postupne zašifruje všetky naše súbory, a k opätovnému dešifrovaniu žiada výkupné

1.3.8 Rootkit

Hlavnou úlohou rootkitu je skrývať akýkoľvek škodlivý software pred detekčnými softwarmi. Rootkit zvyčajne skrýva prítomnosť malwaru skrývaním zložiek, v ktorých sa nachádzajú, skrývanie procesov, sieťovej aktivity a pod. Tento druh je spomedzi malwaru najťažšie detegovať a následne odstrániť.

V takýchto prípadoch sa zvyčajne odporúča vymazať celý disk a systém nainštalovať nanovo [25].

1.3.9 Spyware

Spyware sa snaží zhromažďovať resp. kradnúť informácie z počítača obete a odosielať ich bez jej vedomia útočníkovi. Rozlišujeme niekoľko typov spywaru [26]:

- **keylogger** - zaznamenáva postupnosť stlačených kláves na klávesnici, vďaka čomu môže útočník získať prihlasovacie údaje, emailovú komunikáciu atď.
- **spyware kradnúci heslá** - kradnú sa akékoľvek zadané heslá na infikovanom počítači, či už to sú systémové heslá, heslá do rôznych internetových služieb atď.
- **spyware kradnúci informácie** - zbierajú sa akékoľvek citlivé údaje ako prihlasovacie údaje, emailové adresy, história internetového prehliadača, osobné dokumenty a pod.
- **banking spyware** - tento typ malwaru útočí na internetové bankovníctva, digitálne peňaženky, a iné finančné portály, kde sa snažia získať prístup, pozmeňovať transakcie, vytvárať nové transakcie a pod.

1.3.10 Potentionally Unwanted Program

Špecifickým typom škodlivého softwaru sú tzv. potencionálne nechcené programy(skratka PUP). Môžeme sa stretnúť s praktikou, že tvorcovia softwaru vydávajú inštalačný balíček, kedy sa nám spolu s požadovaným softwarom nainštaluje aj ďalší, nechcený, v podobe rôzneho adwaru, toolbarov, v horšom prípade aj spyware. Druhou možnosťou sú tiež webové portály zamerané na sťahovanie softwaru s voľnou licenciou, kedy takisto pribalia k inštalácií nechcenú aplikáciu. Spoločnosti zaoberajúce sa detekciou malwaru označujú takéto aplikácie ako PUP. PUP sa od malwaru líši v tom, že si ho do počítača nainštalujeme viac-menej dobrovoľne [27].

Kapitola 2

Úvod do strojového učenia

Machine learning alebo tiež strojové učenie je vedná disciplína, zaoberajúca sa metódami, ktoré dokážu automaticky detegovať vzory z veľkého množstva dát za účelom predpovedať výsledok doteraz neznámych pozorovaní podľa predtým identifikovaných vzorov [28].

Podľa [29] pozostáva postup pri strojovom učení typicky z týchto krokov :

1. zber dát - tento krok je veľmi dôležitý, na kvalite dát závisí, ako spoľahlivý model budeme mať
2. príprava dát na spracovanie - normalizácia, čistenie dát atď.
3. výber vhodného modelu
4. trénovanie modelu pomocou trénovacej sady
5. testovanie modelu pomocou testovacej sady a vyhodnotenie
6. úprava parametrov modelu (v prípade, ak chceme zlepšiť model, vrátime sa na krok 4)
7. predikcia

2.1 Typy učení

Strojové učenie ďalej delíme do 4 kategórií:

- supervised learning (učenie s učiteľom)
- semi-supervised learning
- unsupervised learning (učenie bez učiteľa)
- reinforcement learning

2.1.1 Supervised learning

Cieľom tohto učenia je „naučiť“ algoritmus priraďovať pre vstup x správny výstup y. K tomu nám pomôže tzv. trénovacia sada $\mathcal{D}$, ktorá obsahuje dvojice vstup-výstup $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, kde N je počet týchto dvojíc. O takejto sade tiež hovoríme, že sa skladá z dvoch skupín tzv. features a labels. Features, resp. x_i, sú konkrétne hodnoty daných vlastností, ktoré chceme pozorovať, a na základe ktorých bude náš model predikovať výsledky (napr. výška a váha). Labels, teda y_i, sú správne výsledky, ktoré očakávame po aplikovaní funkcie f, teda $y_i = f(x_i)$. Tieto označenia sú potrebné, aby sa funkcia naučila predikovať výsledky [28].

Algoritmy supervised learningu sa ďalej delia na klasifikačné a regresné. Klasifikačné algoritmy sa používajú v situácii, kedy výstupom má byť zaradenie do nejakej kategórie napr. malware alebo legitímny súbor, kým regresné algoritmy sa používajú v situácií, kedy výstupom je nejaká reálna hodnota napr. odhad ceny nehnuteľnosti [30].

Overfitting a underfitting

Pri supervised learning algoritmoch sa stretávame s problémom tzv. overfitting, kedy sa model učí príliš detailne a snaží sa naučiť správne klasifikovať aj šum alebo náhodnú fluktuáciu. Tento problém negatívne ovplyvňuje schopnosť modelu zovšeobecňovať získané vedomosti, čím sa zhoršuje aj úspešnosť samotnej predikcie. Underfitting je zas opakom overfittingu. K underfittingu dochádza, keď model nie je schopný presne zachytiť vzťahy medzi skúmanými vlastnosťami a ich označeniami [31].

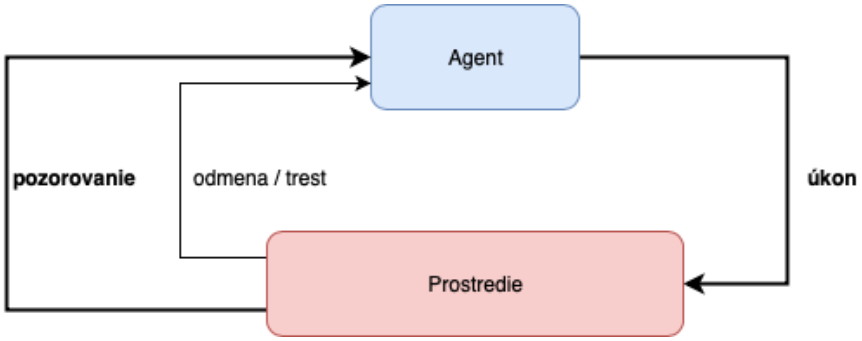
2.1.2 Semi-supervised learning

Semi-supervised learning je dosť podobný supervised learningu. Na rozdiel od supervised learningu sa k trénovaniu spolu s trénovaciou sadou $\mathcal{D}$ používa aj trénovacia sada $\mathcal{U} = \{(x_i)\}_{i=1}^{N}$, pri ktorej nemáme k dispozícií označenia. Tomuto typu učenia sa budeme detailnejšie venovať v kapitole 3.

2.1.3 Unsupervised learning

Unsupervised learning algoritmy sa učia z dát, ktoré nie sú nijak označené, či roztriedené. V porovnaní so supervised learningom, pri tomto učení nie je možné použiť algoritmy na klasifikáciu, či regresiu, nakoľko vôbec nepoznáme, čo by mohlo predstavovať výstupné hodnoty. Je teda na samotných algoritmoch preskúmať jednotlivé vlastnosti dát a nájsť medzi nimi nejaké vzťahy [28].

Algoritmy môžeme rozdeliť do dvoch skupín, a to zhlukovanie a asociácia. Zhlukovanie je problém, kedy sa jednotlivé prvky zoskupujú do zhlukov na základe spoločných vlastností. Asociačná analýza je problém, kedy hľadáme



Obr. 21: Agent pracuje v istom prostredí, v ktorom vykonáva rôzne úkony. Jednotlivé úkony sú analyzované a následne ohodnotené.

spoločné pravidlá, ktoré spájajú jednotlivé objekty v rámci veľkej skupiny dát [32]. Táto metóda sa často používa napr. v internetových obchodoch, kedy na základe nakúpených produktov nám sú ponúknuté iné produkty, ktoré by sa nám tiež mohli páčiť.

2.1.4 Reinforcement learning

Táto metóda nie je v oblasti strojového učenia veľmi bežná. Podobne ako aj pri unsupervised learningu nemáme k dispozícii žiadne správne výsledky. Modely tohto učenia sa teda učia výhradne pomocou spätnej väzby, kedy za jednotlivé činy dostanú buď odmenu alebo trest [28]. Na rozdiel od unsupervised learningu, kde sa model snaží nájsť vzťahy medzi jednotlivými vzorkami, tento model pomocou tzv. agenta hľadá postupnosť takých krokov, za ktoré získa čo najväčšiu odmenu. Tento typ učenia sa najčastejšie používa v hrách pri vytváraní AI, napr. v bojových hrách, kedy vojsko mení taktiku boja na základe krokov vykonaných hráčom. Základný princíp fungovania tohto učenia je zobrazený na obrázku 21.

2.2 Typy príznakov

V sfére strojového učenia sú dáta najčastejšie rozdeľované do dvoch kategórií, a to číselné a kategorické.

Číselné premenné predstavujú hodnoty, ktoré opisujú merateľné množstvo. Tento typ premennej ďalej rozdeľujeme na:

- **diskrétna premenná** - premenná, ktorej hodnota je celé číslo resp. dosahuje konečný počet, napr. počet položiek, ktoré si zákazník zakúpil
- **spojitá premenná** - premenná, ktorá môže dosahovať akúkoľvek hodnotu v rámci nejakého rozsahu, napr. čas, ktorý strávil zákazník v obchode.

Kategorické premenné predstavujú hodnoty, ktoré opisujú vlastnosti údajov napr. typ, kategóriu a pod. Tieto premenné sú vo vzájomne vylučujúcom sa vzťahu. Kategorické premenné môžeme ďalej deliť na:

- **ordinálna premenná** - premenné, ktorých hodnoty je možné zmysluplne zoradiť, napr. známky zo skúšky - A,B,C,D,E,F

- **nominálna premenná** - premenné, ktorých hodnoty nie je možné usporiadať, resp. všetky sú rovnocenné, napr. národnosti

2.2.1 Typy príznakov používané na detekciu malwaru

Pri detekcii malwaru pomocou strojového učenia je možné použiť viacero rôznych prístupov, čo sa týka typu príznakov, ktorými budeme trénovať náš model. Popíšeme si niekoľko z nich:

- **N-gram reťazce** - v spustiteľných súboroch sa reťazce vyskytujú v rôznych formách napr. URL odkazy, komentáre, importované knižnice atď. na základe, ktorých je možné identifikovať podozrivé súbory. N-gram je technika pri dolovaní dát z textu, kde n reprezentuje počet po sebe idúcich výrazov poprípade znakov. Túto metódu použil napr. Schultz a kol. [33] vďaka ktorej dosiahli presnosť 97,76%. Islam a kol. [34] použili počas výskumu techniku Printable String Information, ktorá je takisto založená na analýze reťazcov. Zo spustiteľného súboru sa získavajú názvy volajúcich funkcií a zároveň sa zaznamenáva počet ich výskytov. Touto technikou dosiahli až 98,86% presnosť detekcie.

- **N-gram postupnosti bajtov** - podobne ako na reťazce, sa n-gram technika dá aplikovať aj na postupnosť bajtov vyskytujúcich sa v preloženom zdrojovom kóde do strojového kódu.

- **PE hlavička** - PE formát je dátová štruktúra obsahujúca informácie pre operačný systém Windows, ktoré sú potrebné pre spustenie daného súboru. Niektoré informácie z tejto štruktúry sú takisto vhodné na detekciu malwaru. Tento typ príznakov využil pri svojej analýze aj Bai a kol. [35].

- **DLL knižnice** - obľúbenou technikou autormi malwaru je injektovať škodlivý kód do samotných DLL knižníc. PE štruktúra obsahuje zoznam všetkých potrebných DLL knižníc.

- **OpCode** - Santos a kol. [36] navrhli detekciu neznámeho malwaru na základe frekvencie výskytu postupností operačných kódov.

2.3 Škálovanie príznakov

Často sa nám môže stať, že získaný dataset, s ktorým chceme pracovať, obsahuje vlastnosti, ktoré majú veľmi rozdielne hodnoty, rozsah a pod. Pri následnom použití algoritmov strojového učenia, ktoré využívajú Euklidovskú metriku na získanie vzdialenosti dvoch bodov (napr. K - najbližších susedov alebo SVM), je potrebné tieto dáta škálovať tak, aby všetky mali rovnakú váhu významnosti. V opačnom prípade môže dôjsť k tomu, že zvolený algoritmus strojového učenia nebude správne reagovať na jednotlivé rozdiely v hodnotách vlastností. Ďalšou výhodou škálovaných vlastností je aj rýchlejší trénovací proces. Najčastejšie sa používajú nasledovné typy škálovaní.

Normalizácia

Normalizácia, tiež známa aj ako min-max normalizácia, sa používa v prípade potreby naškálovať hodnoty na interval, ktorý je typicky $[0, 1]$. Škálovanie prebieha podľa nasledovného vzťahu:

$$x_{new} = \frac{x - min(x)}{max(x) - min(x)} \tag{2.1}$$

kde x_{new} je normalizovaná hodnota, x je pôvodná hodnota, $max(x)$ a $min(x)$ je maximálna resp. minimálna hodnota zo súboru danej vlastnosti.

Štandardizácia

Štandardizácia umožňuje, aby hodnoty danej vlastnosti mali nulovú strednú hodnotu a rozptyl rovný 1. Štandardizáciu definujeme nasledovne:

$$x_{new} = \frac{x - \bar{x}}{\sigma} \tag{2.2}$$

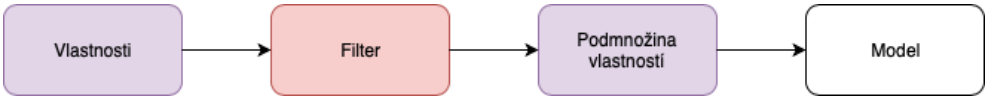
kde $\bar{x}$ je priemer hodnôt danej vlastnosti, a σ smerodajná odchýlka.

2.4 Výber príznakov

Súčasné datasety, s ktorými sa stretávame, zvyčajne obsahujú veľké množstvo rôznych vlastností. Niektoré z nich však môžu byť pre náš pripravovaný model úplne zbytočné, a tým pádom ich nie je vhodné pridávať do finálnej trénovacej sady. V takomto prípade je potrebné vybrať také vlastnosti, ktoré nám pomôžu zlepšiť presnosť modelu a zároveň odstrániť vlastnosti, ktoré sú irelevantné alebo redundantné. Tento proces nazývame aj tzv. feature selection.

Feature selection je jednou z najdôležitejších častí pri vytváraní modelu. Podľa [37] má tento proces tri hlavné funkcie:

1. zlepšenie úspešnosti predikcie



Obr. 22: Princíp fungovania filtračných metód

2. rýchlejšia a cenovo efektívnejšia predikcia

3. pochopenie procesu, ktorý generoval dáta

Výber vhodných vlastností však nie je jednoduchý problém. Ako je uvedené v [38], predpokladajme, že chceme vybrať podmnožinu S vlastností o veľkosti n z pôvodnej množiny m. Takýchto kombinácií existuje:

$$S = \frac{m!}{(m-n)!n!} \tag{2.3}$$

Ak by sme napríklad chceli vybrať 10 vlastností z 25, museli by sme posudzovať až 3 268 760 podmnožín, čo by bolo časovo náročne. Existujú však metódy, ktoré dokážu tento problém riešiť efektívnejšie. Tieto metódy delíme podľa viacerých kritérií. Podľa [38] existujú dve základne stratégie na výber vlastností:

1. **optimálne metódy:** metódy, v ktorých sa vyhľadáva hrubou silou, môžu viesť k optimálnemu riešeniu, no sú výpočtovo náročné, a preto sú vhodné len pre veľmi malé problémy

2. **suboptimálne metódy:** metódy, v ktorých ide o kompromis medzi rýchlosťou výpočtu a optimality

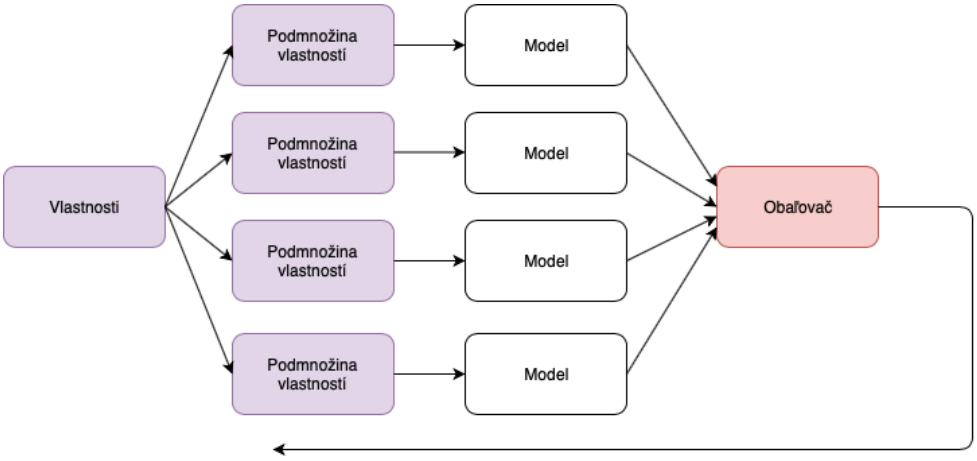
Algoritmy na výber vlastností môžeme ďalej podľa [39] deliť do štyroch kategórií: filtračné, obaľovacie (wrapper), embedded a hybridné.

2.4.1 Filtračné metódy

Filtračné metódy vyberajú vlastnosti bez ohľadu na to, aký bude použitý model strojového učenia. Trénovací proces teda začína až po zvolení najlepšej podmnožiny vlastností (viď obr. 22). Tento postup zabezpečuje vysokú výpočtovú efektivitu a je zároveň odolný voči tzv. overfittingu. Tieto metódy rozdeľujeme na jednorozmerné (univariate) a viacrozmerné (multivariate).

Jednorozmerné metódy

Jednorozmerné metódy posudzujú každú jednu vlastnosť individuálne bez toho, aby sa brali do úvahy väzby s ostatnými vlastnosťami. Podľa [38] je



Obr. 23: Princíp fungovania obaľovacích metód

jednou z najjednoduchších metód patriacich do tejto kategórie metóda Best individual N, ktorá vyberá N najlepších vlastností. Funkcia J priraďuje každej vlastnosti x_1 až x_p hodnotu kvality. Tieto vlastnosti sú následne zoradené:

$$J(x_1) \geq J(x_2) ... \geq J(x_p) \tag{2.4}$$

Po zoradení vyberieme podmnožinu vlastností s najvyššími hodnotami o veľkosti N

$$\{x_i \mid i \leq N\} \tag{2.5}$$

Viacrozmerné metódy

Multirozmerné metódy posudzujú podmnožiny vlastností naraz a snažia sa medzi nimi nájsť nejaké väzby. Tento problém je však veľmi zložitý, pre n vlastností, existuje až 2^n možných podmnožín.

2.4.2 Obaľovacie metódy

Obaľovacie (wrapper) metódy sú iteratívne metódy, ktoré v každej iterácii ohodnocujú niekoľko množín vlastností, pomocou ktorých sú natrénované modely. Wrapper teda ohodnotí kvalitu každej množiny, a množina vďaka ktorej bol model najkvalitnejší, postupuje do ďalšej iterácie, kde sa znovu bude posudzovať jeho kvalita spolu s novými množinami (viď obr. 23). Tento typ metód je oproti filtračným oveľa pomalší. Filtračné metódy však môžu pri hľadaní najlepšej podmnožiny zlyhať, no obaľovacie metódy nám vždy poskytnú najlepšiu množinu príznakov [40]. Obaľovacie metódy delíme na deterministické a randomizované.

Deterministické obaľovače

Do tejto kategórie patrí aj algoritmus Sequential Forward Selection (SFS) [38]. Tento algoritmus pracuje na princípe bottom-up, čo znamená, že algoritmus začína s prázdnou množinou X, do ktorej sa postupne pridávajú vlastnosti x_1 až x_p, až kým sa nedosiahne maximálny povolený počet vlastností. Algoritmus pracuje nasledovne: pra každú vlastnosť funkcia J spočíta hodnotu kvality ako $J(X+x_i)$ za predpokladu, že x_i sa zatiaľ v množine X nenachádza. Vlastnosť, pre ktorú bude hodnota funkcie J najvyššia bude pridaná do množiny X a proces sa zopakuje. Hlavnou nevýhodou je, že algoritmus nevie pridané vlastnosti z množiny odobrať. Opakom tohto algoritmu je Sequential Backward Selection (SBS) [38], ktorý pracuje na opačnom princípe top-down. V tomto prípade algoritmus začína s naplnenou množinou X, z ktorej je v každom kroku odobratá jedna vlastnosť x_i taká, ktorá hodnotu funkcie $J(X - x_i)$ znižuje najmenej. Tento proces sa opakuje, kým nezostane v množine X požadovaný počet vlastností.

Randomizované obaľovače

Tieto obaľovače väčšinou využívajú genetické algoritmy alebo simulované žíhanie. Jedným z takých algoritmov je Best Incremental Ranked Subset. Tento algoritmus vyhodnocuje gény na základe ich hodnoty a označenia kategórie. Následne sa na identifikáciu redundantných génov aplikuje funkcia incremental ranked usefulness [41].

2.4.3 Embedded metódy

Embedded metódy majú v porovnaní s obaľovacími metódami menšiu výpočtovú náročnosť, no ich nevýhodou je, že sú závisle na výbere klasifikátora. To znamená, že ak pri hľadaní najlepších vlastností bol použitý nejaký klasifikátor, neznamená to, že výsledná množina vlastností bude najlepšia možná aj pre iný klasifikátor. Jednou z najznámejších metód tohto typu sú napr. náhodné lesy.

2.4.4 Hybridné metódy

Hybridné metódy boli navrhnuté za účelom skombinovať najlepšie vlastnosti filtračných a obaľovacích metód. Filtračné metódy redukujú dimenzionálny priestor a získavajú tak niekoľko potencionálnych kandidátov. Následne sa použijú obaľovacie metódy, ktoré nájdu najlepšiu podmnožinu vlastností. Vďaka zlúčením týchto dvoch metód dosahujú hybridné metódy vysokú presnosť a efektivitu. Príkladom sú napr. fuzzy náhodné lesy.

2.5 Klasifikátory strojového učenia

Počas samotnej klasifikácie malwaru pomocou semi-supervised learningu budeme v nasledujúcich častiach využívať niekoľko rôznych klasifikátorov. V súčasnosti existuje mnoho algoritmov, ktoré riešia klasifikačný problém, no je ťažko určiť, ktoré sú lepšie, a ktoré horšie. Všetko závisí od toho, aký problém riešime, od dát, ktoré máme k dispozícií, a či ich vieme lineárne rozdeliť atď. Klasifikátory sa zvyknú rozdeľovať na „lenivé“ a „usilovné“.

Lenivé klasifikátory si počas fázy trénovania len ukladajú dáta bez akýchkoľvek ďalších operácií. Až v momente, kedy majú k dispozícií testovacie dáta, začnú na základe podobnosti vzoriek proces klasifikácie. Tieto klasifikátory sú rýchle vo fáze trénovania, no o dosť pomalšie počas predikcie. Patrí sem napr. model K - najbližších susedov. Usilovné klasifikátory sú zas klasifikátory, ktoré si zostavujú celý klasifikačný algoritmus už počas trénovania. Oproti lenivým klasifikátorom sú pomalšie v trénovacej fáze, no o dosť rýchlejšie počas predikcie. Medzi usilovné klasifikátory radíme napr. rozhodovací strom alebo neurónové siete.

V stručnosti si spomenieme niekoľko základných klasifikačných algoritmov, s niektorými z nich budeme pracovať aj v ďalších častiach:

- **Rozhodovací strom** - jeden z najstarších klasifikátorov, ktorý rozdeľuje dátovú sadu na menšie podmnožiny tým, že v každom uzle sa testuje nejaká vybraná vlastnosť. Tento algoritmus takto delí dátovú sadu až do doby, kým všetky vlastnosti nepatria do jednej kategórie. Na výber najvhodnejšej vlastnosti na základe ktorej sa bude strom vetviť sa používa buď informačný zisk (information gain), alebo Giniho koeficient.

- **Support Vector Machine** - tento algoritmus vytvára nadrovinu, ktorá oddeľuje dve rôzne kategórie. Optimálna nadrovina je taká, ktorá rozdeľuje dve odlišné kategórie v čo najväčšom odstupe od krajných bodov jednotlivých kategórií. Tento algoritmus má okrem lineárnej aj nelineárnu alternatívu (Kernel SVM), ktorá sa používa v prípade, ak nie je možné oddeliť kategórie lineárne.

- **Náhodný les** - je súborom rozhodovacích stromov, ktorých počet si vopred určíme. O výsledku sa v závere algoritmu hlasuje.

- **K - najbližších susedov** - algoritmus, ktorý hľadá K najbližších susedov z trénovacich vzoriek k vzorke testovacej. Na základe toho, koľko susedov z ktorej kategórie je najbližšie k vzorke, ktorú vyhodnocujeme, sa vypočíta pravdepodobnosť a podľa majority sa určí kategória, do ktorej patrí.

Stretnúť sa však môžme s mnohými ďalšími ako je Naive Bayes, ktorý je založený na Bayesovej vete, veľmi známe neurónové siete alebo s rôznymi

		Skutočné hodnoty	
		Malware	Legitímne
Predikované hodnoty	Malware	TP	FP
	Legitímne	FN	TN

Tabuľka 21: Konfúzna matica

kombináciami jednotlivých klasifikátorov. Takéto kombinácie (napr. Boosting, Bootstrap agregating) spájajú niekoľko rôznych klasifikátorov za účelom získať presnejší výsledok. Získané výsledky sa priemerujú, poprípade sa za ne hlasuje, a tým sa získava finálny výsledok. Pre podrobnejšie informácie o týchto klasifikátoroch odporúčame nahliadnuť do [28].

2.6 Vyhodnotenie modelu

Po zbere dát sme potrebovali dáta vhodne pripraviť pre ďalšie fázy strojového učenia, čo spočíva najmä v ich škálovaní a vo výbere tých najrelevantnejších vlastností, vďaka ktorým získame čo najviac spoľahlivý model. Po trénovacej fáze modelu prichádza na rad jeho vyhodnotenie. V tejto časti si ukážeme, ako zistiť, resp. vyjadriť kvalitu vytvoreného modelu.

2.6.1 Konfúzna matica

Konfúzna matica sa používa na zobrazenie chýb, ktoré vznikli počas procesu klasifikácie. Hodnoty na hlavnej diagonálne matice určujú počet správne klasifikovaných položiek, kdežto ostatné hodnoty zas určujú počet nesprávne klasifikovaných prípadov. Nasledujúca matica a metriky sú ukázané na modele predikujúcom malware, čím sa zaoberá aj táto práca.

Táto matica (viď tabuľka 21) obsahuje 4 položky:

- True Positive - správne klasifikovaný malware ako malware
- True Negative - správne klasifikovaný legitímny súbor ako legitímny
- False Positive - nesprávne klasifikovaný legitímny súbor ako malware
- False Negative - nesprávne klasifikovaný malware ako legitímny súbor

Z tejto matice vieme následne vypočítať ďalšie metriky, na základe ktorých môžeme vyjadriť presnosť modelu [42]:

- False Positive Rate - vyjadruje pomer nesprávne klasifikovaných legitímnych súborov ako malware voči všetkým legitímnym súborom

$$FPR = \frac{FP}{FP + TN} \tag{2.6}$$

- False Negative Rate - vyjadruje pomer nesprávne klasifikovaného malwaru ako legitímny súbor voči všetkým malware súborom

$$FNR = \frac{FN}{FN + TP} \tag{2.7}$$

- Precision - určuje pomer správne klasifikovaného malwaru voči všetkým súborom klasifikovaným ako malware, či už správne alebo nie

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

- Recall - tiež označovaný ako True Positive Rate, vyjadruje pomer správne klasifikovaného malwaru voči celkovému počtu malwaru

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

- Presnosť (accuracy) - metrika, ktorá určuje všeobecnú presnosť modelu, vyjadruje pomer správne klasifikovaných súborov voči všetkým súborom

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.10}$$

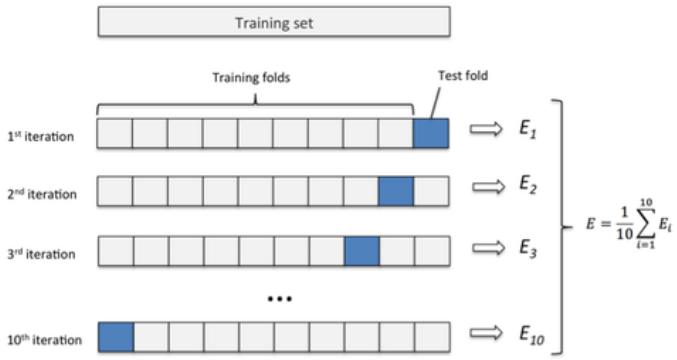
- F1 skóre - metrika, takisto určujúca celkovú presnosť modelu, ktorá je váženým priemerom hodnôt Precision a Recall

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.11}$$

Ukázali sme si niekoľko základných vzťahov, na základe ktorých vieme následne vypočítať všeobecnú spoľahlivosť modelu. Otázkou však zostáva, ktorá je pre nás z vyššie uvedených vhodnejšia. Podľa [43] je skóre F1 zvyčajne užitočnejšie, než hodnota ACC. Accuracy je spoľahlivá metrika, ak máme zhruba rovnaký počet falošne pozitívnych a falošne negatívnych klasifikácií. V prípade, ak je medzi počtom týchto misklasifikácií veľký rozdiel, mali by sme brať do úvahy metriku F1.

2.6.2 Krížová validácia

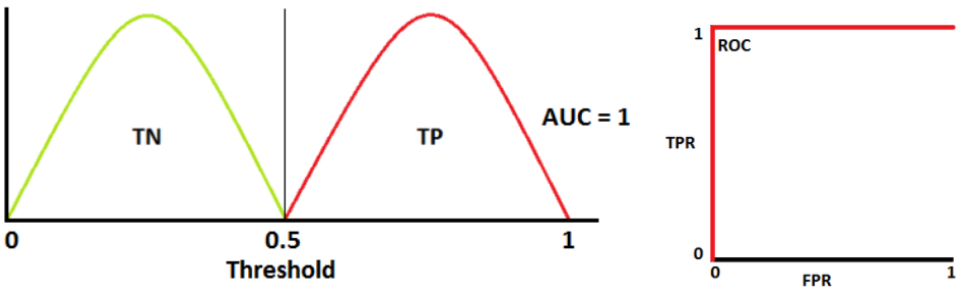
Pri trénovaní modelov môže občas dochádzať k overfittingu, o ktorom sme si povedali v 2.1.1. Na overenie toho, či sa tento problém týka aj nášho natrénovaného modelu môžeme použiť práve k-fold krížovú validáciu. Táto metóda rozdeľuje dátovú sadu na k vzájomne sa vylučujúcich podmnožín rovnakej veľkosti s tým, že jedna podmnožina je testovacia a zvyšných $k - 1$ sa použije na trénovanie. Tento proces sa opakuje k - krát, a jeho princíp je ilustrovaný na obrázku 24.



Obr. 24: K-fold krížová validácia[1]

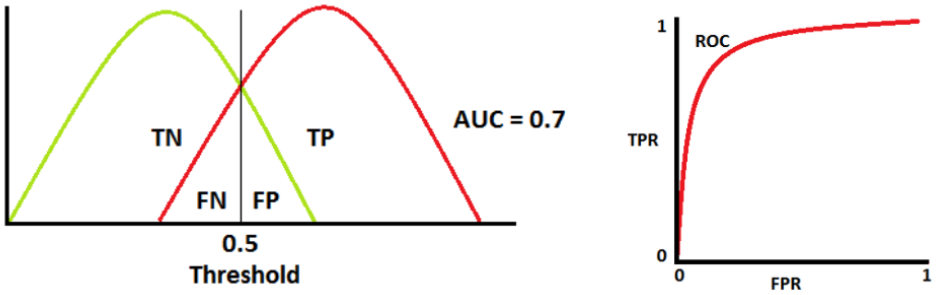
2.6.3 ROC krivka

Ďalšou často používanou metrikou pri hodnotení modelu je Receiver Operating Characteristic (ROC) krivka, ktorá sa používa pre hodnotenie binárneho klasifikátora. Táto krivka vyjadruje vzťah medzi TPR a FPR. K tejto metrike sa viaže aj ďalší dôležitý pojem, a tým je Area Under the Curve (AUC), ktorý vyjadruje obsah plochy pod ROC krivkou. Čím väčšia táto plocha je, tým lepší model máme. Ak je AUC hodnota rovná 1, máme ideálny klasifikátor (viď obr. 25), ak je hodnota rovná 0,5, náš klasifikátor je na úrovni náhodného hádzania mincou (viď obr. 27). ROC krivka priemernej hodnoty AUC tj. 0,7 je znázornená na obrázku 26. Hodnota AUC je teda pravdepodobnosť s akou bude náš model schopný správne klasifikovať danú položku.

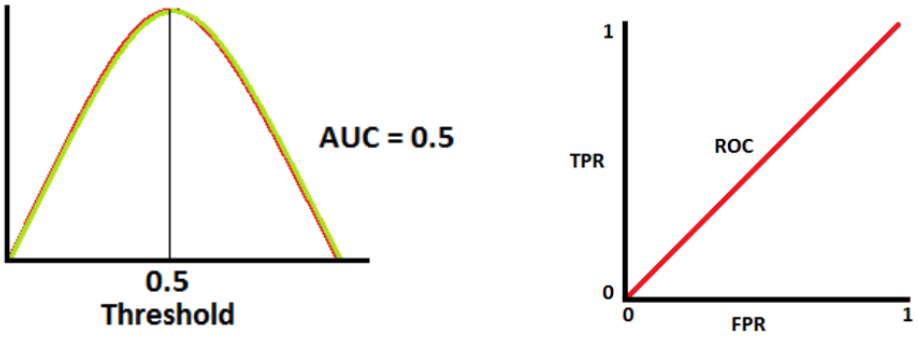


Obr. 25: ROC krivka - ideálny prípad, kedy klasifikátor bez chýb klasifikuje medzi dvoma prípadmi[2]

[1] https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623
[2] https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5



Obr. 26: ROC krivka - prípad, kedy už dochádza k prekryvu, a teda pri klasifikácií vznikajú chyby[2]



Obr. 27: ROC krivka - model, ktorý popisuje táto krivka je nepoužiteľný, keďže pravdepodobnosť správnej klasifikácie je 0,5[2]

[2] https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Kapitola 3

Semi-supervised Learning

V časti 2.1.1 sme si uviedli, že pri supervised learningu je potrebné mať k dispozícií dostatočne veľké množstvo dát, ktoré sú zároveň označené. Označenia k jednotlivým dátam je často zložité získať, nakoľko je to časovo a finančne náročný proces. Na rozdiel od označených dát sa k neoznačeným dátam vieme dostať oveľa jednoduchšie a lacnejšie. Semi-supervised learning je teda proces, kedy sa k učeniu využíva veľké množstvo neoznačených dát spolu s označenými dátami. Neoznačené dáta nám pomáhajú zvyšovať presnosť modelu zatiaľ čo nám klesá finančná a časová náročnosť. Tento typ učenia sa často používa pri klasifikácií webových stránok, hlasovom rozpoznávaní, či sekvenovaní DNA [44].

Rovnako ako pri supervised learningu, aj tu sa algoritmy delia na klasifikačné a regresné.

3.1 Self – Training

Self–training je jeden z najčastejšie používaných semi-supervised learning algoritmov. V tomto učení sa klasifikátor najprv učí z menšieho množstva označených dát. Po tomto procese sa klasifikujú neoznačené dáta a tie, ktoré majú vysokú pravdepodobnosť správnej predikcie sú spolu s predikovanými označeniami presunuté do trénovacej sady. Klasifikátor je následne znovu trénovaný z rozšírenej trénovacej sady a celý algoritmus sa znova opakuje. Tento algoritmus je možné aplikovať na akýkoľvek model supervised learningu [44]. Pseudokód tohto algoritmu je naznačený v algoritme 1

Pri tomto procese môže dôjsť k situácií, kedy model chybne označí neoznačené dáta, čo spôsobí nepresnosti pri následnej predikcii. Ming Li a kol. navrhli algoritmus s názvom SETRED [45], ktorý využíva špecifickú metódu editácie dát na určenie a odstránenie nesprávne označených dát. V každej iterácií self-training procesu sa teda štatisticky vyhodnotí, či novo označený údaj je dôveryhodný alebo nie. Do tréningovej sady sa potom pridávajú len dôveryhodné údaje.

Algoritmus 1 Pseudokód algoritmu Self - Training

VSTUP: L - sada označených dát, U - sada neoznačených dát
VÝSTUP: Natrénovaný model pomocou označených a neoznačených dát

1: M = natrénuj supervised learning model s L
2: **while** $U \neq \emptyset$ **do**
3: Aplikuj M na neoznačenú sadu dát U
4: Vyber N vzoriek s najväčšou pravdepodobnosťou správnej klasifikácie
5: Pridaj tieto vzorky s predpovedaným označením do L
6: Odober tieto vzorky z U
7: Pretrénuj model M s novou sadou dát L
8: **end while**

3.2 Co - Training

Blum a Mitchell [46] v roku 1998 navrhli nový model učenia, ktorý predpokladá, že jednotlivé vlastnosti je možné rozdeliť do dvoch množín, kde:

- každá množina vlastností je postačujúca na natrénovanie klasifikátora
- obe množiny sú navzájom nezávislé

Tento model teda vyžaduje dva rôzne pohľady na dáta s tým, že obe tieto vlastnosti poskytujú rozdielne a doplňujúce informácie o danom objekte, sú podmienene nezávislé, a zároveň každá z týchto dvoch vlastností je postačujúca na to, aby sme dokázali vytvoriť spoľahlivý klasifikátor. V počiatočnej fáze algoritmu najprv trénujeme nezávisle oba klasifikátory, každý klasifikátor na jednej z dvoch množín vlastností. Následne každý klasifikátor klasifikuje neoznačené dáta a učí druhý klasifikátor na dátach, ktoré majú vysokú pravdepodobnosť správnej predikcie. Tieto dáta odoberieme z množiny neoznačených dát a proces opakujeme. Pseudokód algoritmu Co - Training nájdeme v algoritme 2.

Takéto modely sa vo všeobecnosti označujú aj ako multi-view learning algoritmy. Vyššie spomenutý co-training a napr. aj tri-training patria práve do tejto skupiny. Základnou myšlienkou je teda používanie viacerých pohľadov na dáta.

3.3 Tri - Training

Zhou and Li navrhli tri-training [47], ktorý využíva tri klasifikátory. Podľa [47] je algoritmus nasledovný. V algoritme sa využívajú tri klasifikátory, ktoré sú trénované na náhodných podmnožinách označených dátových sád. Následne je každý z nich iteratívne znovu natrénovaný s novými, predtým neoznačenými vzorkami, o ktorých sa na klasifikácií zhodli dva zvyšné klasifikátory. Nesprávnou predikciou dvoch klasifikátorov dochádza k vytváraniu šumu, čo

Algoritmus 2 Pseudokód algoritmu Co - Training

VSTUP: L - sada označených dát, U - sada neoznačených dát
VÝSTUP: Natrénované dva modely pomocou označených a neoznačených dát

1: Rozdeľ L do dvoch označených sád L_1 a L_2
2: M_1 = natrénuj supervised learning model s L_1
3: M_2 = natrénuj supervised learning model s L_2
4: **while** $U \neq \emptyset$ **do**
5: Aplikuj M_1 a M_2 na neoznačenú sadu dát U
6: Vyber N vzoriek s najväčšou pravdepodobnosťou správnej klasifikácie modelu M_1 a M_2
7: Pridaj najviac pravdepodobné vzorky predpovedané M_1 do sady L_2
8: Pridaj najviac pravdepodobné vzorky predpovedané M_2 do sady L_1
9: Odober tieto vzorky z U
10: Pretrénuj model M_1 s L_1 a M_2 s L_2
11: **end while**
12: Pre následnú predikciu využi priemer alebo hlasovanie

následne znižuje presnosť klasifikátorov. Aby sa takejto situácií predišlo, zavádza sa obmedzenie tzv. hornou hranicou šumu. Klasifikátor bude pretrénovaný len ak bude platiť nasledujúci vzťah:

$$0 < \frac{e^t}{e^{t-1}} < \frac{|L^{t-1}|}{|L^t|} < 1 \tag{3.1}$$

kde $|L^t|$ je počet vzoriek, ktoré môžu byť pridané v t-tej iterácií (na predikovanej kategórii sa musia zhodnúť dva klasifikátory) a e^t je horná hranica chybovosti novoooznačených vzoriek L^t v t-tej iterácií. Ak je $|L^t|$ príliš veľké na to, aby spĺňalo vzťah 3.1, počet vzoriek môže byť redukovaný tak, že sa vyberie náhodná podmnožina, ktorá bude obsahovať s vzoriek, kde

$$s = \lceil \frac{e^{t-1}|L^{t-1}|}{e^t} - 1 \rceil \tag{3.2}$$

Algoritmus ďalej pokračuje do doby, kým aspoň jeden z troch klasifikátorov dokáže splniť vzťah 3.1. Pseudokód tohto algoritmu je naznačený v algoritme 3.

3.4 Algoritmy založené na teórii grafov

Semi-supervised algoritmy založené na teórii grafov definujú graf nad celou dátovou sadou $\mathcal{D} = L \cup U$, kde $L = \{(x_1, y_1), ..., (x_l, y_l)\}$ je množina označených dát, a $U = \{x_{l+1}, ..., x_{l+u}\}$ je množina neoznačených dát. Vektor príznakov $x_i \in \mathbb{R}^m, i = 1, ..., l + u$ je k dispozícií pre všetky označené aj

Algoritmus 3 Pseudokód algoritmu Tri - Training

VSTUP: L - označená sada dát, U - neoznačená sada dát
VÝSTUP: Natrénované tri modely

1: Rozdeľ L do troch označených sád L_1, L_2 a L_3
2: M_1 = natrénuj supervised learning model s L_1
3: M_2 = natrénuj supervised learning model s L_2
4: M_3 = natrénuj supervised learning model s L_3
5: **for all** M_i **do**
6: Z dátovej sady U vyber vzorky L_a s predikovanými označeniami, s ktorými súhlasia dva klasifikátory
7: Ak je splnená rovnica 2.1, pretrénuj M_i s $L_i \cup L_a$
8: Ak žiadny z klasifikátorov už nie je možné pretrénovať **break**
9: **end for**
10: Pre následnú predikciu využi hlasovanie

neoznačené dáta a $y_i \in \mathbb{R}^k, i = 1, ..., l$, kde k je počet kategórií sú označenia, resp. kategórie, do ktorých patria jednotlivé vzorky označených dát. Uzly budú následne predstavovať označené a neoznačené vzorky z dátovej sady a hrany zas predstavujú istú podobnosť medzi týmito vzorkami. Ak sú dve vzorky prepojené „silnou“ hranou, ich označenie by malo byť rovnaké. Čím je teda väčšia podobnosť medzi uzlami, tým je väčšia pravdepodobnosť rovnakého označenia. Informácie o označení sa propagujú z uzlov označených vzoriek na uzly neoznačených vzoriek.

Blum a Chawla [48] predstavili algoritmus založený na probléme minimálneho rezu. V binárnom prípade, kladné označenia predstavujú zdroje a záporné označenia zas ústia. Cieľom je nájsť minimálnu množinu hrán, ktorých odstránením dôjde k prerušeniu všetkých tokov medzi zdrojom a ústím. Uzly spájajúce zdroje sú teda označené kladne a tie, ktoré spájajú ústia zas negatívne.

3.5 Learning with Local and Global Consistency

Zhou a kol. [49] navrhli LLGC semi-supervised learning algoritmus, ktorý umožňuje klasifikáciu s ohľadom na vnútornú štruktúru známych označených a neoznačených bodov. Algoritmus je iteratívny a konštruuje koherentnú funkciu s dvomi predpokladmi:

- susedné body majú rovnaké označenia
- body rovnakej štruktúry (zhluk, varieta) majú rovnaké označenie

Ukážeme si algoritmus tak, ako sa uvádza aj v [49]. Daná je množina $\mathcal{X} = \{x_1, ..., x_l, x_{l+1}, ..., x_n\} \subset \mathbb{R}^m$ a množina označení $\mathcal{L} = \{1, ..., c\}$. Prvých l bodov $x_i (i \leq l)$ je označených ako $y_i \in \mathcal{L}$ a zvyšné body $x_u (l + 1 \leq u \leq n)$ sú

neoznačené body. Označme ďalej $\mathcal{F}$ ako množinu $n \times c$ matíc s nezápornými hodnotami. Matica $F = [F_1^T, ...F_n^T]^T \in \mathcal{F}$ zodpovedá klasifikácii dátovej sady $\mathcal{X}$ označením každého bodu x_i značkou $y_i = \arg\max_{j \leq c} F_{i,j}$. F môžeme chápať ako vektorovú funkciu $F : \mathcal{X} \to \mathbb{R}$, ktorá priradí vektor F_i, každému bodu x_i. Y je $n \times c$ matica, $Y \in \mathcal{F}$, ak x_i je označený ako $y_i = j$, potom $Y_{ij} = 1$ inak $Y_{ij} = 0$. Samotný algoritmus pozostáva z týchto krokov:

1. Forma afinnej matice W je definovaná ako $W_{ij} = \exp(\frac{-\|x_i - x_j\|^2}{2\sigma^2})$ ak $i \neq j$ a $W_{ii} = 0$.

2. Skonštruuj maticu $S = D^{-1/2} \cdot W \cdot D^{-1/2}$, kde D je diagonálna matica s prvkami (i, i), ktoré sú rovné sume i-tého riadku matice W.

3. Iteruj $F(t+1) = \alpha \cdot S \cdot F(t) + (1 - \alpha) \cdot Y$ kým nekonverguje, pričom α je parameter z intervalu $(0, 1)$.

4. F^* označuje limitu sekvencie $\{F(t)\}$. Označ každý bod x_i so značkou $y_i = \arg\max_{j \leq c} F^*_{ij}$.

Tento algoritmus najprv definuje párový vzťah W na dátovej sade $\mathcal{X}$ s diagonálnymi prvkami, ktoré sú rovné 0. Predpokladajme, že graf $G = (V, E)$ je definovaný na dátovej sade $\mathcal{X}$, kde množina uzlov V je rovná $\mathcal{X}$ a množina hrán E je váha hodnôt matice W. V ďalšom kroku algoritmus symetricky normalizuje maticu W. Tento krok je nutný kvôli zabezpečeniu konvergencie každej iterácie. Počas každej iterácie v treťom kroku získa každý bod informácie od svojho suseda a zároveň si zachováva svoju pôvodnú informáciu. Parameter α špecifikuje relatívne množstvo informácií od suseda a jeho počiatočnú informáciu. Informácia je rozširovaná symetricky vďaka tomu, že S je symetrická matica. Na záver algoritmus určí kategóriu každého neoznačeného bodu na základe toho, koľko informácií dostal o jednotlivých kategóriach počas procesu iterácie.

Kapitola 4

Portable Executable format

Portable Executable formát je formát, ktorý využívajú spustiteľné súbory, objektové súbory a DLL knižnice v operačnom systéme Windows. Tento formát je dátovou štruktúrou, ktorá obsahuje informácie potrebné pre spustenie požadovaného súboru. Súbory uložené v tomto formáte sú prenositeľné medzi všetkými 32 a 64 bitovými verziami Windowsu, na čo poukazuje aj výraz portable v samotnom názve. Štruktúra PE formátu je rozdelená do niekoľkých častí, ktoré môžme vidieť na obrázku 41. Z niektorých častí vieme získať mnoho zaujímavých informácií o danom súbore, ktoré môžeme následne použiť k analýze, či daný súbor je škodlivý alebo nie. V nasledujúcich sekciách si popíšeme jednotlivé časti štruktúry PE formátu.

4.1 DOS MZ Header

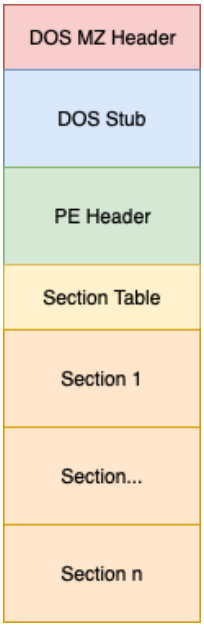
DOS MZ Header je zachovanou časťou z čias operačného systému MS-DOS. Táto časť je v tejto štruktúre z jediného dôvodu, a to pre prípad, ak by sme daný súbor chceli spustiť pod operačným systémom MS DOS. Tento systém samozrejme nedokáže spustiť súčasné používané súbory, no dokáže ho prečítať, a upozorniť nás, že požadovaný súbor nie je možné spustiť v DOS móde.

4.2 DOS Stub

DOS Stub informuje užívateľa o tom, že spúšťaný súbor potrebuje pre spustenie operačný systém Windows v prípade, ak by sme spúšťali súbor na systéme, ktorý nepozná PE formát.

4.3 PE Header

Ak spúšťame súbor v PE formáte na operačnom systéme, ktorý vie s týmto formátom pracovať, prvé dve časti sa automaticky preskakujú a pokračuje sa



Obr. 41: Štruktúra PE formátu

priamo na adresu, kde začína samotný PE header. Táto hlavička obsahuje štruktúru IMAGE_NT_HEADERS, ktorá obsahuje ďalej adresy na štruktúry IMAGE_FILE_HEADER a IMAGE_OPTIONAL_HEADER.

Image File Header

V tejto štruktúre môžeme nájsť rôzne informácie o tom, pre akú architektúru bol daný súbor skompilovaný (x86, x64), počet sekcií, časovú pečiatku, kedy bol súbor vytvorený, atď.

Image Optional Header

Hneď za File Header sa nachádza Optional Header. Tu nájdeme informácie o tom, ako má systém s daným súborom zaobchádzať. Dozvieme sa tu napr. verziu aplikácie, entry point (adresa, kde začína prvá inštrukcia programu), koľko miesta na zásobníku má systém pre súbor rezervovať, minimálnu požadovanú verziu operačného systému atď.

V tejto hlavičke sa ďalej nachádza pole s dátovými adresármi, kde každá položka obsahuje adresu a jeho veľkosť. Odtiaľ sa vieme prepracovať až k importnej tabuľke adries (IAT), odkiaľ vieme zistiť názvy importovaných knižníc a použitých funkcií. Tieto poznatky nám môžu pomôcť pri následnej analýze súboru.

4.4 Section table

Táto tabuľka je pole štruktúr, kde každá jedna štruktúra obsahuje informácie o jednej sekcii. Každá sekcia je rozdelená na hlavičku a telo. Tieto sekcie obsahujú samotný obsah súboru, teda zdrojový kód v časti .text, dáta v častiach .bss (napr. premenné deklarované ako static), .rdata (read-only data napr. konštanty) a .data, a zdrojové informácie v časti .rsrc.

Kapitola 5

Realizácia

V tejto kapitole si postupne rozoberieme jednotlivé kroky, ktorými sme postupovali od získavania dát až po samotné vyhodnotenie natrénovaných modelov.

5.1 Nástroje

Pred samotným spracovaním dát je potrebné spomenúť aj najdôležitejšie nástroje, ktoré sme počas celého postupu používali.

Anaconda

Anaconda[1] je open-source distribúcia Pythonu a programovacieho jazyka R, ktorá je určená hlavne pre vedecké výpočty, dátovú vedu, štatistické analýzy, strojové učenie a pod.

Scikit-learn

Scikit-learn[2] je machine learning knižnica určená pre Python. Obsahuje rôzne nástroje, ktoré sú počas celej fázy vytvárania modelu potrebné t.j. algoritmy na výber vhodných príznakov, klasifikačné, regresné a zhlukovacie algoritmy, nástroje na vyhodnotenie modelu, atď.

Spyder

Spyder[3] je open-source IDE určené pre Python navrhnuté pre vedcov, inžinierov a analytikov. Ponúka pokročilé funkcie na úpravu, analýzu a ladenie vývoja. Spyder umožňuje spúšťať kód po jednotlivých častiach, čo je v našej situácií veľmi užitočné. Toto prostredie sme použili na spracovanie dát, trénovanie a samotné vyhodnotenie výsledku.

[1] https://www.anaconda.com
[2] https://scikit-learn.org/stable/
[3] https://github.com/spyder-ide/spyder

5.2 Dátova sada

Podľa [50] si detekcia malwaru pomocou strojového učenia zatiaľ nezískala takú popularitu ako iné odvetvia napr. rozpoznávanie písaného textu, rozpoznávanie hlasu, rozpoznávanie dopravných značiek, k čomu je možné získať kvalitné dátové sady, čo sa žiaľ o dátovej sade k detekcii malwaru povedať nedá. Dôvodov je hneď niekoľko:

- **Licenčné obmedzenia** - k binárnym súborom malwaru sa vieme dopracovať napr. na stránkach ako VirusShare[4] alebo VX Heaven[5], no oveľa väčší problém je s legitímnymi súbormi. Tie zvyčajne podliehajú licenčným obmedzeniam, a nie je možné ich zdieľať.

- **Označenia** - na rozdiel od označovania napr. obrázkov, alebo hlasu, ktoré je pomerne rýchle a môže ho vykonávať aj laik, označiť konkrétny súbor ako malware alebo legitímny je o dosť náročnejší proces, ku ktorému potrebujeme mnoho znalostí a oveľa viac času.

- **Bezpečnostné riziká** - zverejnenie rozsiahlej dátovej sady, ktorá obsahuje množstvo malwaru, môže pre neskúsených užívateľov predstavovať bezpečnostné riziko.

Tento problém rieši dátova sada s názvom Malware BEnchmark for Research (EMBER)[6]. Táto sada obsahuje zbierku tzv. JSON Lines (označuje sa aj ako JSONL) súborov, kde každý riadok obsahuje jeden JSON objekt. Každý objekt obsahuje nasledujúce informácie:

- sha256 hash pôvodného súboru, ktorý zároveň slúži ako unikátny identifikátor

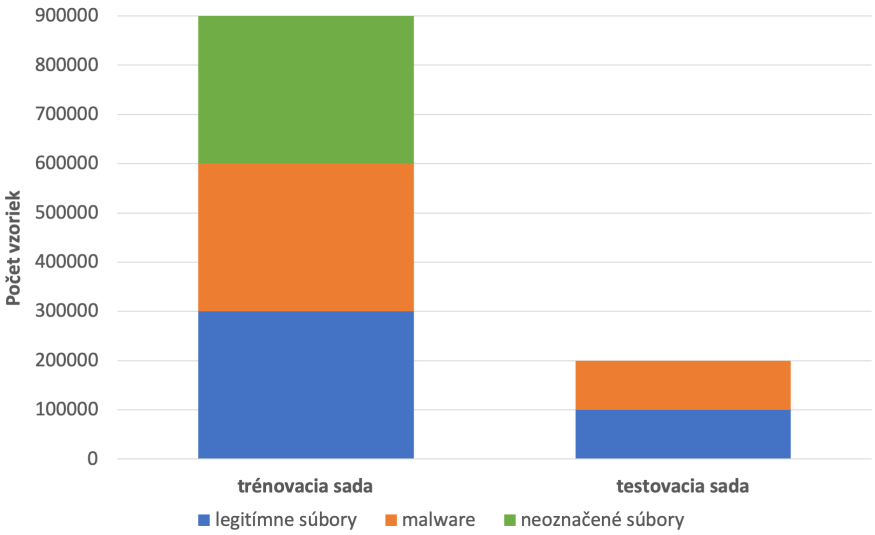
- časová značka, ktorá informuje o tom, kedy bol súbor prvýkrát objavený

- označenie súboru, 1 označuje malware, 0 legitímny súbor a -1 predstavuje neoznačený súbor

- 8 skupín príznakov, ktoré obsahujú hodnoty rozparsované z PE formátu

Rozdelenie dát, resp. počet legitímnych súborov a malwaru zobrazuje graf 51. K tejto dátovej sade je okrem iného pribalená aj knižnica, ktorá obsahuje funkcie na prácu s touto sadou. Pre nás bude užitočná funkcia create_vectorized_features, ktorá vektorizuje príznaky, resp. spracuváva získané dáta do takej podoby, aby nimi bolo možné trénovať jednotlivé modely strojového učenia.

[4] https://virusshare.com
[5] http://83.133.184.251/virensimulation.org/
[6] https://github.com/endgameinc/ember



Obr. 51: Rozdelenie dát v rámci dátovej sady EMBER

5.2.1 Príznaky

V tejto časti si popíšeme príznaky, ktoré obsahujú jednotlivé vzorky. Tieto príznaky sú reprezentované numerickými hodnotami, ale aj reťazcami. Aby sme mohli pri trénovaní modelu používať aj príznaky reprezentované reťazcami, je nutné ich upraviť do vhodnej podoby. Vďaka knižnici, ktorá je dostupná spolu s touto sadou, je tento problém vyriešený pomocou tzv. hashing triku [51]. Príznaky jednotlivých vzoriek sú rozdelené do dvoch kategórií:

Príznaky extrahované z PE súboru

- **Všeobecné informácie** - množina príznakov obsahujúca základné informácie ako virtuálna veľkosť súboru, počet importovaných a exportovaných funkcií, informácia, či súbor obsahuje ladiacu sekciu, thread local storage a počet symbolov.
- **Hlavičkové informácie** - z PE hlavičky sú vyexportované údaje ako časová pečiatka vzniku súboru, cieľová platforma (reťazec) a image characteristics (zoznam reťazcov). Z časti optional header máme k dispozícií informácie o cieľovom subsystéme (reťazec), DLL characteristics (zoznam reťazcov), magic (reťazec), major a minor image verzie, verzia linkeru, systému a subsystému, a na záver veľkosti samotného kódu a hlavičky.
- **Importované funkcie** - názvy importovaných funkcií, ktoré sú získané z importnej tabuľky adries.
- **Exportované funkcie** - zoznam exportovaných funkcií.

- **Informácie o sekciách** - vlastnosti každej sekcie vrátane názvu, veľkosti, entropie, virtuálnej veľkosti a zoznam reťazcov reprezentujúcich charakteristiku sekcií.

Format - agnostic príznaky

Tieto príznaky nie sú vyextrahované priamo z PE formátu, ale sú získané na základe iných metód, ktoré sú uvedené v [52].

- **Histogram bajtov** - histogram obsahuje 256 číselných hodnôt, ktoré reprezentujú počet výskytov každého bajtu v rámci daného súboru.
- **Histogram entropie bajtov** - histogram entropie bajtov aproximuje úplne združené rozdelenie $F(H, X)$, kde H je entropia a X je hodnota bajtu. Vytvorenie tohto histogramu je bližšie popísane v [52].
- **Informácie o reťazcoch** - štatistické informácie o reťazcoch, ktoré majú dĺžku aspoň 5 znakov. Patrí tu počet takýchto reťazcov, ich priemerná dĺžka, histogram znakov v rámci týchto reťazcov a entropia naprieč všetkými reťazcami.

5.3 Výber príznakov

Po vektorizácii surových dát získame maticu príznakov o veľkosti 900000×2351 v prípade trénovacej sady, resp. 200000×2351 pre testovaciu sadu. Tieto sady môžu obsahovať mnoho irelevantných dát, ktoré nám zbytočne predlžujú čas, ktorý je potrebný na natrénovanie modelu a takisto môže podľa [53] dochádzať k overfittingu, ktorý sme si už spomenuli. Aby sme predišli týmto problémom, je vhodné pred samotným trénovaním vybrať z množiny príznakov tie, ktoré sú najrelevantnejšie.

Knižnica *scikit-learn* ponúka niekoľko metód na výber príznakov. My sme na porovnanie výsledkov a pre správny výber najlepších príznakov použili dve z nich.

5.3.1 SelectKBest

Ide o jednorozmernú metódu, ktorú sme si spomenuli v 2.4.1. Táto metóda ohodnotí všetky príznaky v závislosti na zvolenej funkcii a podľa ohodnotenia vyberie K príznakov s najlepším ohodnotením, pričom K si vopred volíme. Pre ohodnotenie sme zvolili funkciu vzájomnej informácie (mutual information)[54].

Vzájomná informácia medzi dvomi náhodnými premennými je nezáporná hodnota, ktorá meria ich vzájomnú závislosť. Ak je hodnota rovná nule, takéto dve premenné označíme ako nezávisle. V opačnom prípade čím vyššia je táto hodnota, tým je medzi premennými väčšia závislosť.

5.3.2 SelectFromModel

Táto metóda môže byť použitá pri akomkoľvek modely z knižnice *scikit-learn*, ktorý má k dispozícií atribút coef_ alebo feature_importances_. Po natrénovaní zvoleného modelu sú príznaky, ktoré boli označené ako nedôležité odstránené. Takéto rozhodnutie je učinené na základe hodnoty coef_ resp. feature_importances, ktorá je nižšia ako zvolený prahový parameter. Na určenie prahu je možné využiť aj heuristiku na nájdenie toho najoptimálnejšieho. K dispozícii sú tri typy: priemer, medián alebo násobky ako napr $0,1\times$ priemer.

5.3.3 Vyhodnotenie

Pre testovanie uvedených metód sme z pôvodnej dátovej sady vybrali podmnožinu o veľkosti 120 000 vzoriek s tým, že počet legitímnych súborov v sade bol 60 000 a počet malwaru takisto 60 000.

Pri metóde SelectKBest sme za K volili postupne hodnoty od 5 do 100. Na následne testovanie dosiahnutej presnosti sme použili model rozhodovací strom.

Pri metóde SelectFromModel sme použili podľa dokumentácie jeden z odporúčaných modelov ExtraTreesClassifier, ktorý trénuje naraz N zvolených rozhodovacích stromov na náhodne vybratých podmnožinách. Na predikciu následne využíva priemer z týchto stromov. Počet stromov N sme volili postupne od 10 do 50.

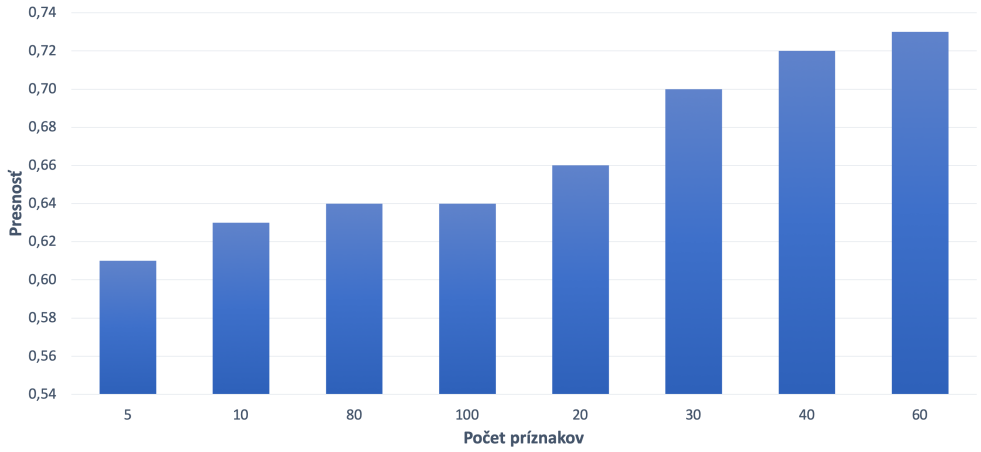
Na grafe 52 sú zobrazené dosiahnuté výsledky metódy SelectKBest. Môžeme si všimnúť, že najvyššie presnosti sme dosiahli pri použití 30 až 60 príznakov. Pri vyššom počte príznakov už presnosť modelu klesala.

Na grafe 53 sú zas zobrazené dosiahnuté výsledky pri použití metódy SelectFromModel. V porovnaní s metódou SelectKBest je potrebných o mnoho viac príznakov, pritom dosiahnutá presnosť bola značne nižšia.

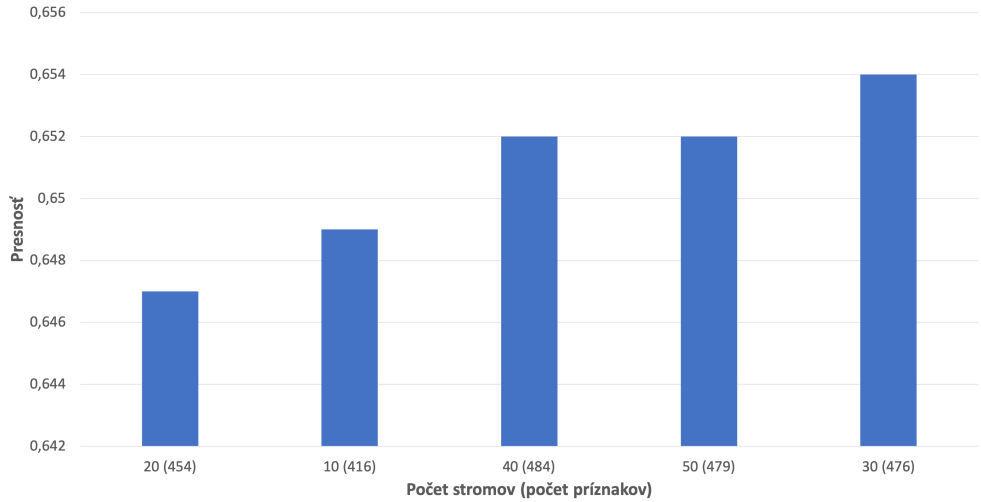
V ďalších častiach budeme používať príznaky vybraté pomocou metódy SelectKBest nakoľko v porovnaní s počtom využitých príznakov, dosahujeme oveľa väčšiu presnosť. Rozhodli sme sa zvoliť 40 príznakov ako kompromis medzi počtom príznakov a rýchlosťou trénovacieho procesu.

Medzi vybrané príznaky patria tieto skupiny príznakov:

- Všeobecné informácie
- Histogram bajtov
- Histogram entropie bajtov
- Informácie o reťazcoch



Obr. 52: Presnosť natrénovaného modelu v porovnaní s počtom využitých príznakov vybraných pomocou metódy SelectKBest.



Obr. 53: Presnosť natrénovaného modelu v porovnaní s počtom použitých stromov resp. počet príznakov vybraných pomocou metódy SelectFromModel.

5.4 Trénovanie modelov

V tejto časti si detailnejšie ukážeme, ako sme postupovali pri trénovaní jednotlivých modelov. Na trénovanie sme použili tri modely, a to rozhodovací strom, náhodný les, a K-najbližších susedov. Predtým, než si popíšeme jednotlivé implementované algoritmy, uvádzame jednu funkciu z knižnice *scikit-learn*, a to kalibrovanie modelu, ktorá bude kvôli použitiu niektorých algoritmov veľmi dôležitá.

5.4.1 Kalibrácia modelu

V kapitole 3, ktorá sa zaoberala algoritmami semi-supervised learningu, sme sa už dozvedeli, že pri klasifikácii jednotlivých vzoriek nás nebude zaujímať len samotná kategória, do ktorej skúmajúca vzorka patrí, ale takisto bude pre nás dôležitá pravdepodobnosť jej označenia. Táto pravdepodobnosť nám dáva akúsi istotu predikcie, ktorá je napr. pri algoritme Self - Training alebo Co - Training veľmi dôležitá. Na základe tejto pravdepodobnosti, sa náš model môže zlepšovať aj pomocou neoznačených vzoriek.

Niektoré modely však poskytujú dosť nepresný odhad pravdepodobností, s akou daná vzorka patrí do určitej kategórie a niektoré zas takúto možnosť neposkytujú vôbec. Táto funkcia dokáže skalibrovať náš model a vylepšiť predpovedanú pravdepodobnosť, poprípade modelom, ktoré touto funkciou nedisponujú, ju kalibráciou môžeme pridať.

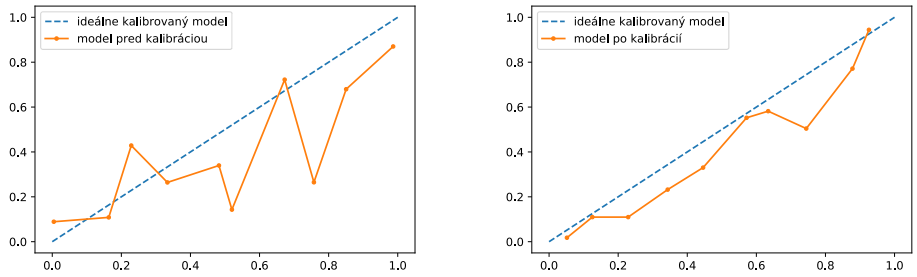
Podľa [55] sú dobre kalibrované klasifikátory také, ktorých výstup funkcie predict_proba môže byť priamo interpretovaný ako určitý level spoľahlivosti. Správne kalibrované binárne klasifikátory by mali klasifikovať vzorky tak, aby pravdepodobnosť kategórie, do ktorej vzorka patrí bola zhruba 80%. Detailnejšie informácie ako funguje samotná kalibrácia modelov, je možné nájsť priamo v dokumentácií [55] knižnice *scikit-learn*. O tom, ako sa vyjadruje stav modelu, si povieme hneď v nasledujúcej časti.

5.4.2 Self - Training

Pri realizácii tohto algoritmu (viď kapitola 3.1) sme použili 25 000 označených vzoriek (11 928 legitímnych súborov a 13 072 malwaru) a 100 000 neoznačených vzoriek. Pre porovnanie výsledkov sme natrénovali tri rôzne modely, a to rozhodovací strom, náhodný les a model K - najbližších susedov. Testovanie prebiehalo na dátovej sade o veľkosti 50 000 vzoriek.

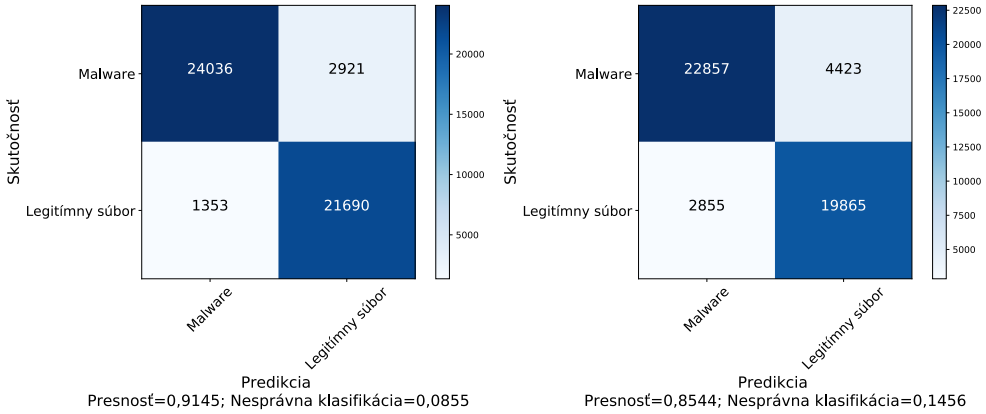
Rozhodovací strom

Po natrénovaní rozhodovacieho stromu, sme zistili, že náš model nespĺňa podmienky správne kalibrovaného klasifikátora, ako sme si uviedli v predchádzajúcej časti. Klasifikátor priradil mnohým vzorkám 100% pravdepodobnosť zaradenia do správnej kategórie, čo podľa [55] nesplňuje požiadavku správne



(a) Stav modelu pred kalibráciou (b) Stav modelu po kalibrácií

Obr. 54: Priebeh kalibrácie modelu - rozhodovací strom



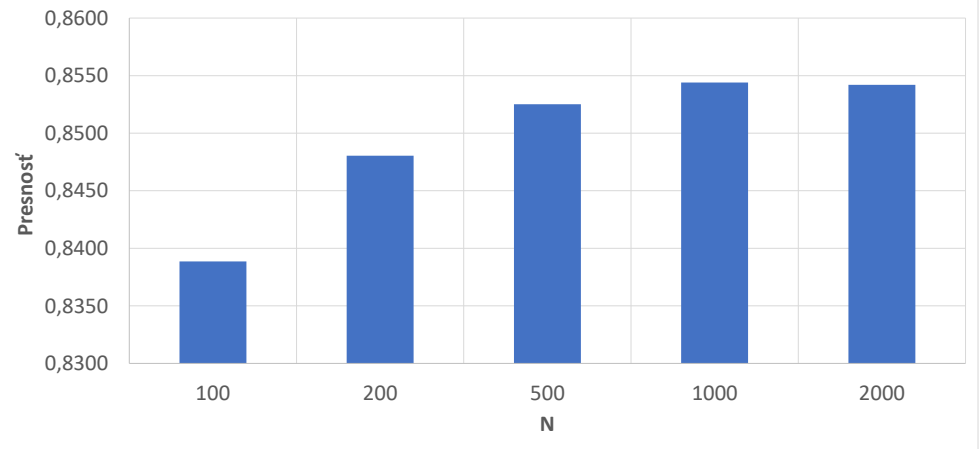
(a) pred Self - Trainingom (b) po Self - Trainingu

Obr. 55: Konfúzna matica - rozhodovací strom

kalibrovaného modelu. Predtým, než sme aplikovali samotný algoritmus Self-Trainingu bolo potrebné náš model skalibrovať.

Na grafe 54a vidíme náš model pred samotnou kalibráciou. Diagonála (prerušovaná čiara) predstavuje ideálne nakalibrovaný model, oranžová krivka zas predstavuje kalibráciu nášho modelu. Čím je oranžová krivka bližšie k diagonále, tým je lepšie model kalibrovaný. Ak je krivka pod diagonálou, model má prehnanú predpoveď, a teda pravdepodobnosti sú príliš veľké. Naopak ak je krivka pod diagonálou, model má slabú predpoveď a pravdepodobnosti sú nízke. Na grafe 54b už vidíme náš model po kalibrácií. Krivka sa značne presunula bližšie k ideálnemu modelu. Samotnou kalibráciou sa väčšinou zlepší aj presnosť modelu, a to nebolo výnimkou ani v našom prípade. Z pôvodných 0,88 sme sa dostali na hodnotu 0,91. Na obrázku 55a je zobrazená konfúzna matica tohto modelu.

Po kalibrácií sme na daný model už aplikovali algoritmus Self-Training



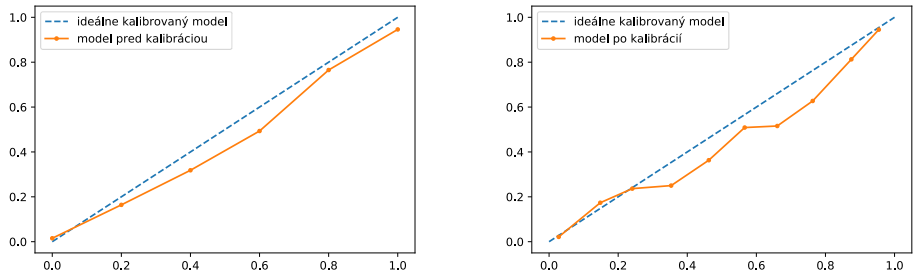
Obr. 56: Presnosť modelu v závislosti na voľbe N - rozhodovací strom

podľa postupu, ako sme si uviedli v časti 3.1. Postupne sme za N volili hodnoty v rozmedzí od 100 po 2000, čo znamenalo koľko najviac pravdepodobných novooznačných vzoriek sa presunie z neoznačenej dátovej sady do označenej sady, ktorou bude pôvodný model pretrénovaný. Pri všetkých zvolených hodnotách však dochádzalo k zníženiu presnosti, čo bude najskôr dôsledkom toho, že model nesprávne označil niektoré vzorky a táto chyba bola ďalej propagovaná, čo spôsobilo aj zhoršenie predikcie. Na grafe 56 je znázornené, ako sa menila presnosť modelu v závislosti na zvolenej hodnote N. Najlepší výsledok sme dosiahli pri hodnote 1000 (viď konfúzna matica na obrázku 55b). S následným zvyšovaním hodnoty N sa presnosť už nijak razantne nemenila.

K - najbližších susedov

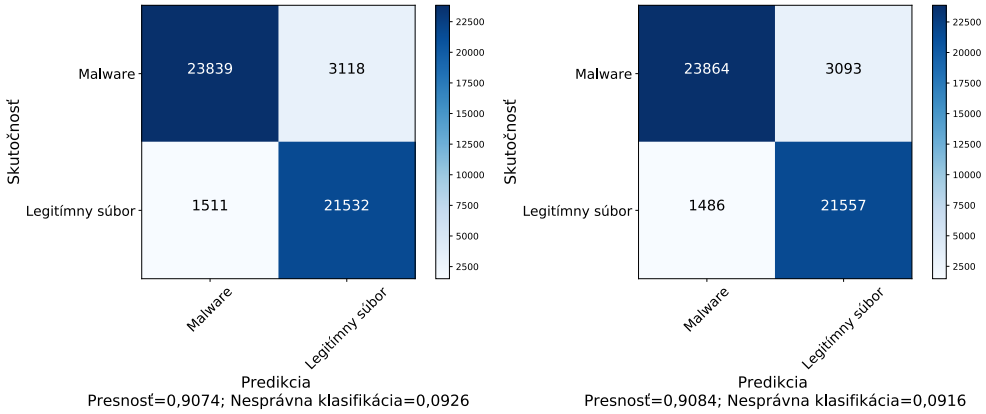
Pri tomto modeli sme postupovali podobne. Pred samotným trénovaním bolo však potrebné dáta na rozdiel od rozhodovacích stromov najprv naškálovať. Následne sme hľadali najvhodnejší počet najbližších susedov. Výsledok experimentu ukázal, že najlepšia hodnota je 5 susedov. V porovnaní s rozhodovacím stromom je tento model o čosi menej presný, no ako si môžeme všimnúť na grafe 57, je model oveľa lepšie kalibrovaný, aj bez kalibrácie. Hodnoty pravdepodobností predikovaných vzoriek sú takmer rovnaké aj bez kalibrácie, preto sme sa ďalej rozhodli používať nekalibrovaný model, ktorý je o čosi rýchlejší. Konfúzna matica nekalibrovaného modelu je zobrazená na obrázku 58.

Pri aplikovaní algoritmu Self-Training sme postupovali podobne ako aj pri modely rozhodovacieho stromu. Počet presunutých najviac pravdepodobných vzoriek z neoznačenej sady do označenej sady sme opäť volili od 100 po 2000. V tomto prípade sme zaznamenali malé zlepšenie presnosti modelu. Môžeme povedať, že pre všetky hodnoty N bol rozdiel v presnosti zaned-



(a) Stav modelu pred kalibráciou (b) Stav modelu po kalibrácií

Obr. 57: Priebeh kalibrácie modelu - K - najbližších susedov



(a) pred Self - Trainingom (b) po Self - Trainingu

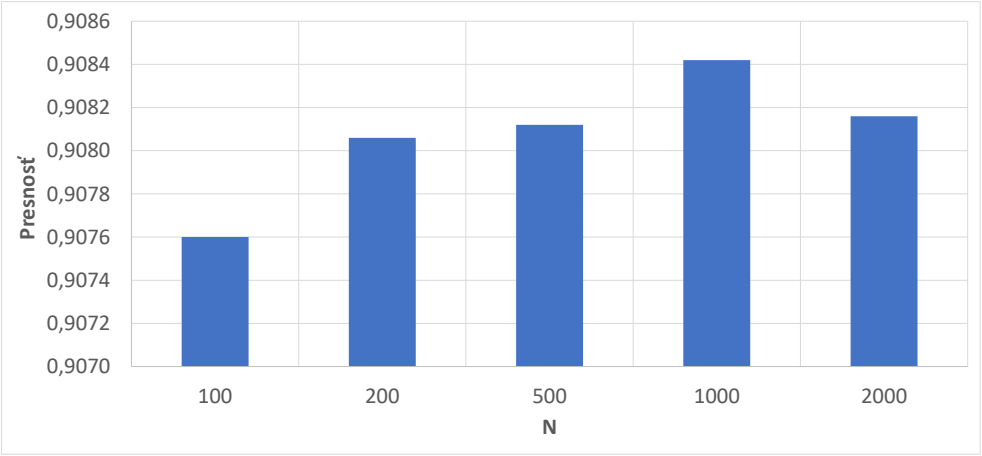
Obr. 58: Konfúzna matica - K- najbližších susedov

bateľný, no napriek tomu najvyššiu presnosť sme dosiahli pre $N = 1000$, a to $0,90842$. Na grafe 59 môžeme vidieť porovnanie dosiahnutej presnosti v závislosti na voľbe N.

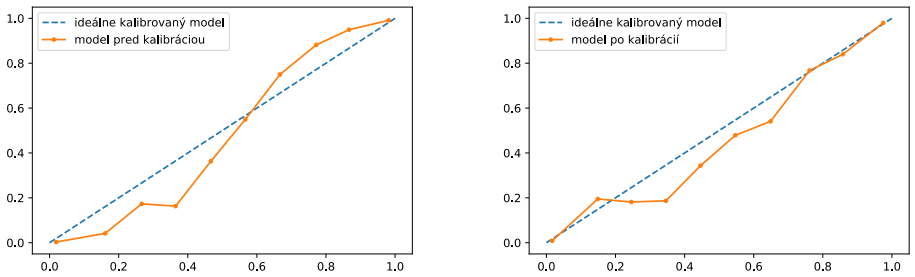
Náhodný les

Náhodný les je veľmi podobný rozhodovaciemu stromu s tým rozdielom, že obsahuje súbor niekoľkých rozhodovacích stromov. Nám sa ako najlepší počet rozhodovacích stromov v náhodnom lese javil 15, kedy sme dosiahli presnosť už aj s kalibráciou $0,9389$. Priebeh kalibrácie je znázornený na grafe 510.

Podobne ako aj pri rozhodovacom strome došlo po aplikácii algoritmu Self-Training k zhoršeniu presnosti modelu. Na grafe 511 si môžeme všimnúť dosiahnuté presnosti natrénovaného modelu v závislosti na voľbe hodnoty N. Obrázok 512 zobrazuje porovnanie konfúznych matíc pred a po aplikovaní algoritmu Self - Training.



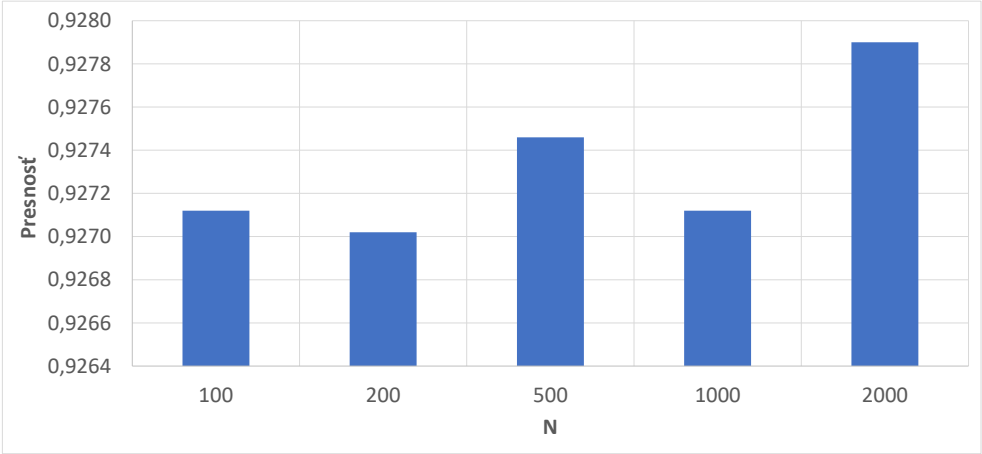
Obr. 59: Presnosť modelu v závislosti na voľbe N - K- najbližších susedov



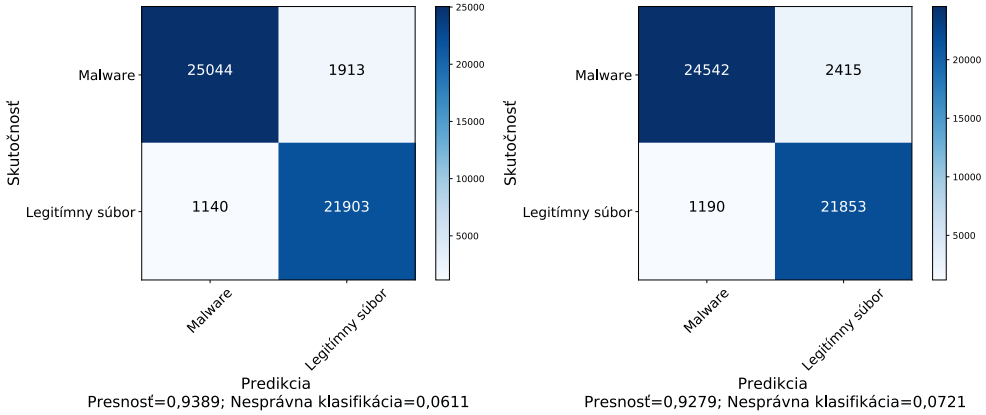
(a) Stav modelu pred kalibráciou

(b) Stav modelu po kalibrácií

Obr. 510: Priebeh kalibrácie modelu - náhodný les



Obr. 511: Presnosť modelu v závislosti na voľbe N - náhodný les



(a) pred Self - Trainingom (b) po Self - Trainingu

Obr. 512: Konfúzna matica - náhodný les

5.4.3 Co - Training

Algoritmus sme implementovali rovnako, ako je znázornený v kapitole 3.2. K trénovaniu modelov sme použili dve množiny označených vzoriek o veľkosti 25 000 (množina L_1 obsahovala 13 072 vzoriek malwaru a 11 928 vzoriek legitímnych súborov, množina L_2 zas 11 708 vzoriek malwaru a 13 292 vzoriek legitímnych súborov). Prvá množina obsahovala rovnaké príznaky, ako v predchadzajúcich prípadoch. V prípade druhej množiny sme na výber najlepších príznakov použili opäť metódu SelectKBest, no vylúčili sme príznaky, ktoré boli vybrané pre prvú množinu. Medzi vybrané príznaky pre množinu L_2 patria tieto skupiny príznakov:

- Hlavičkové informácie
- Importované funkcie
- Informácie o sekciách

Na realizáciu tohto algoritmu sme postupne kombinovali tri modely, a to rozhodovacie stromy, náhodný les, a K-najbližších susedov, ktoré sme použili aj v predchádzajúcej časti. Konfigurácia jednotlivých modelov bola rovnaká ako v časti o Self-Trainingu.

V iteráciách, kedy sme postupne neoznačené vzorky označovali a pridávali do označených dátových sád, bolo pridávaných pre každý model vždy 2000 nových vzoriek. Natrénované modely sme testovali s rovnakou dátovou sadou, ako aj pri Self-Trainingu. Pre predikciu po natrénovaní oboch modelov s neoznačenými vzorkami sme využili priemerovanie, a to tak, že predikované hodnoty pravedpodobností oboch modelov sme sčítali a vydelili dvomi. Podľa výslednej hodnoty sme rozhodli, či je súbor legitímny alebo ide o malware.

	Rozhodovací strom	Náhodný les	Priemer
pred **Co - Trainingom**	0,9145	0,8758	0,9370
po **Co - Trainingu**	0,8983	0,8009	0,9076

Tabuľka 51: Porovnanie presností modelov - rozhodovací strom a náhodný les

	Rozhodovací strom	K-najbližších susedov	Priemer
pred **Co - Trainingom**	0,9145	0,8639	0,9073
po **Co - Trainingu**	0,8928	0,7984	0,9001

Tabuľka 52: Porovnanie presností modelov - rozhodovací strom a K-najbližších susedov

	Náhodný les	K-najbližších susedov	Priemer
pred **Co - Trainingom**	0,9389	0,8639	0,9327
po **Co - Trainingu**	0,9283	0,8672	0,9205

Tabuľka 53: Porovnanie presností modelov - náhodný les a K-najbližších susedov

Rozhodovací strom - Náhodný les

Kombinácia rozhodovacieho stromu a náhodného lesu bola jediná, pri ktorej sme dospeli k výraznejšiemu zhoršeniu presnosti. Dosiahnuté presnosti uvádzame v tabuľke 51, kde sú znázornené presnosti ako samotných modelov, tak aj ich výsledok po priemerovaní pred aplikáciou algoritmu Co-Training a po jeho aplikovaní. Môžeme si všimnúť, že z presnosti 0,937 sme aplikovaním Co-Trainingu klesli na hodnotu 0,9076.

Rozhodovací strom - K-najbližších susedov

V tabuľke 52 sú znázornené presnosti rozhodovacieho stromu a K-najbližších susedov. V tomto prípade sa presnosť zhoršila len zanedbateľne.

Náhodný les - K-najbližších susedov

Rovnako, ako aj v predchádzajúcom prípade, sa nám pri kombinácií náhodného lesa a K-najbližších susedov presnosť zhoršila približne o jednu stotinu. Presnosti sú znázornené v tabuľke 53.

	Náhodný les	K - najbližších susedov	Rozhodovací strom	Priemer	Hlasovanie
pred	0,8758	0,8838	0,8876	0,9215	0,9126
po	0,8754	0,8601	0,8856	0,9020	0,9017

Tabuľka 54: Porovnanie presností modelov - pred aplikovaním modifikovaného Co-Trainingu a následne po jeho aplikácií

5.4.4 Label Spreading

Knižnica *scikit-learn* obsahuje aj priamo algoritmy pre semi - supervised learning. Jedným z nich je aj algoritmus Label Spreading, ktorý by podľa dokumentácie mal fungovať podobne ako algoritmus LLGC, ktorý sme si spomenuli v časti 3.5. Ten teda priamo pracuje s jednou dátovou sadou, v ktorej sa nachádzajú označené aj neoznačené vzorky. My sme použili sadu, ktorá obsahovala 24 780 vzoriek malwaru, 25 220 vzoriek legitímnych súborov a 150 000 neoznačených vzoriek. Dosiahnutá presnosť tohto algoritmu bola 0,7171.

5.4.5 Modifikovaný Co-Training

Pri implementácií upraveného algoritmu Co - Training nás z časti inšpiroval algoritmus Tri - Training, ktorý sme si spomenuli v časti 3.3. K dvom nezávislým klasifikátorom sme pridali tretí, a každý jeden klasifikátor učil ostatné dva. Použili sme tri klasifikátory, a to rozhodovací strom, náhodný les a K-najbližších susedov. Na počiatočné trénovanie s označenými vzorkami boli použité tri množiny o veľkosti 25 000 prvkov. V tomto prípade došlo k porušeniu podmienky nezávislosti, keďže množina L_3 obsahovala pár príznakov z L_2. Množina L_1 obsahovala 13 072 vzoriek malwaru a 11 928 vzoriek legitímnych súborov, množina L_2 11 708 vzoriek malwaru a 13 292 vzoriek legitímnych súborov a napokon množina L_3 13 602 vzoriek malwaru a 11 398 vzoriek legitímnych súborov. Množina neoznačených dát obsahovala 150 000 vzoriek, a výslednú presnosť sme testovali s rovnakou množinou ako aj v predchádzajúcich prípadoch, kde bolo 50 000 vzoriek. Pre výslednú predikciu sme použili obe možnosti, priemerovanie (podobne ako pri Co-Trainingu) a hlasovanie. Pseudokód algoritmu je naznačený v algoritme 4.

Po aplikovaní nášho algoritmu, došlo opäť v oboch prípadoch k malému zhoršeniu presnosti. V tabuľke 54 sú znázornené dosiahnuté presnosti jednotlivých klasifikátorov.

5.5 Vyhodnotenie

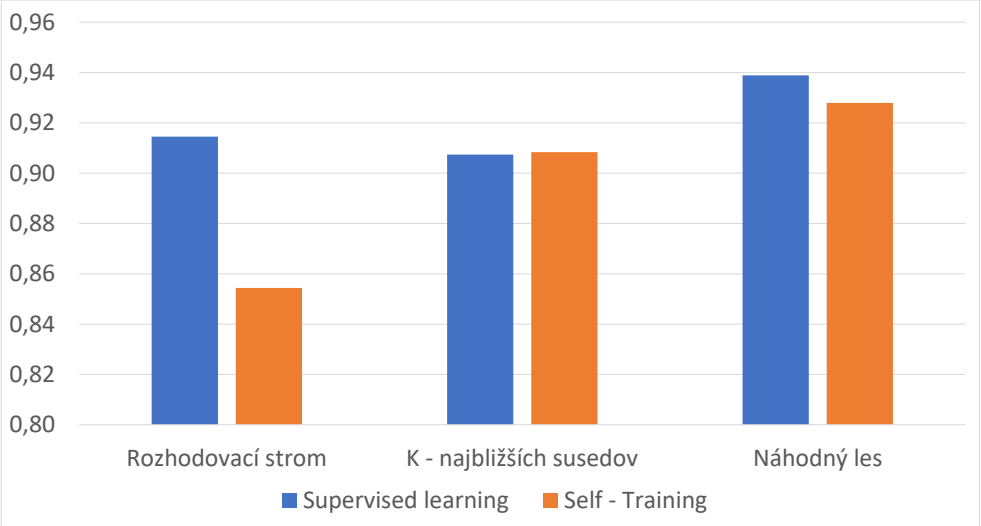
V rámci tejto kapitoly sme natrénovali niekoľko modelov a použili viacero rôznych typov algoritmov semi - supervised learningu. Dosiahnuté výsledky v porovnaní so supervised learning algoritmami vo väčšine prípadov nie sú ani

Algoritmus 4 Pseudokód modifikovaného algoritmu Co - Training

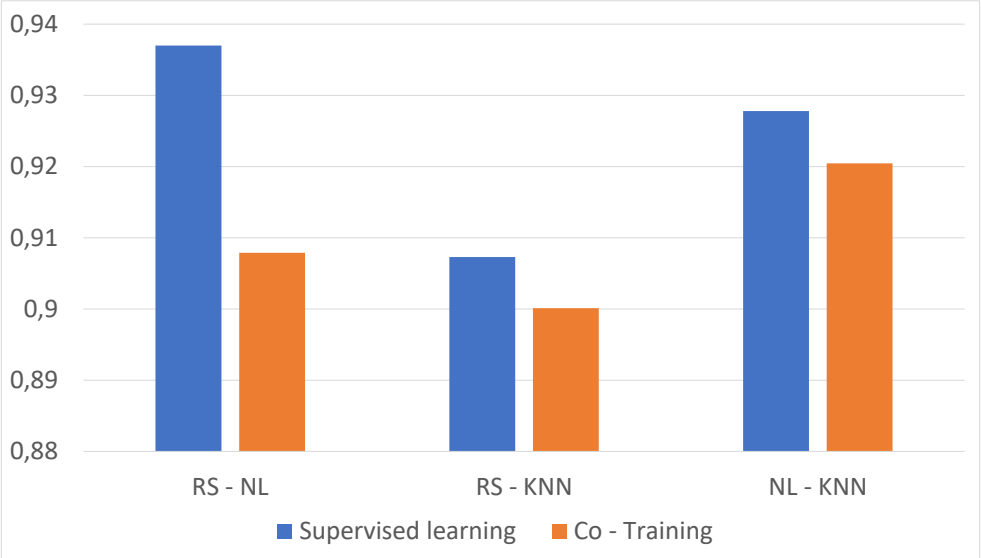
VSTUP: L - sada označených dát, U - sada neoznačených dát
VÝSTUP: Natrénované dva modely pomocou označených a neoznačených dát

Rozdeľ L do troch označených sád L_1, L_2, L_3
M_1 = natrénuj supervised learning model s L_1
M_2 = natrénuj supervised learning model s L_2
M_3 = natrénuj supervised learning model s L_3
while $U \neq \emptyset$ **do**
    Aplikuj M_1, M_2 a M_3 na neoznačenú sadu dát U
    Vyber N vzoriek s najväčšou pravdepodobnosťou správnej klasifikácie modelu M_1, M_2 a M_3
    Pridaj najviac pravdepodobné vzorky predpovedané M_1 do sady L_2 a L_3
    Pridaj najviac pravdepodobné vzorky predpovedané M_2 do sady L_1 a L_3
    Pridaj najviac pravdepodobné vzorky predpovedané M_3 do sady L_1 a L_2
    Odober tieto vzorky z U
    Pretrénuj model M_1 s L_1, M_2 s L_2 a M_3 s L_3
end while
Pre následnú predikciu využi priemer alebo hlasovanie

lepšie, no ani horšie. Väčšinou sme najväčšie zhoršenie presnosti pozorovali pri použití klasifikátora rozhodovací strom, no naopak pri klasifikátore K-najbližších susedov, v prípade Self-Trainingu, zas malé zlepšenie presnosti. Na grafoch 513 a 514 je zobrazené súhrnné porovnanie presností algoritmov Self-Training a Co-Training.



Obr. 513: Porovnanie presností Supervised learningu s Self - Training algoritmom



Obr. 514: Porovnanie presností Supervised learningu s Co - Training algoritmom (RS - rozhodovací strom, NL - náhodný les, KNN - K - najbližších susedov)

Záver

Cieľom tejto práce bolo implementovať semi-supervised learning algoritmy, ktoré sú vhodné pre detekciu malwaru a porovnať dosiahnuté výsledky so supervised learning algoritmami. Podarilo sa nám implementovať algoritmy Self-Training a Co-Training, ktoré sme otestovali v kombinácii s niekoľkými rôznymi klasifikátormi. Tieto algoritmy v jednom prípade vylepšili presnosť výsledného modelu, na základe dotrénovania klasifikátorov pomocou neoznačených vzoriek. Otestovali sme aj algoritmus založený na teórii grafov, ten však priniesol oproti predchádzajúcim spomenutým algoritmom najmenej uspokojivé výsledky.

Ďalším cieľom bolo modifikovať jeden z existujúcich semi-supervised learning algoritmov. Modifikovali sme algoritmus Co-Training, kde sme ku dvom nezávislým klasifikátorom pridali tretí. Aplikovaním tohto algoritmu sa celková presnosť nijak výrazne nezmenila.

K samotnému trénovaniu sme z použitej dátovej sady vybrali len niekoľko príznakov pomocou algoritmu pre výber príznakov z knižnice *scikit-learn*. Pre ďalšie pokračovanie v tejto práci by bolo vhodné vyberať tieto príznaky na základe iných postupov a porovnať následnú presnosť modelov s presnosťou modelov dosiahnutých v tejto práci.

Literatúra

[1] van Tilborg, H. C. A.; Jajodia, S. (editoři): *Malware*. Boston, MA: Springer US, 2011, ISBN 978-1-4419-5906-5, s. 750–752, doi:10.1007/978-1-4419-5906-5_1320. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_1320

[2] AV-TEST - The Independent IT-Security Institute. *Malware Statistics and Trends Report [online]*, [cit. 2019-01-22]. Dostupné z: https://www.av-test.org/en/statistics/malware/

[3] Sukwong, O.; Kim, H.; Hoe, J.: Commercial Antivirus Software Effectiveness: An Empirical Study. *Computer*, ročník 44, č. 3, March 2011: s. 63–70, ISSN 0018-9162, doi:10.1109/MC.2010.187.

[4] Kuncheva, L. I.: Full-class set classification using the Hungarian algorithm. *International Journal of Machine Learning and Cybernetics*, ročník 1, č. 1, Dec 2010: s. 53–61, ISSN 1868-808X, doi:10.1007/s13042-010-0002-z. Dostupné z: https://doi.org/10.1007/s13042-010-0002-z

[5] Kang, B.; Kim, T.; Kwon, H.; aj.: Malware Classification Method via Binary Content Comparison. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, RACS '12, New York, NY, USA: ACM, 2012, ISBN 978-1-4503-1492-3, s. 316–321, doi:10.1145/2401603.2401672. Dostupné z: http://doi.acm.org/10.1145/2401603.2401672

[6] Kolter, J. Z.; Maloof, M. A.: Learning to Detect and Classify Malicious Executables in the Wild. *J. Mach. Learn. Res.*, ročník 7, Prosinec 2006: s. 2721–2744, ISSN 1532-4435. Dostupné z: http://dl.acm.org/citation.cfm?id=1248547.1248646

[7] Sami, A.; Yadegari, B.; Rahimi, H.; aj.: Malware Detection Based on Mining API Calls. In *Proceedings of the 2010 ACM Symposium on App-*

lied Computing, SAC '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-639-7, s. 1020–1025, doi:10.1145/1774088.1774303. Dostupné z: http://doi.acm.org/10.1145/1774088.1774303

[8] Longford, N. T.: 12 - FISHER SCORING ALGORITHM FOR VARIANCE COMPONENT ANALYSIS OF DATA WITH MULTILEVEL STRUCTURE. In *Multilevel Analysis of Educational Data*, editace R. D. Bock, San Diego: Academic Press, 1989, ISBN 978-0-12-108840-8, s. 297 – 310, doi:https://doi.org/10.1016/B978-0-12-108840-8.50019-6. Dostupné z: http://www.sciencedirect.com/science/article/pii/B9780121088408500196

[9] Lee, J.; Jeong, K.; Lee, H.: Detecting Metamorphic Malwares Using Code Graphs. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-639-7, s. 1970–1977, doi:10.1145/1774088.1774505. Dostupné z: http://doi.acm.org/10.1145/1774088.1774505

[10] Saini, A.; Gandotra, E.; Bansal, D.; aj.: Classification of PE Files Using Static Analysis. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, New York, NY, USA: ACM, 2014, ISBN 978-1-4503-3033-6, s. 429:429–429:433, doi:10.1145/2659651.2659679. Dostupné z: http://doi.acm.org/10.1145/2659651.2659679

[11] Willems, C.; Holz, T.; Freiling, F.: Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security Privacy*, ročník 5, č. 2, March 2007: s. 32–39, ISSN 1540-7993, doi:10.1109/MSP.2007.45.

[12] The Sandbox. *The Sandbox — Understanding CyberForensics [online]*, [cit. 2019-01-25]. Dostupné z: https://cwsandbox.org

[13] Bayer, U.; Moser, A.; Kruegel, C.; aj.: Dynamic Analysis of Malicious Code. *Journal in Computer Virology*, ročník 2, č. 1, Aug 2006: s. 67–77, ISSN 1772-9904, doi:10.1007/s11416-006-0012-2. Dostupné z: https://doi.org/10.1007/s11416-006-0012-2

[14] SecuriTeam™. *Red Pill... Or How To Detect VMM Using (Almost) One CPU Instruction [online]*, [cit. 2019-01-26]. Dostupné z: https://cwsandbox.org

[15] Salomon, D.: *Foundations of Computer Security*. Springer, London, 2007, ISBN 978-1-84628-341-3, s. 33–124.

[16] Norton by Symantec. *The 8 Most Famous Computer Viruses of All Time [online]*, [cit. 2019-01-26]. Dostupné z: https://uk.norton.com/norton-blog/2016/02/the_8_most_famousco.html

[17] PCWorld. *Storm: The Largest Botnet in the World? [online]*, [cit. 2019-01-26]. Dostupné z: https://www.pcworld.com/article/137854/article.html

[18] VERACODE. *Common Malware Types: Cybersecurity 101 [online]*, 2019, [cit. 2019-01-28]. Dostupné z: https://www.veracode.com/blog/2012/10/common-malware-types-cybersecurity-101

[19] Comodo Group, Inc. *What is Backdoor Virus? [online]*, 2019, [cit. 2019-01-28]. Dostupné z: https://antivirus.comodo.com/blog/comodo-news/backdoor-virus/

[20] Malwarebytes. *Bootkit [online]*, 2018, [cit. 2019-01-29]. Dostupné z: https://blog.malwarebytes.com/detections/bootkit/

[21] Symantec Corporation. *Coinminer protection and removal with Endpoint Protection [online]*, 2018, [cit. 2019-01-29]. Dostupné z: https://support.symantec.com/en_US/article.TECH249302.html

[22] Symantec Corporation. *Downloader [online]*, 2018, [cit. 2019-01-29]. Dostupné z: https://www.symantec.com/security-center/writeup/2002-101518-4323-99

[23] Trend Micro Incorporated. *exploit - Definition [online]*, 2018, [cit. 2019-01-30]. Dostupné z: https://www.trendmicro.com/vinfo/us/security/definition/exploit

[24] Malwarebytes. *Ransomware - What Is It and How To Remove It [online]*, 2019, [cit. 2019-01-30]. Dostupné z: https://www.malwarebytes.com/ransomware/

[25] Malwaretruth. *A List of Malware Types and Their Definitions [online]*, 2019, [cit. 2019-01-30]. Dostupné z: http://www.malwaretruth.com/the-list-of-malware-types/

[26] MANAGED SOLUTION. *4 Common Types of Spyware and How To Detect Them [online]*, 2019, [cit. 2019-01-30]. Dostupné z: https://www.managedsolution.com/4-common-types-of-spyware-and-how-to-detect-them/

[27] ESET North America. *What is a potentially unwanted application or potentially unwanted content? [online]*, 2019, [cit. 2019-01-30]. Dostupné z: https://support.eset.com/kb2629/?locale=en_US&viewlocale=en_US

[28] Murphy, K. P.: *Machine Learning - A Probabilistic Perspective*. Massachusetts Institute of Technology, 2012, ISBN 978-0-262-01802-9, s. 33–124.

[29] Towards Data Science. *The 7 Steps of Machine Learning [online]*, 2019, [cit. 2019-02-01]. Dostupné z: https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e

[30] Towards Data Science. *Supervised vs. Unsupervised Learning [online]*, 2019, [cit. 2019-02-02]. Dostupné z: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[31] Machine Learning Mastery. *Overfitting and Underfitting With Machine Learning Algorithms [online]*, 2019, [cit. 2019-02-05]. Dostupné z: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[32] Machine Learning Mastery. *Supervised and Unsupervised Machine Learning Algorithms [online]*, 2019, [cit. 2019-02-05]. Dostupné z: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[33] Schultz, M. G.; Eskin, E.; Zadok, F.; aj.: Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, May 2001, ISSN 1081-6011, s. 38–49, doi:10.1109/SECPRI.2001.924286.

[34] Islam, R.; Tian, R.; Batten, L.; aj.: Classification of Malware Based on String and Function Feature Selection. In *2010 Second Cybercrime and Trustworthy Computing Workshop*, July 2010, s. 9–17, doi:10.1109/CTC.2010.11.

[35] Bai, J.; Wang, J.; Zou, G.: A Malware Detection Scheme Based on Mining Format Information. In *The Scientific World Journal, Volume 2014*, June 2014, ISSN 260905, str. 11.

[36] Santos, I.; Brezo, F.; Ugarte-Pedrero, X.; aj.: Opcode Sequences As Representation of Executables for Data-mining-based Unknown Malware Detection. *Inf. Sci.*, ročník 231, Květen 2013: s. 64–82, ISSN 0020-0255, doi:10.1016/j.ins.2011.08.020. Dostupné z: https://doi.org/10.1016/j.ins.2011.08.020

[37] Guyon, I.; Elisseeff, A.: An Introduction to Variable and Feature Selection. *J. Mach. Learn. Res.*, ročník 3, Březen 2003: s. 1157–1182, ISSN 1532-4435. Dostupné z: http://dl.acm.org/citation.cfm?id=944919.944968

[38] Webb, A. R.: *Statistical Pattern Recognition, Second Edition*. John Wiley and Sons Ltd, 2002, ISBN 0-470-84514-7, s. 307–318.

[39] Jović, A.; Brkić, K.; Bogunović, N.: A review of feature selection methods with applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2015, s. 1200–1205, doi:10.1109/MIPRO.2015.7160458.

[40] Analytics Vidhya. *Feature Selection methods with example (Variable selection methods) [online]*, [cit. 2019-04-25]. Dostupné z: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

[41] Huan Liu, H. M.: *Computational Methods of Feature Selection*. Huan Liu, Hiroshi Motoda, 2007, ISBN 9781584888796, str. 153.

[42] Fielding, A. H.; Bell, J. F.: A review of methods for the assessment of prediction errors in conservation presence/absence models. In *Environmental Conservation 24*, December 1996, s. 38–49.

[43] Exsilio Solutions. *Accuracy, Precision, Recall and F1 Score [online]*, 2019, [cit. 2019-03-01]. Dostupné z: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

[44] Zhu, X.: *Semi-Supervised Learning Literature Survey*. Diplomová práce, University of Wisconsin – Madison, 2007.

[45] Li, M.; Zhou, Z.-H.: SETRED: Self-training with Editing. In *Advances in Knowledge Discovery and Data Mining*, editace T. B. Ho; D. Cheung; H. Liu, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ISBN 978-3-540-31935-1, s. 611–621.

[46] Blum, A.; Mitchell, T.: Combining Labeled and Unlabeled Data with Co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, New York, NY, USA: ACM, 1998, ISBN 1-58113-057-0, s. 92–100, doi:10.1145/279943.279962. Dostupné z: http://doi.acm.org/10.1145/279943.279962

[47] Zhou, Z.-H.; Li, M.: Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, ročník 17, č. 11, Nov 2005: s. 1529–1541, ISSN 1041-4347, doi: 10.1109/TKDE.2005.186.

[48] Blum, A.; Chawla, S.: Learning from Labeled and Unlabeled Data Using Graph Mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, ISBN 1-55860-778-1, s. 19–26. Dostupné z: http://dl.acm.org/citation.cfm?id=645530.757779

[49] Zhou, D.; Bousquet, O.; Lal, T. N.; aj.: Learning with Local and Global Consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, Cambridge, MA, USA: MIT Press, 2003, s. 321–328. Dostupné z: http://dl.acm.org/citation.cfm?id=2981345.2981386

[50] Anderson, H. S.; Roth, P.: EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *CoRR*, ročník abs/1804.04637, 2018, 1804.04637. Dostupné z: http://arxiv.org/abs/1804.04637

[51] Weinberger, K.; Dasgupta, A.; Langford, J.; aj.: Feature Hashing for Large Scale Multitask Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-516-1, s. 1113–1120, doi: 10.1145/1553374.1553516. Dostupné z: http://doi.acm.org/10.1145/1553374.1553516

[52] Saxe, J.; Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, Oct 2015, s. 11–20, doi:10.1109/MALWARE.2015.7413680.

[53] Liu, Y. H.: *Machine Learning - A Probabilistic Perspective*. Python Machine Learning By Example, 2017, ISBN 9781783553112, s. 50–52.

[54] C Ross, B.: Mutual Information between Discrete and Continuous Data Sets. *PloS one*, ročník 9, 02 2014: str. e87357, doi:10.1371/journal.pone.0087357.

[55] scikit-learn. *Probability calibration [online]*, 2019, [cit. 2019-04-16]. Dostupné z: https://scikit-learn.org/stable/modules/calibration.html

Dodatok **A**

Zoznam použitých skratiek

DLL Dynamically Loaded Library

AI Artificial intelligence

IAT Import Address Table

PE Portable Executable format

PUP Potentially Unwanted Program

SBS Sequential Backward Selection

SFS Sequential Forward Selection

SVM Support Vector Machine

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
  - `zadanie.pdf` ..... zadanie práce vo formáte PDF