Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Algoritmy detekcie malvéru na báze strojového učenia a statickej analýzy kódu

Diplomová práca

2019 **Bc. Lukáš Židzik**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Algoritmy detekcie malvéru na báze strojového učenia a statickej analýzy kódu

Diplomová práca

Štúdijný program:	Informatika
Štúdijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Ján Hurtuk, PhD.

Košice 2019 **Bc. Lukáš Židzik**

Názov práce:	Algoritmy detekcie malvéru na báze strojového učenia a statickej analýzy kódu
Pracovisko:	Katedra počítačov a informatiky, Technická univerzita v Košiciach
Autor:	Bc. Lukáš Židzik
Školiteľ:	Ing. Ján Hurtuk, PhD.
Dátum:	23. 4. 2019
Kľúčové slová:	Statická analýza, strojové učenie, diplomová práca, malvér

Abstrakt: Diplomová práca sa zaoberá problematikou detekcie škodlivého softvéru. Cieľom tejto práce bolo overiť, či sa strojové učenie dá využiť na vytvorenie takého klasifikačného modelu, ktorý bude vedieť spoľahlivo predikovať neznáme vzorky do dvoch tried (škodlivá / čistá). Dôraz sa kládol na čo najvyššiu presnosť modelu a čo najmenší výskyt nedetekovaných vzoriek škodlivého softvéru. V práci je opísaná základná a pokročilá statická analýza malvéru a jej techniky. Práca taktiež analyzuje strojové učenie, klasifikačné modely a vyhodnocovacie metriky. V rámci praktickej časti je natrénovaných viacero modelov využívajúc rôzne algoritmy. Dáta potrebné na natrénovanie modelov sú získané v procese statickej analýzy vzoriek. Všetky modely dosahujú vysokú úspešnosť. V závere práce sú zhrnuté výsledky s návrhmi ďalších rozšírení tejto práce.

Thesis title:	Malware detection algorithms based on machine learning and static malware analysis
Department:	Department of Computers and Informatics, Technical university of Košice
Author:	Bc. Lukáš Židzik
Supervisor:	Ing. Ján Hurtuk, PhD.
Date:	23. 4. 2019
Keywords:	Static analysis, machine learning, diploma thesis, malware

Abstract: This diploma thesis deals with the issue of malware detection. The aim of this thesis was to prove that machine learning is suitable for creating classification model, which will be able to reliably predict an unknown sample into two classes (malicious / beging). Emphasis was placed on as high as possible accuracy and the smallest number of undetected malicious samples. This thesis describes basic and advanced static analysis, their's methods and techniques, machine learning methods, classification models and evaluation metrics. The practical part includes training of severel models using different algorythms. Data needed for training are collected in the process of static malware analysis. We came to the conclusion that all tested models achieved high detection rate. There are summarized results and suggestions for further extensions of this work in conclusion.

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

Algoritmy detekcie malvéru na báze strojového učenia a statickej analýzy kódu

Malware Detection Algorithms Based on Machine Learning and Static Code Analysis

Študent: **Bc. Lukáš Židzik**

Školiteľ: **Ing. Ján Hurtuk, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Analyzovať momentálny stav danej problematiky.
2. Na základe predošlej analýzy vybrať vhodné technológie.
3. Navrhnúť koncept pripravovaného riešenia.
4. Zhodnotiť a otestovať pripravené riešenie.
5. Vypracovať dokumentáciu na základe požiadaviek vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 26.04.2019

Dátum zadania diplomovej práce: 31.10.2018

doc. Ing. Jaroslav Porubän, PhD.
vedúci garantujúceho pracoviska

TECHNICKÁ UNIVERZITA V KOŠICIACH

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 23.4.2019 .....

Vlastnoručný podpis

Poďakovanie

Úprimná vďaka patrí môjmu konzultantovi Ing. Jánovi Hurtukovi, PhD., ktorý si vždy našiel čas a bol ochotný odborne konzultovať a usmerňovať ma počas písania tejto práce.

Poďakovať by som chcel taktiež celej mojej rodine, v ktorej som vždy našiel oporu a stabilitu, čo mi umožnilo naplno sa venovať štúdiu.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Slovník termínov

Softvér je programové vybavenie počítača, súhrn všetkých programov.

Malvér je škodlivý softvér.

Zdrojový kód je postupnosť príkazov napísaných v programovacom jazyku.

Algoritmus je konečná postupnosť definovaných inštrukcií.

Implementácia je realizácia, napísanie kódu pre určitú funkčnosť.

Hash je reťazec, ktorý je jednoznačný identifikátor vstupu.

Zoznam symbolov a skratiek

MD5 Message-Digest Algorithm 5 - kryptografická hašovacia funkcia s dlžkou 128 bitov

SHA-1 Secure Hash Algorithm 1 - kryptografická hašovacia funkcia s dlžkou 160 bitov

URL Uniform Resource Locator -jednotný identifikátor zdroja

PE Portable Executable - súborový formát pre spustiteľné súbory

ASCII American Standard Code for Information Interchange - americký štandardný kód pre výmenu informácií

DLL Dynamic Link Library - dynamicky pripojená knižnica

PDF Portable Document Format - súborový formát dokumentov

API Application programming interface - rozhranie pre programovanie aplikácií

SVC Support Vector Machines - metóda podporných vektorov

KNN K-nearest neighbors - metóda K najbližších susedov

Úvod

Dnešnú dobu môžeme označiť ako dobu informačnú. Je to doba, kde všade okolo nás sú samé informácie a zariadenia pracujúce s nimi. Častejšie vidíme malé deti so smartfónmi v rukách ako s loptou na ihrisku. V každej domácnosti je viacero počítačov, notebookov a rôznych ďalších zariadení pracujúcich s informáciami. Informácií je strašne veľa, no napriek tomu sú veľmi cenné. A práve ich cena z nich robí zaujímavý cieľ aj pre rôznych zločincov, ktorí to chcú zneužiť.

Popri vzniku užitočného softvéru vznikol aj škodlivý softvér nazývaný aj malvér. V súčastnosti rastie počet programov vytvorených za účelom páchať kriminálnu činnosť veľmi rýchlo. Podľa spoločnosti *Kaspersky Lab* [1], ktorá sa zaoberá vývojom antivírusových programov, počet útokov s cieľom ukradnúť peniaze prostredníctvom online prístupu k bankovým účtom bol 1 996 324 a celkovo 4 000 000 nových unikátnych vzoriek malvéru bolo odhalených v roku 2015. Sú to obrovské čísla a spolu s nimi musia rásť aj finančné prostriedky investované do boja s touto počítačovou kriminalitou. Spoločnosť *Juniper Research* [2] predpokladá, že v roku 2019 bude celosvetovo vynaložených 2,1bilióna amerických dolárov na ochranu dát.

Na ochranu proti takýmto praktikám začali vznikať programy, ktoré odhaľujú škodlivý softvér a zabraňujú tak jeho činnosti. Tieto programy však vedia detekovať iba vzorky, ktoré už poznajú a majú v svojej databáze alebo sa správajú podľa nejakého zisteného vzoru. Vzhľadom na veľký počet nových druhov malvéru, ktoré každodenne pribúdajú, takéto riešenie už nie je dostačujúce. Analytici vedia vymyslieť spôsob, akým detekovať vzorky, ktoré sú známe. Tvorcovia malvéru však stále prichádzajú s niečím novým, čo dokáže obísť bezpečnostné programy. Všetko čo človek vymyslí na ochranu sa dá nejako obísť.

A čo keby začal vymýšľať bezpečnostné opatrenia namiesto človeka stroj? V dnešnej dobe je to možné. Umelá inteligencia sa využíva v mnohých systémoch s ktorými denno-denne pracujeme. Strojové učenia, podoblasť umelej inteligencie,

vie spracovať obrovské množstvo informácií, čo človek nedokáže. Vie nájsť rôzne prepojenia, ktoré si človek nedokáže všimnúť.

Práve z týchto dôvodov som sa rozhodol skúmať v tejto diplomovej práci či a ako úspešne by vedel natrénovaný stroj odhaľovať nové vzorky škodlivého softvéru. Cieľom bude vytvoriť z dostupných dát model, ktorý bude vedieť predpovedať, či je neznáma vzorka škodlivý program alebo nie. Dôraz sa bude klásť na čo najvyššiu presnosť predikcie a zároveň na čo najnižší počet tzv. false negative prípadov, kedy by bola škodlivá vzorka označená za bezpečnú.

Touto diplomovou prácou sa budem snažiť zistiť, či využitie strojové učenia bude fungovať aj v tejto oblasti a do akej mieri. V prípade úspešnosti by táto práca mohla byť základom pre vytvorenie účinných antivírusových programov, ktoré by vedeli pracovať aj s neznámymi vzorkami. Chcel by som prispieť k výskumom v oblasti škodlivého softvéru a pomôcť tak ochrániť bežných užívateľov pred stratou dát a iných nežiaducich aktivít.

1 Formulácia úlohy

Cieľom tejto diplomovej práce je pomocou strojového učenia a dát zo statickej analýzy malvéru natrénovať a vytvoriť klasifikačný model, ktorý bude vedieť detekovať škodlivý softvér. Základom bolo podrobne naštudovať a dôkladne analyzovať daný problém. Po tomto prvotnom oboznámení sa s problémom sme zistili, že potrebujeme splniť nasledujúce ciele:

- Oboznámiť sa so statickou analýzou malvéru, jej technikami a nástrojmi, ktoré sa počas nej používajú.
- Analyzovať strojové učenie. Podrobne rozobrať celý proces od získania dát až po overenie úspešnosti vytvoreného modelu. Preskúmať metriky vyhodnotenia úspešnosti modelov.
- Preskúmať existujúce riešenia daného problému a analyzovať ich výhody a nevýhody.
- Zozbierať vzorky čistých a škodlivých súborov a pomocou statickej analýzy z nich extrahovať dôležité atribúty.
- Vybrať vhodný klasifikačný model a natrénovať ho na dátach získaných v predchádzajúcom kroku.
- Overiť úspešnosť jednotlivých natrénovaných klasifikačných modelov a na základe dosiahnutých výsledkov vyvodiť záver.

2 Statická analýza malvéru

Táto kapitola obsahuje vysvetlenie pojmu statická analýza, rozdelenie na základnú a pokročilú a techniky využívané v jednotlivých fázach analýzy. Popísané sú aj nástroje, ktoré napomáhajú analytikom v tejto činnosti.

2.1 Statická analýza

Analýza je jedna zo základných metód a myšlienkových operácií, je to rozklad nejakého celku na jeho zložky, súčasti, ktorý smeruje k nejakým relatívne najjednoduchším zložkám (prvky), za ktoré sa ďalej už nepokračuje [3].

Analýza malvéru je teda rozloženie škodlivého súboru na jeho základné zložky, s cieľom zistiť čo daný malvér vykonáva, ako ho identifikovať a ako sa proti nemu brániť. Existujú dve základné typy malvérovej analýzy, a to statická a dynamická analýza. Základný rozdiel medzi nimi je v tom, že počas dynamickej analýzy sa skúmaná vzorka spúšťa v bezpečnom prostredí narozdiel od statickej analýzy, kde sa skúma prevažne zdrojový kód bez spustenia. Obidva tieto typy sa ďalej delia na základnú a pokročilú analýzu.

Hlavnou úlohou analýzy je preskúmať, čo sa skúmaná vzorka pokúša urobiť, ktoré časti počítačového systému napáda, prípadne či modifikuje, vytvára alebo maže dáta. Ďalšou úlohou je zistiť, ako sa vyhýba bezpečnostným opatreniam, ako sa maskuje a šíri. Záverom každej analýzy by mala byť ucelená správa, v ktorej sú zhrnuté poznatky nadobudnuté počas analýzy a samozrejme aj jednoznačná identifikácia vzorky[4].

Statická analýza je v podstate čítanie zdrojových kódov, z ktorých sa snažíme zistiť správanie skúmanej vzorky[5]. Tento typ analýzy je vo všeobecnosti bezpečnejší ako dynamická analýza, ale má aj svoje nevýhody. Dochádza tu totiž k možnosti prehliadnutia nejakých dôležitých vlastností, najmä pri sofistikovaných

vzorkách. V porovnaní s dynamickou analýzou trvá zvyčajne dlhšie a je náročnejšia. Nespornou výhodou však je, že preskúma všetky možné cesty kódu a nielen jednu, ktorá sa vykoná pri spustení v určitom prostredí. Poskytuje presne informácie o tom, čo autor vzorky chcel dokázať. Nedostupnosťou internetu, výpočtových zdrojov alebo rôznymi zlyhaniami systému sa môže vzorka správať úplne inak ako to autor zamýšľal. Sofistikovaný malvér navyše dokáže rozoznať, či sa ho niekto pokúša analyzovať, či sa spúšťa v bezpečnom prostredí virtuálnych strojov a tým zamení svoje správanie. Týmto docieli, že pri dynamickej analýze nebude detekovaný. Statickú analýzu delíme na základnú a pokročilú.

2.2 Základná statická analýza

Základná statická analýza je zvyčajne prvým krokom pri analýze potenciónalne škodlivého softvéru. Poskytuje informácie ohľadom štruktúry programu, pripojených zdrojov a ďalšie základné informácie o skúmanej vzorke. Tento druh analýzy prebieha pomerne rýchlo a slúži na prvotné oboznámenie sa so vzorkou. Z tejto jednoduchosti vyplýva aj nízka efektivita. Analytik sa ľahko stratí alebo prehliadne dôležité fakty. V procese základnej analýzy sa využívajú viaceré rôzne techniky.

2.2.1 Skúmanie pomocou antivírusových programov

Najjednoduchšou vecou akú môže analytik urobiť je preskúmať podozrivú vzorku pomocou antivírusového programu. Ak antivírus spozná danú vzorku, označí ju za škodlivú a môže poskytnúť aj rozširujúce informácie. Existujú však aj prípady tzv. „false positive“ ked je vzorka označená za škodlivú, pričom škodlivou vôbec nie je. Pre zvýšenie presnosti by sa preto mala vzorka preskúmať viacerými rôznymi antivírusovými programami. Na to slúžia webové služby ako *VirusTotal* [6] alebo *VirSCAN* [7]. Sú to online služby, ktoré zdarma otestujú nahratý súbor na prítomnosť škodlivého softvéru na viacerých antivírusových jadrách. Na záver poskytnú informáciu, či a akými programami bola vzorka detekovaná. Na obrázku 2.1 môžeme vidieť výslednú správu po analýze vzorky službou *VirusTotal*.

Obr. 2.1: Analyzovanie súboru pomocou služby VirusTotal

2.2.2 Použitie „odtlačkov prstov“

Z každej vzorky vieme pomocou určitých programov vypočítať hash – reťazec, ktorý je jedinečný pre každý program a jednoznačne ho identifikuje. Je to niečo podobné ako sú odtlačky prstov u ľudí. Existuju rôzne programy pre výpočet hash-u a takisto niekoľko hash-ovacích algoritmov. Najznámenšie algoritmy sú *Message-Digest Algorithm 5 (MD5)* a *Secure Hash Algorithm 1 (SHA-1)*.

```
C:\md5>MD5DEEP MD5DEEP.TXT
c21c1ef344ef9751fae853ba613290d6  C:\md5\MD5DEEP.TXT
```

Obr. 2.2: Vytvorenie hash-u v príkazovom riadku

Po získaní hash-u ho vieme použiť na vyhľadanie informácií o danom programe na internete, nakoľko jednoznačne identifikuje program. Program s čo i len malými zmenami by už mal iný hash.

2.2.3 Vyhľadávanie reťazcov

Reťazec v programe je postupnosť znakov, ktorá je ukončená špeciálnym znakom tzv. NULL terminátorom. Program používa reťazce :

- ak chce niečo vypísať na výstup (na obrazovku, tlačiareň, …),

- na definovanie cesty ak che modifikovať, mazať, vytvárať alebo kopírovať súbory,
- na definovanie URL - identifikátor adresy pri pripájaní na sieť ďalšie menej podstatné využitia reťazcov.

Tieto reťazce nám môžu veľa napovedať o funkcionalite programu. Z reťazcov vieme vyčítať presné názvy volaných knižníc, funkcií, rôzne výpisy ale aj sieťové adresy v prípade ak sa program pokúša pripojiť sa na sieť.

Vyhľadávanie reťazcov je pomerne ľahký spôsob ako získať veľa informácií o vzorke. To si uvedomujú aj tvorcovia malvéru a preto sa snažia svoje programy zamaskovať. Na takéto maskovanie sa najčastejšie využíva obfuskácia a zbaľovanie. Obfuskácia je technika, pri ktorej sa autor snaží transformovať kód programu s cieľom zmiasť analytika, pričom zachováva pôvodnú funkcionalitu.

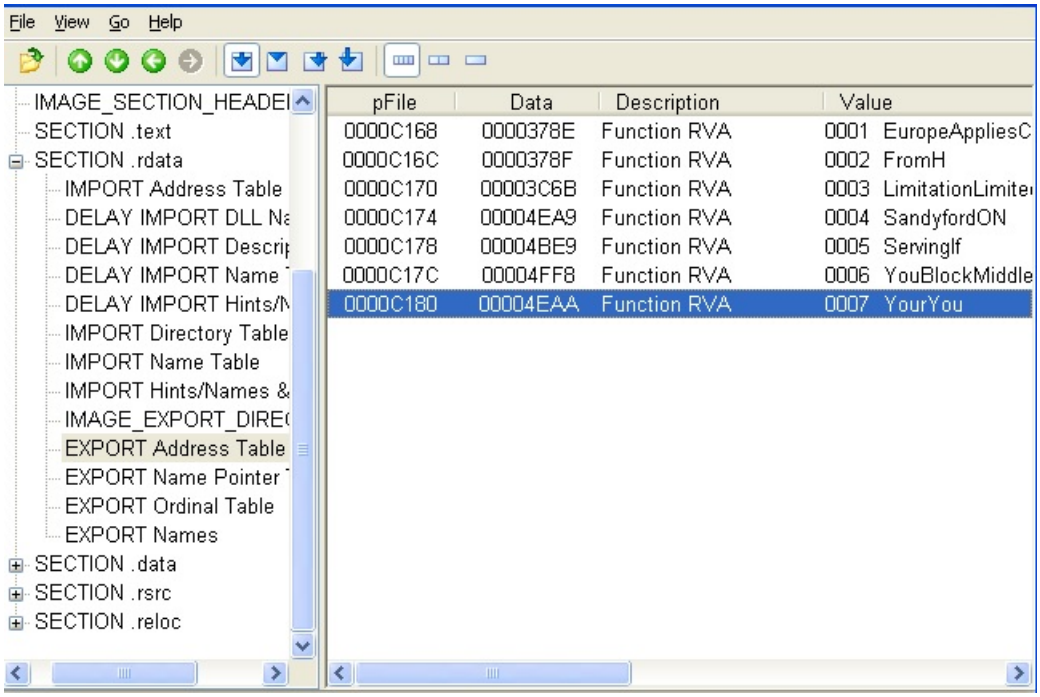
Existuje mnoho druhov obfuskácie no najčastejšie sa využíva dátová obfuskácia. Zbaľovanie programu je účinný spôsob ako rapídne zredukovať počet reťazcov. Je to v podstate kompresia pôvodného súboru, ktorý sa pri spustení rozbalí. Zabalený program je však ťažšie analyzovateľný. Na detekovanie takýchto súborov môžeme využiť PEiD program, ktorý vie zistiť typ nástroja použitého za zbalenie a následne takýto súbor rozbaliť.

2.2.4 Informácie z „inicializačného súboru“

Takmer každý spustiteľný program na Windowse obsahuje informácie v tzv. PE (portable executable) formáte. Je to dátová štruktúra, ktorá obsahuje informácie nevyhnutné pre operačný systém umožňujúce načítanie a spustenie programu [8]. Tento súbor sa skladá z mnohých častí. Najčastejšie sa vyskytujúce sú:

- hlavička – obsahuje informácie o samotnom súbore, kóde, type programu, potrebných knižniciach a nárokoch na pamäť. Tieto informácie vieme získať pomocou nástrojov *Peview*, *PEBrowse Professional* alebo *PE Explorer*.
- .text – obsahuje vykonateľný kód – inštrukcie pre procesor
- .rdata – obsahuje informácie určené iba na čítanie, napr. importované a exportované knižnice
- .data – obsahuje globálne dáta, ku ktorým možno pristupovať odkiaľkoľvek z programu

- .rsrc – obsahuje zdroje, ktoré sú využívané programom (ikonky, obrázky, menu, preklady,...). Na prezeranie týchto súborov môžeme použiť voľne dostupný nástroj Resource Hacker.
- mnohé ďalšie (.edata, .pdata, .reloc)



Obr. 2.3: Analýza súboru pomocou PE View

2.3 Pokročilá statická analýza

Pokročilá statická analýza slúži na hlbšie oboznámenie sa so vzorkou. Stále ide o statickú analýzu, teda dodržuje sa základné pravidlo – vzorka sa nespúšťa. Pokročilá statická analýza je v podstate čítanie kódu so snahou pochopiť ho a zistiť čo autor danej vzorky chcel vykonať. Využíva sa tzv. reverzné inžinierstvo – proces, v ktorom sa analytik snaží zistiť ako skúmaná vec funguje.

2.3.1 Reverzné (spätné) inžinierstvo

Reverzné inžinierstvo je proces získavania poznatkov alebo návrhov plánov z hocičoho, čo človek vytvoril. Tento koncept vznikol dávno pred počítačmi a modernou dobou, pravdepodobne niekedy v období priemyselnej revolúcie. Tento proces je podobný vedeckému výskumu. Rozdiel však je v tom, že pokiaľ

konvenčný výskum sa zaoberá objektami vytvorenými prírodou, reverzné inžinierstvo sa zaoberá objektami vytvorenými človekom. Reverzné inžinierstvo sa zvyčajne zaoberá získavaním informácií, myšlienok, návrhov, ktoré sú nedostupné. V niektorých prípadoch sú tieto informácie vlastnené niekym, kto sa s nimi nechce podeliť [9].

Softvérové reverzné inžinierstvo je o otvorení programu a zistení ako funguje. Toto reverzné inžinierstvo vyžaduje kombináciu zručností a dôkladné pochopenie počítačov a vývoju softvéru. Softvérové reverzné inžinierstvo integruje niekoľko častí: rozbitie kódu, riešenie hádaniek, programovanie a logická analýza.

Reverzné inžinierstvo sa využíva vo veľkej miere aj v škodlivom softvéri. Vývojári škodlivého softvéru často využívajú reverzné inžinierstvo na vyhľadanie zraniteľných miest v operačných systémoch a inom softvéri. Takéto zraniteľné miesta potom využívajú na preniknutie do systémov, obchádzanie bezpečnostných opatrní, získavanie citlivých informácií alebo získaním úplnej kontroly nad napadnutým systémom. Aj autori antivírusových systémov často využívajú reverzné inžinierstvo. Používajú rôzne reverzné techniky na sledovanie každého kroku, ktorý program vykonal, na posúdenie škôd, ktoré by mohol škodlivý softvér spôsobiť, na spôsob odstránenia z napadnutého systému a zabráneniu infekcii.

Autori malvéru využívajú prevažne jazyky vyššej úrovne na tvorbu ich programov. Tento zdrojový kód je však pomocou kompilátora transformovaný do strojového jazyka, ktorý vie procesor spracovať a vykonať. Malvér sa šíri výhradne v tejto forme a preto sú zdrojové kódy malvéru nedostupné. Strojový jazyk je pre človeka veľmi náročný, ba až nezrozumiteľný a z toho vyplýva potreba preložiť ho spätne do jazyka vyššej úrovne. Bohužiaľ neexistuje opačný proces, k procesu kompilácie a to znamená, že spätne vytvoriť zdrojový kód v jazyku, v ktorom bol vytvorený je nemožné. Pomocou nástrojov nazývaných disassemlery je však možné zo strojového kódu vytvoriť kód v jazyku assembler. Najznámejším disassemblerom je *IDA Pro*.

2.3.2 Assembler

Assembler je jazyk nižšej úrovne, no je pre človeka oveľa jednoduchší ako strojový kód. Je to jazyk najvyššej možnej úrovne, ktorý sa dá vytvoriť zo strojového jazyku. Skladá sa zo súboru jednoduchých operácií ako sú napríklad jmp(operácia skoku), mov(operácia presunutia) a iné. Existuje viacero verzií assemblera, každá z nich je určená pre určité typy mikroprocesorov. Najrozšírenejšou je x86, na kto-

rej sú postavené všetky moderné 32-bitové počítače s Windows-om. Navyše najznámejší výrobcovia procesorov (Intel aj AMD) podporujú architektúru x86 aj na 64-bitových počítačoch. Táto pozícia na trhu predurčuje aj verziu assembleru, do ktorej sa prekladajú vzorky strojového kódu, a preto sa v nasledujúcich častiach tejto práce zameriame na inštrukcie v tejto architektúre.

```
00000000        push    ebp
00000001        mov     ebp, esp
00000003        movzx   ecx, [ebp+arg_0]
00000007        pop     ebp
00000008        movzx   dx, cl
0000000C        lea     eax, [edx+edx]
0000000F        add     eax, edx
00000011        shl     eax, 2
00000014        add     eax, edx
00000016        shr     eax, 8
00000019        sub     cl, al
0000001B        shr     cl, 1
0000001D        add     al, cl
0000001F        shr     al, 5
00000022        movzx   eax, al
00000025        retn
```

Obr. 2.4: Ukážka zdrojového kódu v jazyku Assembler

2.3.3 Inštrukcie

Inštrukcia je základná stavebná jednotka, z ktorej sú zložené celé programy. Pozostáva z názvu inštrukcie a niekoľkých operandov. Názov inštrukcie hovorí, čo má procesor vykonať a operandy slúžia na identifikáciu dát nad ktorými sa má operácia vykonať. Inštrukcie sú napríklad:

- mov – presun dát z jedneho miesta na iné, formát inštrukcie je „mov cieľ, zdroj“
- lea – podobná inštrukcia ako mov s rovnakým formátom, ale namiesto dát presúva adresu pamäte
- add – pripočítanie hodnoty k hodnote cieľu, formát inštrukcie je „add cieľ, hodnota“
- sub – odčítanie hodnoty od hodnoty cieľu, formát inštrukcie je „sub cieľ, hodnota“

Typy operandov:

- konštanty napr. 0x42
- registre – odkaz na register napr. ecx
- miesto v pamäti, kde sa nachádzajú dáta [eax]

Po preložení programu do jazyka assembler nasleduje samotná analýza. Analytik postupne číta jednotlivé inštrukcie a snaží sa pochopiť ako celý program funguje. Pre takého pochopenie musí poznať základné jazykové konštrukcie akými sú priradenie, aritmetické operácie, vetvenie, cykly a volanie funkcií.

2.3.4 Výstup analýzy

Po naštudovaní a pochopení kódu analytik vyvodí záver. Záverom malvérovej analýzy je správa. V tejto správe musí byť jednoznačne určená vzorka, ktorá bola skúmaná (najčastejšie pomocou hash funkcie). V tejto správe je jasne uvedené, čo bolo skúmané, aké metódy sa v procese analýzy použili a aké závery boli vyvodené. Správa by mala obsahovať čo najviac nadobudnutých informácií a jednoznačným určením, či sa jedná o nebezpečný program alebo nie. V prípade zistenia, že ide o malvér správa obsahuje aj informácie o spôsobe infikácie, rezistencii, šírení sa a funkcionalite danej vzorky.

2.4 Nástroje statickej analýzy

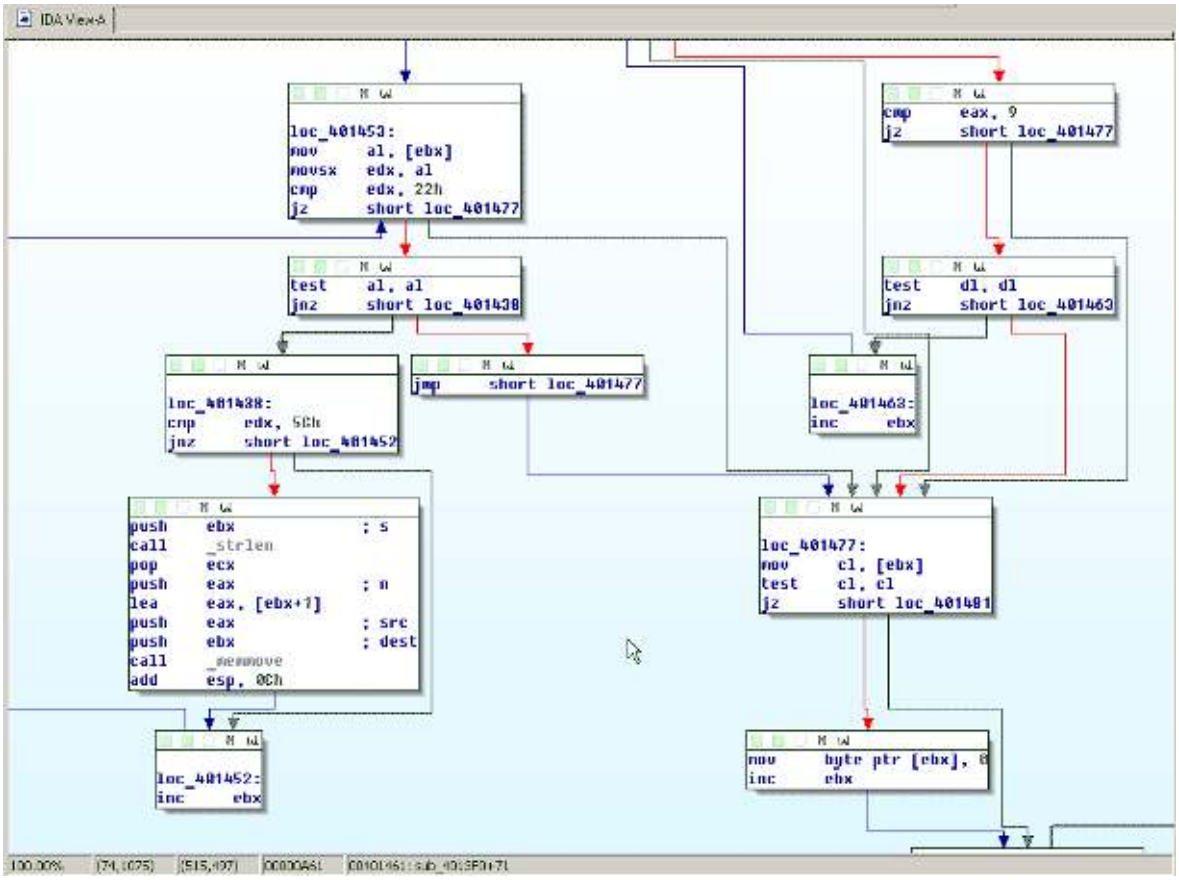
V procese statickej analýzy je možné používať veľa rôznych nástrojov, ktoré uľahčujú analytikom prácu. Mnohé z nich, ako napríklad *Dependency Walker*, neboli vôbec vyvinuté pre túto činnosť, no vzhľadom na ich služby sú využívané. V tejto práci spomeniem najznámejší dissasembler *IDA Pro* a takisto nemenej známy program *Depencency Walker*.

2.4.1 IDA Pro

IDA Pro (Interactive Disassembler Professional) je mimoriadne výkonný disassembler distribuovaný spoločnosťou *HexRays*. Hoci *IDA Pro* nie je jediným disassemblerom, je to jednoznačne najpoužívanejší disassembler pre analytikov malvéru, reverzných inžinierov a analytikov bezpečnosti. Tento nástroj podporuje niekoľko formátov súborov, ako napríklad *PE (Portable Executable), COFF (Common Object File Format), ELF(Executable and Linking Format)* a iné. Dokáže rozložiť celý program a vykonať úlohy, ako napríklad zisťovanie funkcií, analýza zásobníkov či identifikácia lokálnych premenných. Obsahuje taktiež rozsiahle podpisy vrámci technológie *FLIRT (Fast Library Identification and Recognition Technology)*, ktorá umožňuje rozpoznať a označiť kód pridaný kompilátorom z rôznych knižníc. Veľkou

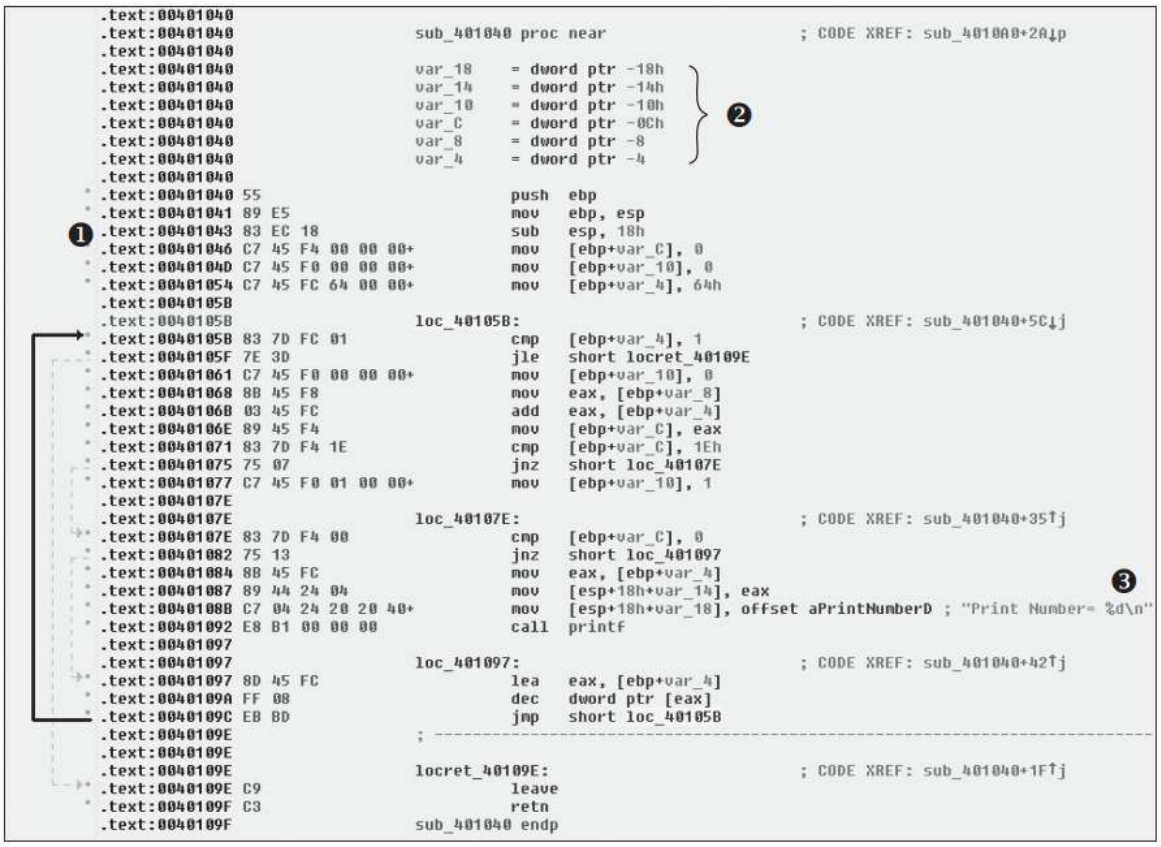
výhodou je taktiež možnosť uložiť si svoju prácu a pokračovať v nej neskôr. *IDA Pro* obsahuje množstvo rozšírení, s možnosťou doplniť si aj vlastné rozšírenia. Po načítaní skúmaného súboru sa zisťujú základné informácie ako je formát vzorky, architektúru systému kde bola vzorka vytvorená či verzia kompilátora. *IDA Pro* obsahuje dva základné módy a to textový a grafový.

1. Grafový mód zobrazuje skúmanú funkciu rozloženú do uzlov, ktoré predstavujú základne bloky kódu v assembleri a ktoré sú navzájom prepojené farebnými šípkami. Šípky znázorňujú tok programu. Aj farba šípok je dôležitá, červená znázorňuje podmienený skok, zelená tok programu ak je skok uskutočnený a modrá nepodmienený skok. Šípky smerujúce nahor označujú cyklus. V tomto móde nie sú zobrazené žiadne dáta.



Obr. 2.5: Grafový mód nástroja IDA Pro

2. Textový mód slúži na prezeranie dátových oblastí vzorky. V ľavej časti je okno so šípkami, ktoré zobrazuje nelineárny tok programu. Plné čiary označujú nepodmienené skoky, prerušované čiary označujú podmienené skoky a šípky smerujúce nahor označujú cyklus. V pravej časti sa zobrazujú komentáre, ktoré generuje *IDA Pro* a pomáhajú pri analýze.



Obr. 2.6: Textový mód nástroja IDA Pro

IDA Pro obsahuje viacero okien, ktoré zobrazujú rôzne informácie o skúmanej vzorke. Najčastejšie používané okná pre analýzu škodlivého softvéru sú:

- Okno s funkciami – zobrazuje zoznam všetkých funkcií v skúmanej vzorke a ich veľkosť. Na základe toho si analytik môže utriediť funkcie podľa veľkosti a zamerať sa na väčšia a zložitejšie z nich, ktoré sú dôležitejšie pri analýze.
- Okno s menami – zobrazuje zoznam všetkých použitých názvov s ich adresou, napríklad názvy funkcií, dát alebo reťazce.
- Okno s reťazcami – zobrazuje všetky nájdené reťazce (preddefinovaná minimálna dĺžka je 5 znakov)
- Okno importovaných funkcií – zoznam všetkých funkcií, ktoré skúmaná vzorka používa
- Okno exportovaných funkcií – zoznam všetkých funkcií, ktoré skúmaná vzorka obsahuje (využíva sa najmä pri analýze DLL)

2.4.2 Dependency Walker

Dependency Walker [10] je voľne dostupný program vyvinutý pre platformu Microsoft Windows, ktorý slúži na analýzu spustiteľných súborov. Funguje na verziách Windows-u (Windows 95 až Windows 10). Zobrazuje zoznam všetkých modulov na ktorých je závislý skúmaný program v hierarchickej stromovej štruktúre a takisto zobrazuje zoznam importovaných a exportovaných názvov funkcií pre jednotlivé moduly. Poskytuje podrobné informácie o všetkých moduloch ako napríklad čas vytvorenie a úpravy, typ procesora kde bol vytvorený, kontrolné súčty, veľkosť, číslo verzie a iné. Kôli kontrole všetkých závislostí sa často používa aj na zistenia chýbajúcich modulov. Hlavná obrazovka sa delí 5 častí :

1. Zobrazenie stromu závislosti modulu
2. Zoznam všetkých modulov s podrobnosťami
3. Zoznam importovaných funkcií
4. Zoznam exportovaných funkcií
5. Log – informácie ohľadom problémov a upozornení v daných moduloch a ich závislostiach

Obr. 2.7: Ukážka hlavnej obrazovky program Dependency Walker

3 Strojové učenie

V tejto kapitole sa bližšie oboznámime s pojmom strojové učenie. Detailne si opíšeme jeho využitie, druhy a najviac využívané algoritmy. Pozrieme sa aj na vyhodnocovanie úspešnosti natrénovaných modelov z pohľadu rôznych metrík.

3.1 Čo je strojové učenie

Strojové učenie je podoblasťou umelej inteligencie, zaoberajúca sa algoritmami a technikami, ktoré umožňujú počítačovému systému „učiť sa“. Učiť sa znamená zlepšiť svoj výkon bez potreby programovania ďalšej funkcionality. Algoritmy sú založené na vytváraní modelu zo vstupných dát, pomocou ktorého predpovedajú výsledky a rozhodujú sa. Tom M. Mitchell [11] definoval strojové učenie ako: „Hovorí sa, že počítačový program sa učí zo skúsenosti E v súvislosti s určitou triedou úloh T a meraním výkonu P, ak sa jeho výkon pri úlohách v T, meraný P, zlepšuje so skúsenosťami E.“ Strojové učenie je v podstate vytváranie modelov zo vstupných dát a ich používanie na predpovedanie. Predpovedať môžeme hocičo, cenu bytov, výsledky športových podujatí, výskyt choroby u človeka alebo zvierat a mnoho ďalšieho. Strojové učenie sa prelína s oblasťami ako je štatistika alebo dolovanie znalosti. Ma širokú škálu využiteľnosti, ktorá sa každým dňom zväčšuje.

3.2 Typy strojového učenia

Existuje viacero prístupov k strojovému učeniu. Každý z týchto typov má k dispozícií iný typ dát a inak ich spracováva. Medzi základné prístupy k strojovému učeniu patria: učenie s učiteľom, učenie bez učiteľa, kombinácia učenia s a bez učiteľa a učenie zo spätnej väzby.

3.2.1 Učenie s učiteľom (supervised learning)

Učenie s učiteľom je prístup, kedy sú k dispozícií vstupné dáta X a aj výstupné dáta Y. Stroj sa pomocou rôznych algoritmov a vstupných dát snaží naučiť mapovaciu funkciu f, ktorá bude mapovať vstup na výstup $Y = f(X)$. Cieľom je vytvoriť funkciu mapovania tak dobre, že keď máme nové vstupné dáta X, tak môžeme predpovedať výstupné premenné Y pre tieto dáta [12]. Názov učenie s učiteľom vznikol, pretože tento proces môžeme prirovnať k výučbe, kde žiak rieši iteratívne úlohy a učiteľ, ktorý pozná správne riešenia ho opravuje v prípade chybného riešenia. Tento proces sa opakuje dovtedy, pokiaľ sa žiak nenaučí riešiť danú úlohu.

Väčšina problémov, ktoré využívajú strojové učenie využíva práve učenie s učiteľom. Tieto problémy môžme rozdeliť do dvoch hlavných skupín:

1. Klasifikácia - výstupná premenná je kategorická hodnota, napr. biela, červená, pravda, nepravda, ...

   Najznámejšie algoritmy učenia s učiteľom v klasifikačných úlohach sú:

   - *Náhodné lesy*
   - *Metóda podporných vektorov*

2. Regresia - výstupná premenná je reálne číslo, napr. hmotnosť, cena, ...

   Najznámejšie algoritmy učenia s učiteľom v regresných úlohach sú:

   - *Lineárna regresia*
   - *Náhodné lesy*

3.2.2 Učenie bez učiteľa (unsupervised learning)

Učenie bez učiteľa je prístup, kedy sú dostupné iba vstupné dáta. Názov je odvodený od toho, že žiak (stroj) nemá k dispozícií učiteľa, ktorý by mu povedal správne výsledky [13]. Cieľom je vytvoriť si vlastnú štruktúru vo vzorke vstupných dát.

Problémy učenia bez učiteľa môžme rozdeliť na dve hlavné skupiny:

1. Zhlukovanie - objavenie skupín vzoriek, ktoré majú podobné správanie (parametre), napr. skupina zákazníkov, ktorý kupujú rovnaké výrobky Medzi najznámejší algoritmus zhlukovacích problémov patrí *K-najbližších susedov*.

2. Združovanie (asociácia) - objavovanie vzájomných vzťahov a pravidiel medzi dátami, napr. ak zákazník kupuje produkt X (mlieko), kupuje aj produkt Y (chlieb) Najznámejším algoritmom asociačných problémov je *Apriori algoritmus*.

3.2.3 Kombinácia učenia bez učiteľa a s učiteľom (semi-supervised learning)

Kombináciu predošlých dvoch prístupov vzniká tento tretí prístup. V tomto prístupe je dostupné veľké množstvo vstupných dát X, ale iba niektoré z nich majú aj výstupné dáta Y [14]. Množstvo reálnych problém spadá práve do tohto typu. Dôvodom chýbajúcich výstupných hodnôt je cena, časová náročnosť alebo potreba expertov v danej oblasti. V tomto prístupe je možné využiť techniky učenia bez učiteľa na zistenie vzťahov medzi dátami a techniky učenia s učiteľom na predikciu dát bez výstupných hôdnot. Takto získané hodnoty sa môžu ďalej používať na lepšie natrénovanie modelu.

3.2.4 Učenie posilňovaním (reinforcement learning)

Učenie posilňovaním je typom strojového učenia, kde sa agenti učia ako sa správať vykonávaním akcií vo svojom prostredí a pozorovaním ich výsledkov [15]. Informácie (pochvala alebo trest) sú poskytnuté ako spätná väzba za strojom vykonanú akciu v dynamickom prostredí. Na základe vykonanej akcie a spätnej väzby sa stroj naučí ako sa má správať v rôznych situáciách. Tento druh strojového učenia našiel využitie najmä v počítačových hrách, kde sa stroje využívajú ako protihráči v hrách pre viacero hráčov. Silu a úspešnosť tohto prístupu demonštruje *OpenAI Five* (systém piatich neurónových sietí), ktorý porazil v hre *DOTA 2* najlepších hráčov tejto hry z celého sveta [16]. Učenie zo spätnej väzby našlo uplatnenie aj v ďalších oblastiach ako sú:

- samojazdiace autá,
- roboty,
- odporúčania a automatické dopĺňanie,
- reklamy a marketing.

3.3 Využitie strojového učenia

Využitie strojového učenia je naozaj široké a uplatňuje sa vo všetkých odvetviach či už je to poľnohospodárstvo, zdravotníctvo alebo finančný sektor. Strojové učenie je aplikované v mnohých zariadeniach a prístrojoch okolo nás, ktoré denno-denne používame, bez toho aby sme o tom niečo vedeli. Prestížny americký magazín *Forbes* [17] uvádza ako najlepšie využitie strojového učenia a umelej inteligencie nasledujúcich desať oblastí:

1. Počítačová bezpečnosť – v roku 2014 sa vo svete objavovalo denne 325 tisíc nových vzoriek škodlivého softvéru [18]. Tento nový škodlivý softvér sa však svojím kódom líšil iba v dvoch až desiatich percentách. Na základe rôznych spoločných častí a vzorov vie naučený systém takého vzorky detekovať s vysokou presnosťou.

2. Osobná bezpečnosť – naučený systém vie minimalizovať čas pri bezpečnostných kontrolách na letiskách a ďalších citlivých miestach a všimnúť si detaily, ktoré ľudský faktor prehliadne. Týmto sa zabezpečí aj väčšia bezpečnosť.

3. Obchodovanie – mnohí ľudia sa snažia predpovedať ceny komodít na svetových trhoch a obchodovať s vidinou vysokého zisku. Moderné spoločnosti však už používajú systémy so strojovým učením, ktoré sa vedia lepšie orientovať v množstve informácií o cenách a objemoch komodít a teda aj spoľahlivejšie predpovedať výhodné obchody.

4. Zdravotníctvo – veľké množstvo informácií získaných pri rôznych vyšetreniach vie naučený systém lepšie spracovať ako človek. Počítačom riadená diagnostika dokáže oveľa skôr objaviť prvé štádia rakoviny. Štúdiou mamografických vyšetrení žien, ktoré mali rakovinu prsníka pomocou strojového učenia sa zistilo, že počítač odhalil až 52 % prípadov viac ako rok pred oficiálnym diagnostikovaním.

5. Reklama – zobrazovať niekomu reklamy produktov o ktoré absolútne nemá záujem je zbytočné, aj tak si to nekúpi. Pomocou strojového učenia sa však získané informácie o potencionálnom zákazníkovi transformujú do cielenej reklamy, ktorá znásobuje možnosť predaja. A získať tieto informácie v dnešnej dobe nie je až tak ťažké, stačí pozrieť čo daný človek vyhľadáva na internete, aké ma záľuby a kde sa pohybuje.

6. Detekcia podvodov – strojové učenie sa stáva čoraz spoľahlivejším nástrojom

na odhalenie rôznych podvodných transakcií. Napríklad spoločnosť PayPal disponuje nástrojmi, ktoré porovnávajú milióny transakcií a odhaľujú pranie špinavých peňazí.

7. Odporúčania – dopĺňanie slov pri písaní do vyhľadávača, vybrané filmy, videá alebo hudba, to všetko nie je náhoda ale výsledok komplexného systému, ktorý porovnáva správanie ľudí na internete a snaží sa im poskytnúť to čo potrebujú.

8. Vyhľadávanie – informácie o tom, kde ste klikli, ktorú stránku ste si zobrazili alebo to, že ste začali nové vyhľadávanie sú vstupmi pre systém so strojovým učením, ktorý sa neustále zlepšuje a snaží sa nájsť presne to čo hľadáte. Nové vyhľadávanie bez otvorenia nájdených stránok indikuje to, že vyhľadávanie nebolo vhodné a je potrebné zobraziť iné výsledky.

9. Spracovanie prirodzeného jazyka – ovládanie rôznych prístrojov a zariadení hovoreným slovom namiesto používania klávesnice, myši alebo iných tlačidiel.

10. Inteligentné autá – pomocou rozpoznávania okolia a naučeného systému Vás auto odvezie presne tak, kde chcete. Samozrejmosťou je nájdenie ideálnej trasy s ohľadom na dopravnú situáciu, čas a pohonné hmoty. Navyše nastaví teplotu, zapne rádio, nastaví sedadlo a to všetko podľa naučených zvykov vodiča.

Ďalším zaujímavým využitím strojového učenia je *Yara N-Sensor ALS* systém [19], ktorý zaznamenáva odraz svetla plodín a na základe toho vypočítava odporúčané množstvo hnojiva pre dané plodiny. Pomocou tejto technológie sa zvýšili výnosy obilnín o 3,5%, pričom bolo ušetrených 14% dusíka. Nemôžeme zabudnúť ani na rozpoznávanie tvári, identifikáciu sa hlasom či predpoveď počasia. Strojové učenie sa dá uplatniť takmer všade, kde je dostatok informácií. A práve tu je rozdiel medzi strojom a človekom. Stroj vie oveľa lepšie spracovávať veľké množstvá informácií a využiť ich vo svoj prospech.

3.4 Proces strojového učenia

Proces strojového učenia môžeme rozdeliť na niekoľko základných fáz. Cieľom tohto procesu je vyvinúť taký model, ktorý bude s čo najväčšou presnosťou predikovať výstup na základe vstupu. Pre dosiahnutie cieľa sa musia vykonať

všetky fázy.

3.4.1 Zozbieranie dát

Zozbieranie dát je prvým krokov k vývoju predikčného modelu. Nájsť zopár dát z riešenej domény nie je problém, no zozbierať dostatočne veľké množstvo je časovo i cenovo náročný proces. Tento krok je veľmi dôležitý, pretože kvalita a množstvo zozbieraných dát priamo ovplyvňuje úspešnosť vyvinutého modelu. Kvalita modelu stúpa so stúpajúcim množstvom dát. Aj to má však svoju cenu a tou je pamäťová náročnosť a výkonnosť. Dôležité je, aby dáta boli vyvážené, teda aby obsahovali dostatok vzoriek zo všetkých dostupných tried. Zozbierané dáta by mali pokryť všetky možné vstupy, nie iba tie najčastejšie.

V tomto kroku je ťažké povedať, aké dáta budeme skutočne potrebovať, preto je dôležité zozbierať všetky dostupné informácie a uložiť ich surovom stave. S dnešnými možnosťami nie je žiaden problém ukladať si veľké množstvo dát. Dáta predstavujú čokoľvek čo obsahuje informáciu, napríklad knihy, obrázky, zvukové nahrávky, fotky, rôzne texty a záznamy. Získavanie dát môže prebiehať rôznym spôsobom. Najväčším úložiskom dát je samozrejme internet, kde môžeme nájsť veľa potrebných dát. Ďalšou možnosťou sú firmy, ktoré sa špecializujú práve na túto činnosť a vedia zozbierať potrebné dáta. Dáta môžu byť rôznych typov:

- nominálne,
- ordinálne,
- numerické,
- spojité.

Najlepšou štruktúrou dát pre použitie v strojovom učení je dvojrozmerná matica, kde riadky reprezentujú jednotlivé vzorky a stĺpce ich vlastnosti, príznaky, atribúty.

3.4.2 Predspracovanie dát

Predspracovanie dát je fáza, v ktorej zo zozbieraných dát potrebujeme vytvoriť dáta vhodné na ďalšie použitie a vytvorenie modelu. Zozbierané dáta sú často ťažko spracovateľné pre počítačový systém. Najlepšou štruktúrou dát pre

použitie v strojovom učení je dvojrozmerná matica, kde riadky reprezentujú jednotlivé vzorky a stĺpce ich vlastnosti, príznaky a atribúty. Obrázky sa transformujú na množinu pixlov, nahrávky na zoznam zvukových frekvencií, atď. V tomto kroku je dobré vizualizovať si zozbierané dáta, medzi ktorými takto môžeme nájsť súvislosti a zároveň aj skontrolovať vyváženosť dát.

V zozbieraných dátach je všetko čo sme našli. Častokrát aj to, čo by sme tam nechceli a preto je potrebné dátovú sadu vhodným spôsobom predspracovať:

- Chýbajúce dáta – nie všetky vzorky musia obsahovať všetky atribúty. Pre natrénovanie modelu však potrebujeme úplné vzorky. Existuje viacero riešení tohto problému. Každé z nich však má aj svoje nevýhody. Najjednoduchším riešením je takého vzorky odstrániť. Odstránením vzoriek môžeme stratiť množstvo cenných informácií obsiahnutých v dátach. Častejšie sa využíva imputácia chýbajúcej hodnoty. Táto hodnota sa však líši od skutočnej hodnoty a preto môžu byť výsledky skreslené. Na nahrádzanie chýbajúcich hodnôt sa používa priemer daného atribútu alebo najčastejšie vyskytujúca sa hodnota v danom atribúte.
- Nezriedka sú zozbierané dáta vo forme kategorických hodnôt. Takého hodnoty treba namapovať na numerické hodnoty (červená = 1, zelená = 2, modrá = 3...), pretože väčšina modelov dokáže pracovať iba s numerickými hodnotami. Toto mapovanie však nie je vždy dostačujúce. Dôvodom je, že takto namapované kategorické hodnoty nevieme porovnávať (1 < 2 ale je červená menej ako zelená ?), utriediť ich alebo vypočítať priemer (priemer červenej a modrej = (1+3)/2 = 2 = zelená ?), čo by v predikčných modeloch spôsobilo chyby. V takýchto prípadoch sa využíva metóda *One hot encoding*, kde sa vytvorí tabuľka s binárnymi hodnotami. Príklad takejto tabuľky je 3.1, kde je zobrazené mapovanie farieb pomocou spomínanej metódy.

farba	numericke mapovanie	one hot encoding		
červená	1	1	0	0
zelená	2	0	1	0
modrá	3	0	0	1

Tabuľka 3.1: Mapovanie kategorických hodnôt na numerické

- V zozbieraných dátach sa vyskytujú aj odľahlé hodnoty, ktoré vznikajú raritnými situáciami a sú ojedinelé. Takéto hodnoty nám však môžu výrazne

vplývať na výsledný model. Preto je lepšie takéto hodnoty detekovať a odstrániť ich.

- Niekedy sú zozbierané dáta vo veľkých rozsahoch, čo nie je dobré pre výkonnosť trénovacieho algoritmu. Takéto dáta vieme upraviť pomocou škálovania tak, že zmenšíme ich rozsah no nezmeníme ich význam vzhľadom k ostatným dátam. Na škálovanie sa najčastejšie využíva normalizácia a štandardizácia, ktoré sú zobrazené v tabuľke 3.2

Vstup	**Štandardizovaná hodnota**	**Normalizovaná hodnota**
0.0	-1.336306	0.0
1.0	-0.801784	0.2
2.0	-0.267261	0.4
3.0	0.267261	0.6
4.0	0.801784	0.8
5.0	1.336306	1.0

Tabuľka 3.2: Porovnanie štandardizácie a normalizácie

Ďalej je potrebné dáta rozdeliť na trénovacie a testovacie. Trénovacie dáta sa budú používať na „výučbu“ modelu. Ak bude náš model dostatočne vytrénovaný otestuje sa pomocou testovacích dát, ktoré neboli použité v procese trénovania. Najčastejšie sa trénovacie a testovacie dáta rozdeľujú v pomeroch 80:20 alebo 90:10, samozrejme aj tento pomer závisí od veľkosti získaných dát.

3.4.3 Výber algoritmu strojového učenia

Vybrať si ten správny algoritmus je taktiež náročné. Každý z algoritmov má svoje výhody aj nevýhody no v tomto kroku je potrebné sa rozhodnúť aký algoritmus bude použitý vzhľadom na získané dáta. Rokmi výskumov bolo vyvinuté veľké množstvo algoritmov pre rôzne dáta. Niektoré pracujú lepšie s numerickými dátami, iné sú vhodné pre prácu s textom alebo so spojitými hodnotami. Algoritmy delíme do troch základných skupín, podľa druhu úlohy ktorý budu vykonávať a to na:

- klasifikačné,
- regresné,
- zhlukovacie (klastrové).

Je možné vybrať si viacero algoritmov a rozhodnúť sa na základe získaných výsledkov. Tento postup zvyšuje časovú náročnosť, no je často využívaný v praxi.

3.4.4 Trénovanie

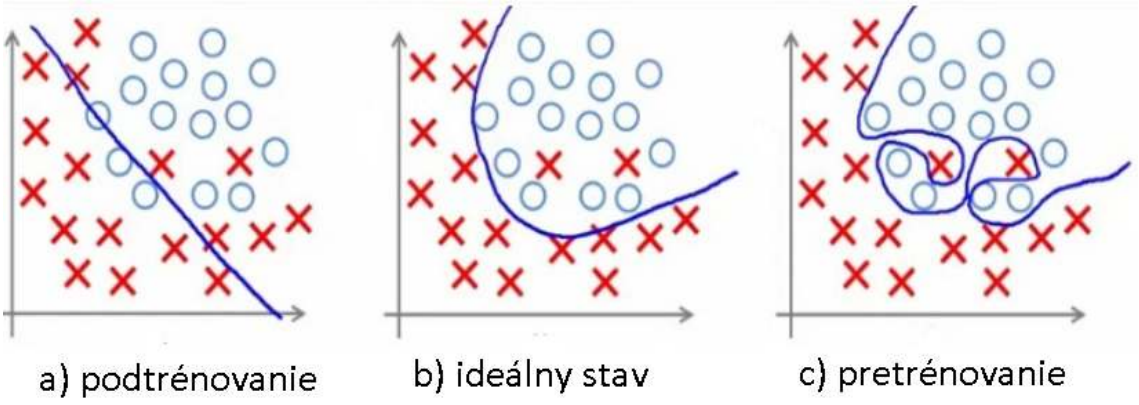
Vo fáze trénovania sa zúčastňujú iba trénovacie dáta. Počas trénovania sa algoritmus snaží naučiť čo najlepšie predikovať výstup. Trénovanie prebieha v iteráciách, v ktorých by sa úspešnosť algoritmu mala zlepšovať. V prvej iterácií sú parametre algoritmu buď náhodne zadané alebo majú svoje inicializačné hodnoty. Pomocou týchto parametrov sa predikujú výsledky. Tieto sú potom porovnávané so skutočnými hodnotami a na základe toho sú upravené parametre. Nasledujú ďalšie iterácie, kde sa robí predikcia s upravenými parametrami a následne sa opäť upravujú podľa najnovších výsledkov predikcie. Tento proces sa cyklicky opakuje až do momentu, kedy sa natrénovaný model už nedá vylepšiť, teda presnosť predpovede sa už nezlepšuje [20]. Najčastejšími problémami pri trénovaní modelu sú podtrénovanie (underfitting) a pretrénovanie (overfitting).

Podtrénovanie je stav, keď algoritmus nedokáže dobre predpovedať ani dáta na ktorých bol natrénovaný, teda ich pozná. Predpoveď na neznámych dátach dosahuje veľmi nízku úspešnosť. Model je príliš jednoduchý. Príčinou môže byť zle zvolený algoritmus, ktorý sa nedokáže naučiť na danej vzorke dát. V takomto prípade sa odporúča vyskúšať nejaký iný algoritmus.

Pretrénovanie je stav, keď je model veľmi spätý a naviazaný na trénovacie dáta. Predpoveď na týchto dátach vykazuje veľmi vysokú úspešnosť, no predikcia na neznámych dátach ma naopak nízku úspešnosť. Vyhnúť sa tomuto problému môžeme rozdelením získaných vzoriek na niekoľko skupín, z ktorých sa niektoré budú využívať na trénovanie a ostatné na testovanie a následnú korekciu parametrov.

3.4.5 Ladenie parametrov

Každý model má svoje parametre, na ktorých závisí výsledná presnosť. Existujú dva druhy parametrov – parametre naučené z dát a parametre nastaviteľné tzv. hyperparametre. Nastaviteľné parametre definujú rôzne vlastnosti algoritmu napríklad maximálnu dovolenú hĺbku pri rozhodovacích stromoch alebo počet vrstiev v neurónovej sieti. Pri vytváraní modelu je potrebné zadať tieto nastaviteľné parametre. Najčastejšie sú to prednastavené hodnoty. Zmenou týchto

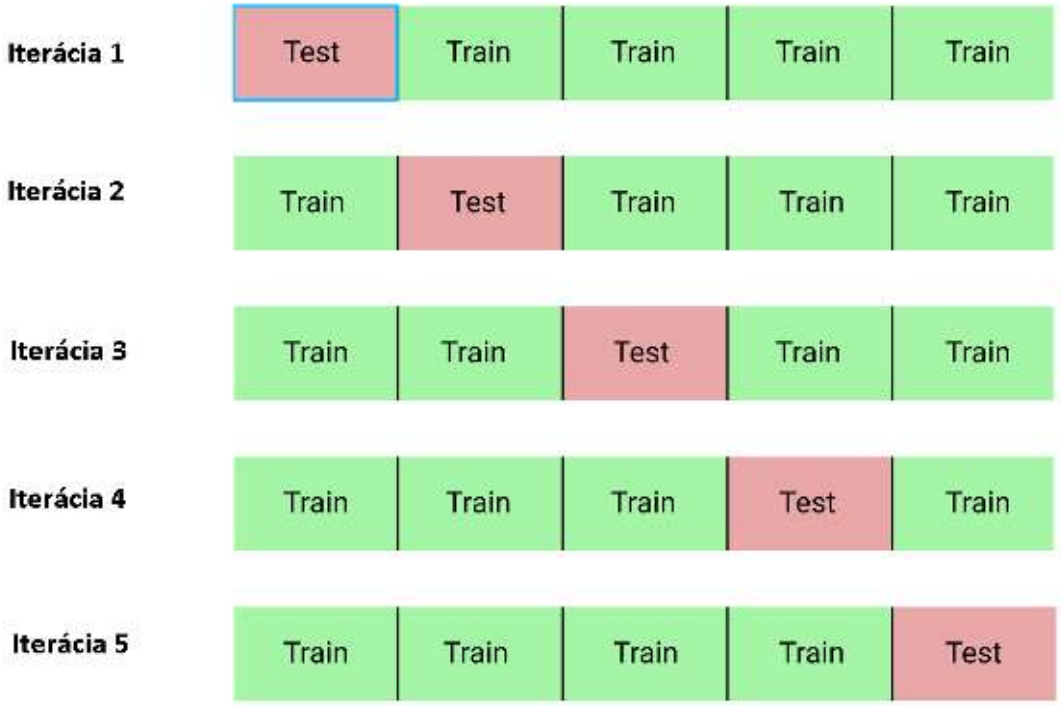


Obr. 3.1: Podtrénovanie a pretrénovanie

parametrov však vieme dosiahnuť aj vyššiu úspešnosť modelu. Preto sa táto fáza nazýva aj ladenie parametrov. Potrebujeme nájsť také parametre, pri ktorých bude náš model dosahovať najlepšie výsledky.

Bohužiaľ neexistujú žiadne všeobecné pravidlá, podľa ktorých by sa tieto parametre mali nastaviť. Jediným možným spôsobom je experimentovanie. Najčastejšou metódu pri vyhľadávaní optimálnych parametrov je mriežkové vyhľadávanie (grid search). Je to tzv. „brutte force“ prístup, kedy sa vyskúšajú všetky možné kombinácie parametrov a nájde sa taká, pri ktorej model dosahuje najlepšie výsledky. Tejto fázy sa zúčastňujú iba trénovacie dáta. Testovacie dáta sú použité až v úplnom závere (po nájdení optimálnych parametrov), aby presnosť modelu na týchto dátach zobrazovala reálnu schopnosť modelu predikovať výsledky.

Úspešnosť modelu sa zisťuje pomocou validácie. Existuje viacero metód validácie modelu. V súčastnosti sa najčastejšie využíva metóda k-zložkovej krížovej validácie (k-fold cross-validation). Princíp tejto metódy je jednoduchý. Rozložíme si zozbierané dáta na „k“ rovnakých častí. Potom „k“-krát vykonáme trénovanie a testovanie, kde trénovacie dáta obsahujú k-1 častí a testovacie dáta obsahujú jednu časť pôvodných dát. Táto testovacia časť sa stále mení a tým je zaručené, že všetky vzorky budú práve raz v testovacej časti. Takto získame „k“ výsledkov presnosti modelu, z ktorých je urobený priemer. Tento priemer určuje presnosť modelu s minimalizovanou závislosťou na dátach. Výsledky presnosti modelu získavame pomocou rôznych metrík, ktoré sú bližšie popísané v 3.6.1.



Obr. 3.2: Rozdelenie dát v 5 zložkovej krížovej validácií

3.4.6 Vyhodnotenie modelu

V tejto časti sa zisťuje, aká presná je predpoveď nášho modelu pomocou testovacích dát. Vzhľadom nato, že testovacie dáta neboli súčasťou trénovania ani ladenia parametrov, modul s nimi pracuje po prvý krát. Môžeme o nich teda predpokladať, že sú to neznáme dáta. Do modelu pošleme vstupy z týchto dát a predikujeme výsledky. Porovnaním výstupu modelu a skutočných hodnôt zistíme presnosť nášho modelu.

3.5 Algoritmy strojového učenia

Rokmi výskumov a experimentov bolo vyvinutých viacero algoritmov strojového učenia. Každý z nich má svoje výhody aj nevýhody a navzájom sa nedajú porovnať, ktorý je najlepší. V tejto kapitole je opísané fungovanie najznámejších modelov.

3.5.1 Lineárna regresia (Linear regression)

Lineárna regresia je znázornená rovnicou, ktorá popisuje vzťah medzi vstupnými premennými X a výstupnou premennou Y pomocou špecifických váh pre vstupné premenné, ktoré sa nazývajú koeficienty [21]. Cieľom trénovania je nájsť najoptimálnejšie koeficienty.

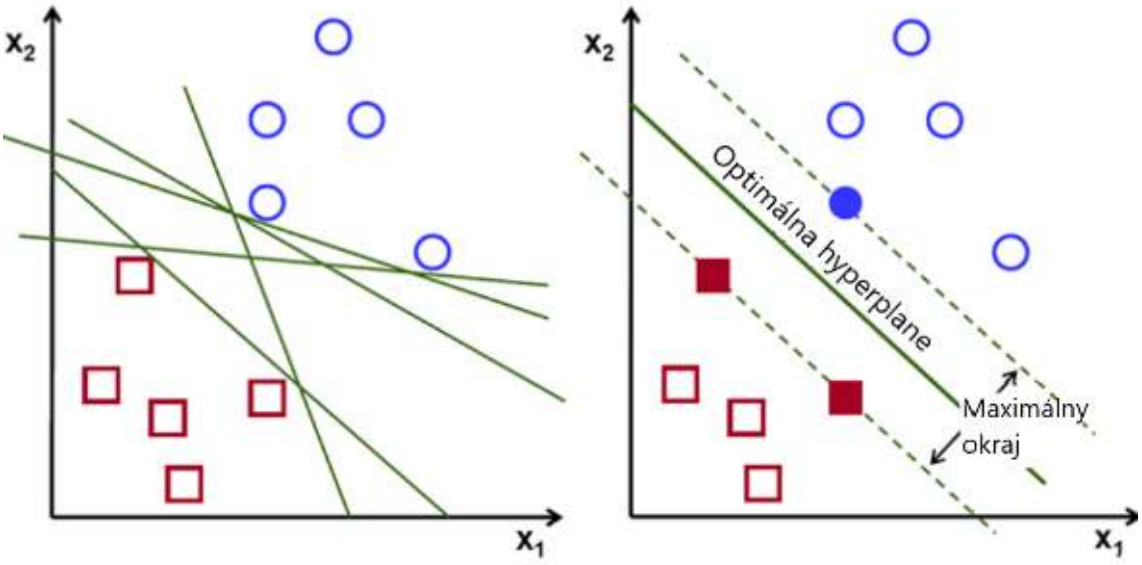
3.5.2 Metóda podporných vektorov (Support Vector Machine)

Metóda podporných vektorov je algoritmus, ktorý vie riešiť klasifikačné aj regresné úlohy. Najčastejšie sa tento model využíva na klasifikáciu do dvoch tried. V takýchto úlohách dosahuje veľmi vysokú úspešnosť. Na oddelenie vzoriek dvoch rôznych tried používa geometrický objekt nadrovina. Nadrovina je podpriestor, ktorého dimenzia je o jedna menšia ako dimenzia okolitého priestoru. To znamená, že v dvojrozmernom priestore je to krivka, v trojrozmernom je to plocha. Tento model však rieši aj viac rozmerné problémy. Dimenzionalita závisí od počtu atribútov.

Algoritmus vyhľadáva koeficienty, ktoré najlepšie popisujú hyperplane. Dôraz sa kladie na nájdenie takej hyperplane, ktorá nielenže bude rozdeľovať dáta do dvoch tried, ale bude mať aj čo najväčší priestor okolo seba, do ktorého nebudú spadať žiadne dáta. To zabezpečí, že dáta budú klasifikované do správnej triedy s vyššou pravdepodobnosťou. Na obrázku 3.3 môžeme vidieť vizualizované dáta dvoch rôznych tried, ktoré oddeľuje hyperplane. V prvom prípade je zobrazených viacero rôznych hyperplane. Všetky spĺňajú podmienku rozdelenia dát do dvoch tried správne. Tieto hyperplany však nie sú optimálne, nakoľko majú veľmi malý okraj – odstup od dát. Na druhom obrázku je optimálna hyperplane, ktorá má najväčší možný okraj.

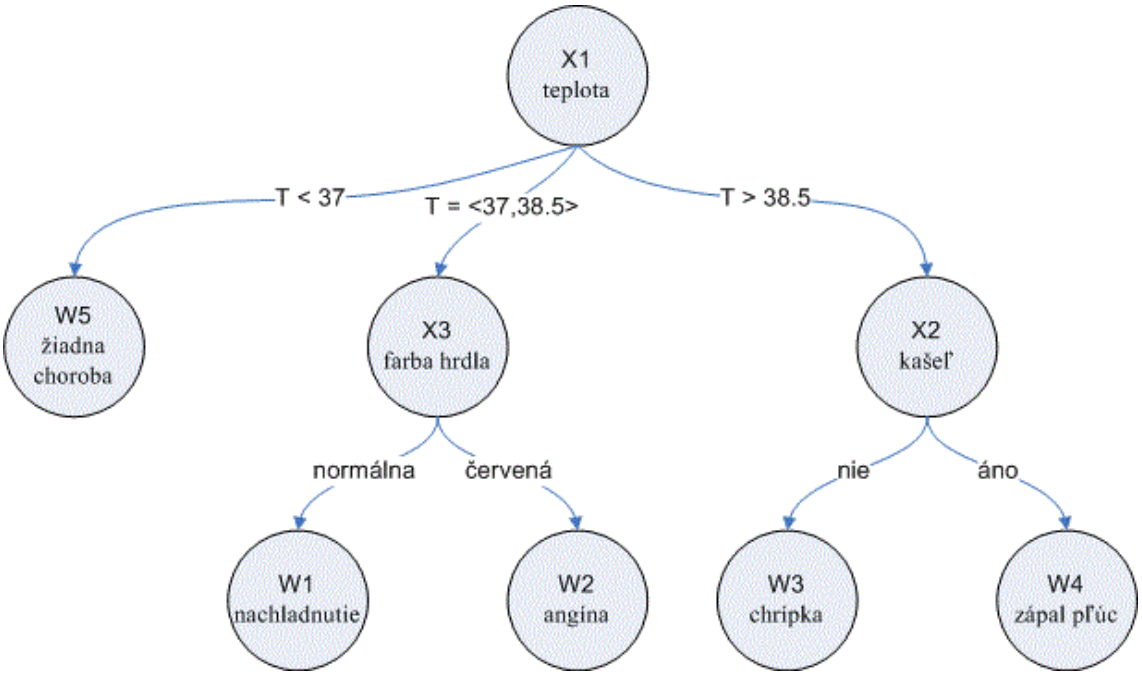
3.5.3 Rozhodovacie stromy (Decision trees)

Rozhodovacie stromy je algoritmus, ktorý je jednoduchý na pochopenie. Vizualizovať by sa dala ako štruktúra podobná vývojovému diagramu. Tento model predstavuje strom, kde v každom uzle je atribút. Na základe tohto atribútu sa algoritmus rozhoduje, ktorým smerom bude pokračovať. Každý list tohto stromu predstavuje výstupnú hodnotu. Algoritmus začína v koreni a postupnými rozhodnutiami sa dostane až do niektorého z listov. Hodnotu tohto listu potom označí



Obr. 3.3: Rozdelenie dát do dvoch tried pomocou hyperplane

za výstupnú hodnotu. Obrázok 3.4 zobrazuje jednoduchý príklad rozhodovacieho stromu, ktorý na základe atribútov (teplota, kašeľ, farba hrdla) predikuje jednu z možných diagnóz (nachladnutie, angína, chrípka, zápal pľúc, žiadna choroba).



Obr. 3.4: Príklad rozhodovacieho stromu

Tento model dobre zvláda zmiešané numerické a kategorické atribúty. Výhodou je ich prehľadnosť a jednoduchá interpretácia, ktorá umožňuje užívateľom rýchlo a ľahko vyhodnocovať získané výsledky, identifikovať kľúčové položky a vyhľadávať zaujímavé segmenty prípadov. Slabinou je náchylnosť na pretrénova-

nie. Aj malá zmena v dátach môže viesť k veľkej zmene v štruktúre stromu. Tento problém rieši nasledujúci algoritmus *Náhodné lesy*.

3.5.4 Náhodné lesy (Random forest)

Náhodné lesy[22] patria do skupiny agregátorov. Zoskupujú výsledky viacerých klasifikátorov (rozhodovacích stromov), aby získali presnejšiu a stabilnejšiu predpoveď. Týmto obmedzujú slabinu rozhodovacích stromov – pretrénovanie. Výhodou sú pomerne vysoké hodnoty úspešnosti aj bez ladenia parametrov. Naopak nevýhodou je väčšia výpočtová náročnosť. Náhodné lesy sú náročné aj na interpretáciu.

3.5.5 K-najbližších susedov (K-Nearest Neighbors)

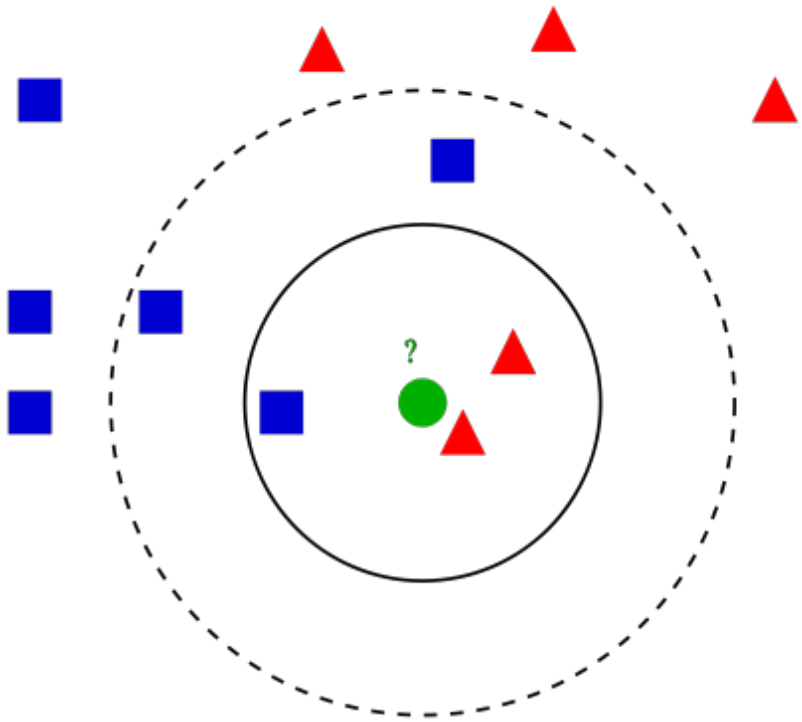
Algoritmus *K-najbližších susedov* rieši klasifikačné aj regresné problémy. Princíp fungovanie tohto modelu je založený na vyhľadávaní vzoriek s podobnými atribútami. Ak si dáta znázorníme, tak takéto vzorky sú blízko skúmanej vzorky. Sú to susedia. Spájaním takýchto susedov vznikajú K-členné skupiny [23].

V klasifikačných úlohách sa neznáma vzorka priradí do tej triedy, ktorá má v skupine najväčšie zastúpenie. V regresných úlohách je neznámej vzorke priradená hodnota, ktorá sa rovná priemeru členov v skupine. V niektorých prípadoch sa berie do úvahy aj vzdialenosť susedov. Namiesto obyčajného priemeru sa počíta vážený priemer. Bližší susedia, ktorý majú s neznámou vzorkou viac spoločného majú vyššiu váhu ako vzdialenejší. K je zvyčajne malé číslo. V prípade ak je $K = 1$ sa neznámej vzorke priradí trieda (resp. hodnota pri regresii) najbližšej známej vzorky. Na obrázku 3.5 vidíme ako môže hodnota K ovplyvniť výsledok predikcie. Ak zvolíme $K = 3$, naša neznáma (zelený kruh) bude klasifikovaná ako červený trojuholník, nakoľko väčšina z jeho troch susedov sú červené trojuholníky. Ak však zvolíme $K = 5$, neznáma bude klasifikovaná ako modrý štvorec, pretože medzi piatimi najbližšími susedmi sa vyskytuje najčastejšie práve modrý štvorec.

Dôležitá je aj metrika, ktorá meria vzdialenosť medzi jednotlivými dátami. Táto metrika priamo ovplyvňuje, ktorá dáta budú označené ako susedia. Medzi najčastejšie používané metriky patria euklidovska, mikowski a manhattan. Algoritmus sa skladá z troch krokov:

1. zvoliť K a metriku vzdialenosti

2. nájsť K najbližších susedov
3. predikovať triedu (hodnotu) neznámej na základe susedov



Obr. 3.5: Predikcia neznámej na základe susedov

3.5.6 Bayesovská sieť (Bayesian Network)

Bayesovská sieť je pravdepodobnostný model, ktorý využíva grafovú reprezentáciu (acyklický orientovaný graf) pre zobrazenie pravdepodobnostných javov medzi uzlami. Každý uzol predstavuje atribút a hrana predstavuje vzťah s inými atribútami. Učenie prebieha výpočtom pravdepodobností zo všetkých vstupných hrán a v každom uzle sa vytvára pravdepodobnostná tabuľka. Následne je možné vypočítať pravdepodobnosť nového uzla použitím Bayesovho pravidla [24]. Výhoda tejto metódy je vo veľkej škálovateľnosti a stálemu sa učeniu (pridaním uzla sa prepočítajú pravdepodobnostné tabuľky susediacich uzlov). Bayesovská sieť očakáva všetky vstupné dáta v binárnej forme. Kategorické dáta musia byť pretransformované a numerické dáta sa veľmi nehodia do tohto modelu.

3.5.7 Neurónová sieť (Neural Network)

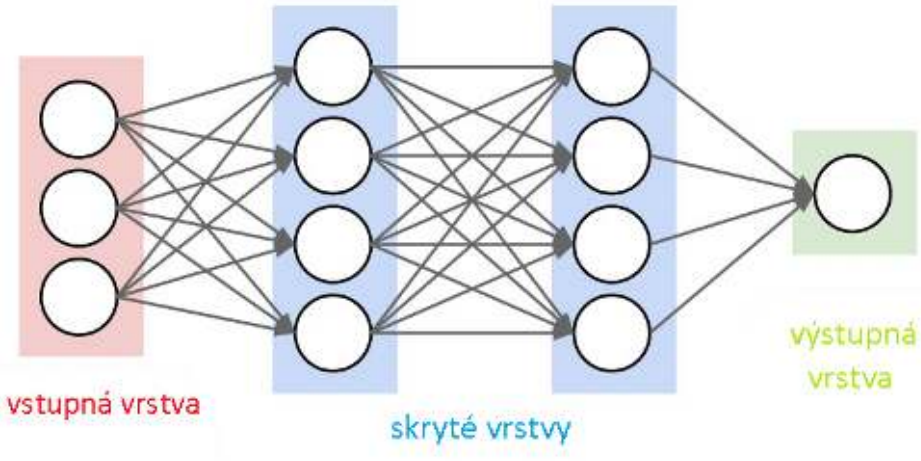
Neurónová sieť je systém inšpirovaný ľudským mozgom, ktorý napodobňuje učenie sa ľudí. Neurónové siete fungujú ako čierne skrinky, do ktorej keď vlo-

žíme vstup, dostaneme z nej výsledok bez vysvetlenia, prečo práve takýto výsledok. Je to sieť zložená z veľkého množstva elementárnych častíc – uzlov (analógia k neurónom v mozgu) [25]. Tieto uzly sú navzájom poprepájané hranami(synapsie), ktoré slúžia na prenos informácií. Každý uzol je jednoduchá a samostatná výpočtová jednotka, ktorá má aspoň jeden vstup a aspoň jeden výstup[26]. Na základe vstupov, váh a aktivačnej funkcie produkuje výstupné hodnoty. Váhy reprezentujú dôležitosť spojenia a ich hodnoty sa v procese učenia menia. Na začiatku sú hodnoty náhodne rozhodené. Algoritmus sa učí iteratívne. Každá iterácia pozostáva z predikovania výstupu na základe vstupu, porovnania výstupnej a reálnej hodnoty a upravenia váh vzhľadom na chybu predikcie. Po nájdení ideálnych hodnôt váh dosahuje tento algoritmus vysokú úspešnosť.

Aktivačné funkcie sú funkcie, ktoré transformujú vstup na výstup. Typy aktivačných funkcií sú:

- Treshold,
- Sigmoid,
- Arcus Tangens,
- Linear.

Tieto aktivačné funkcie sa nachádzajú vo všetkých uzloch okrem vstupných. Neurónová sieť sa skladá z veľkého počtu uzlov, ktoré sú organizované do vrstiev. Každá neurónová sieť obsahuje vstupnú vrstvu, kde vstup sú atribúty trénovacích dát a výstupnú vrstvu, ktorej výstup je predikovaná hodnota. Medzi týmito dvoma základnými vrstvami sa môže nachádzať niekoľko skrytých vrstiev. Vrstvy neúronovej siete sú zobrazené na obrázku 3.6.



Obr. 3.6: Vrstvy neurónovej siete

3.6 Metriky vyhodnotenia úspešnosti

Vyhodnotenie úspešnosti projektu patrí medzi základné činnosti v každom projekte. Existuje viacero rôznych metrík pre meranie úspešnosti. Každá z nich však počíta úspešnosť iným spôsobom, a tým pádom produkuje aj iné výsledky. Môže sa stať, že použitím jednej metriky dosiahnete skvelé výsledky, pričom použitím inej metriky bude úspešnosť veľmi nízka. To znamená, že použitím iba jednej metriky sa dosiahne neúplný obraz o úspešnosti modelu. Metriky úspešnosti sa delia vzhľadom na riešenú úlohu na metriky klasifikátorov a regresné metriky.

3.6.1 Metriky klasifikátorov

Základ tvorí matica zámien. Každý riadok predstavuje správne hodnoty a každý stĺpec predpovedané hodnoty. V jednotlivých bunkách sa potom nachádza počet takýchto stavov. Príklad: Ak by model používal iba dve hodnoty (pravda, nepravda), tabuľka by obsahovala štyri časti. V prvej časti by bol počet správne predpovedaných záznamov, ktoré sú pravdivé. Druhá časť by obsahovala počet nesprávne predpovedaných záznamov, ktoré sú pravdivé. Tretia by obsahovala počet nesprávne predpovedaných nepravdivých záznamov a posledná počet správne predpovedaných nepravdivých záznamov. Na základe tejto tabuľky vieme vypočítať úspešnosť podľa viacerých metrík:

- *Presnosť (Accuracy)* – pomer počtu správne predpovedaných a počtu všetkých predpovedaných. ACC = (TP+TN)/ (FP + FN + TP + TN)
- *Chyba (Error)* - pomer počtu nesprávne predpovedaných a počtu všetkých predpovedaných. ERR = (TP + TN)/(FP + FN + TP + TN)
- *Správnosť (Precision)* – pomer počtu správne predpovedaných pravdivých a počtu všetkých predpovedaných pravdivých PRE = TP/ (TP + FP)
- *Odvolanie(Recall)* – pomer počtu správne predpovedaných pravdivých a počtu všetkých pravdivých REC = TP / (TP + FN)
- *F1 skóre* – je harmonický priemer medzi správnosťou a odvolaním. Hovorí, aký presný a robustný je model. Nadobúda hodnoty od 0 až po 1. F1 = 2* ((PRE * REC) / (PRE + REC))
- *Citlivosť(TPR)* - Pomer správne pravdivých TPR = TP / (TP + FN)

- *Špecifickosť (FPR)* - Pomer nesprávnych pravdivých FPR = FP / (FP + TN)
- *Plocha pod krivkou (Area under curve)* – používa sa na meranie úspešnosti binárnych problémov. Reprezentuje schopnosť rozhodnúť medzi pravdivou a nepravdivou triedou. Ak je plocha rovná jednej, ide o perfektný model, ak je plocha 0,5 model je náhodný.

		Skutočné hodnoty	
		Pozitívne	Negatívne
Predikované hodnoty	Pozitívne	Počet správne predikovaných pozitívnych hodnôt (TP)	Počet nesprávne predikovaných pozitívnych hodnôt (FP)
	Negatívne	Počet nesprávne predikovaných negatívnych hodnôt(FN)	Počet správne predikovaných negatívnych hodnôt (TN)

Tabuľka 3.3: Matica zámen

3.6.2 Regresné metriky

Stredná absolútna chyba (MNE) je priemer rozdielov medzi skutočnou a predpovedanou hodnotou. Táto metrika napovedá, ako ďaleko je náš model od skutočnosti, ale nehovorí, ktorým smerom (či model predpovedá viac alebo menej).

Stredná kvadratická odchýlka (MSE) je priemer druhej mocniny rozdielov medzi skutočnou a predpovedanou hodnotou. Výhodou je, že z nej sa dá ľahko vypočítať smer rozdielu.

4 Návrh riešenia

V tejto kapitole sa budeme zaoberať návrhom náško riešenia. Najprv si popíšeme už existujúce riešenia. Potom podrobne opíšeme všetky kroky výsledného riešenia.

4.1 Existujúce riešenia

Vzhľadom na rozmach strojového učenia do všetkých odvetví a neustále zvyšujúci sa počet škodlivého softvéru už existujú riešenia, ktoré sa zaoberajú detekciou malvéru pomocou natrénovaných modelov strojového učenia. Výsledky natrénovaných modelov výrazne závisia na vstupných informáciách. A práve tu sa riešenia problému detekcie malvéru rozchádzajú. Každé riešenie využíva iné dáta, hľadá iné súvislosti.

4.1.1 Výučba detekcie a klasifikácie škodlivých spustiteľných súborov

Kotler spolu s Maloofom vo svojej práci [27] takisto využili dolovanie znalosti na detekciu a klasifikáciu škodlivého softvéru. Ich dátová vzorka pozostávala z 1971 neškodných vzoriek a 1651 škodlivých vzoriek. Pomocou programu hexdump konvertovali každú analyzovanú vzorku na hexadecimálny kód v ASCII formáte. Používali viacero druhov algoritmov, medzi ktorými boli napríklad *Metódu podporných vektorov*, *Rozhodovacie stromy*, *Bayesovské siete*. Vo výsledkoch dominoval algoritmus *Rozhodovacie stromy*, ktorý dosiahol úspešnosť 98%. Nedetekovaných ostalo iba šesť vzoriek z testovacieho datasetu obsahujúceho spolu 291 škodlivých vzoriek.

4.1.2 Metódy dolovania dát na detekciu nových škodlivých spustiteľných súborov

Schultz a jeho spolupracovníci navrhli metódu dolovania znalostí na detekciu nových škodlivých vzoriek [28]. Dáta pozostávali z 3265 škodlivých vzoriek a 1001 neškodných vzoriek. Vo svojej práci sa zamerali na tri typy atribútov získaných z týchto spustiteľných súborov:

- zoznam dynamicky pripojených knižníc (DLL)
- zoznam funkcií ktoré boli volané
- počet rôznych systémových volaní vrámci jednotlivých knižníc

Ako učiaci algoritmus použili *Bayesovskú sieť*. Tento algoritmus používal na natrénovanie nájdené reťazce postupnosť bytov v daných vzorkách. Výsledky tejto práce sú vcelku zaujímavé. Po nájdení najlepších možných hyperparametrov dosahovala úspešnosť vytrénovaného modelu 97,11%.

4.1.3 Detekcia internetových červov pomocou techník dolovania znalostí

M. Siddiqui a jeho kolegovia sa vo svojej práci [29] zamerali na konkrétnu skupinu malvéru a to na internetové červy. Ich dáta pozostávali z 1444 červov a 1330 čistých vzoriek. Zamerali sa hlavne na údaje z hlavičky PE súboru. Ako hlavný zdroj údajov pre atribúty boli použité sekvencie príkazov kódu. Takmer 94% týchto sekvencií sa našlo iba v jednom type vzoriek (čisté / škodlivé) a preto boli odstránené. Z ostatných častí boli vytvorené príznaky pre učiaci sa algoritmus. V tejto štúdií boli použité algoritmy ako *Rozhodovacie stromy*, *Vrecovanie*, a *Náhodné lesy*. Najlepšie výsledky dosahoval algoritmus *Náhodné lesy* s presnosťou predikcie 95,6%.

4.2 Návrh výsledného riešenia

Pri riešení problému detekcie malvéru sme sa opierali o poznatky nadobudnuté v analytickej časti. Riešenie pozostávalo z viacerých krokov. Zrejmä najdôležitejšie bolo získanie dát a extrahovanie príznakov. Nasledoval výber algoritmu, natrénovanie modelu a validácia modelu s určením dosahovanej presnosti.

Z pohľadu strojového učenia sme riešili klasifikačnú úlohu. Tomu bol prispôsobený aj výber algoritmu. Keďže existuje viacero klasifikačných algoritmov postupne sme vyskúšali niektoré z nich a porovnávali výsledky. V našom riešení sme využívali jazyk *Python* a jeho knižnicu *Scikit-learn*, v ktorej sú implementované modely strojového učenia.

4.2.1 Zozbieranie dát

Základným vstupom do predikčného modelu sú dáta. Keďže úlohou bola klasifikácia neznámej vzorky do dvoch tried (čistá vzorka, škodlivá vzorka) potrebovali sme získať dostanočne veľké množstvo čistých aj škodlivých vzoriek. Rozhodli sme sa skúmať spustiteľné súbory na platforme Windows, nakoľko práve tieto súbory sú najčastejšie napádané. Podarilo sa nám vytvoriť sadu 3064 čistých spustiteľných súborov a 3066 škodlivých súborov, čo je spolu 6130 vzoriek. Sadu čistých vzoriek tvorili rôzne aplikácie všetky s príponou .exe. Boli to grafické programy (prehliadače, grafické editory, 3D modelovacie nástroje,...), kancelárske programy(programy na tvorbu dokumentov, prezentácií, tabuliek, zobrazovanie PDF,...), internetové prehliadače, vývojové prostredia pre programovanie, hry, nástroje na prácu so súbormi a rôzne ďalšie spustiteľné súbory. Sada škodlivých súborov bola získaná z internetového repozitáru virusshare.com, ktorá poskytuje zbierku veľkého množstvá škodlivých vzoriek pre výskum a analýzu malvéru. Nie všetky tieto vzorky však boli spustiteľné súbory, a preto sme museli napísať skript v jazyku *Python*, ktorý nám tieto vzorky roztriedi a vyberie z nich iba spustiteľné súbory.

```
# Copy executables to malware folder
for filepath in glob.iglob('D:\\samples\\*'):
      file = open(filepath, "r")
      content = file.read()
      if content[0] == 'M' and content[1] == 'Z':
            copyfile(filepath, 'D:\\malware\\' + ntpath.basename(filepath))
```

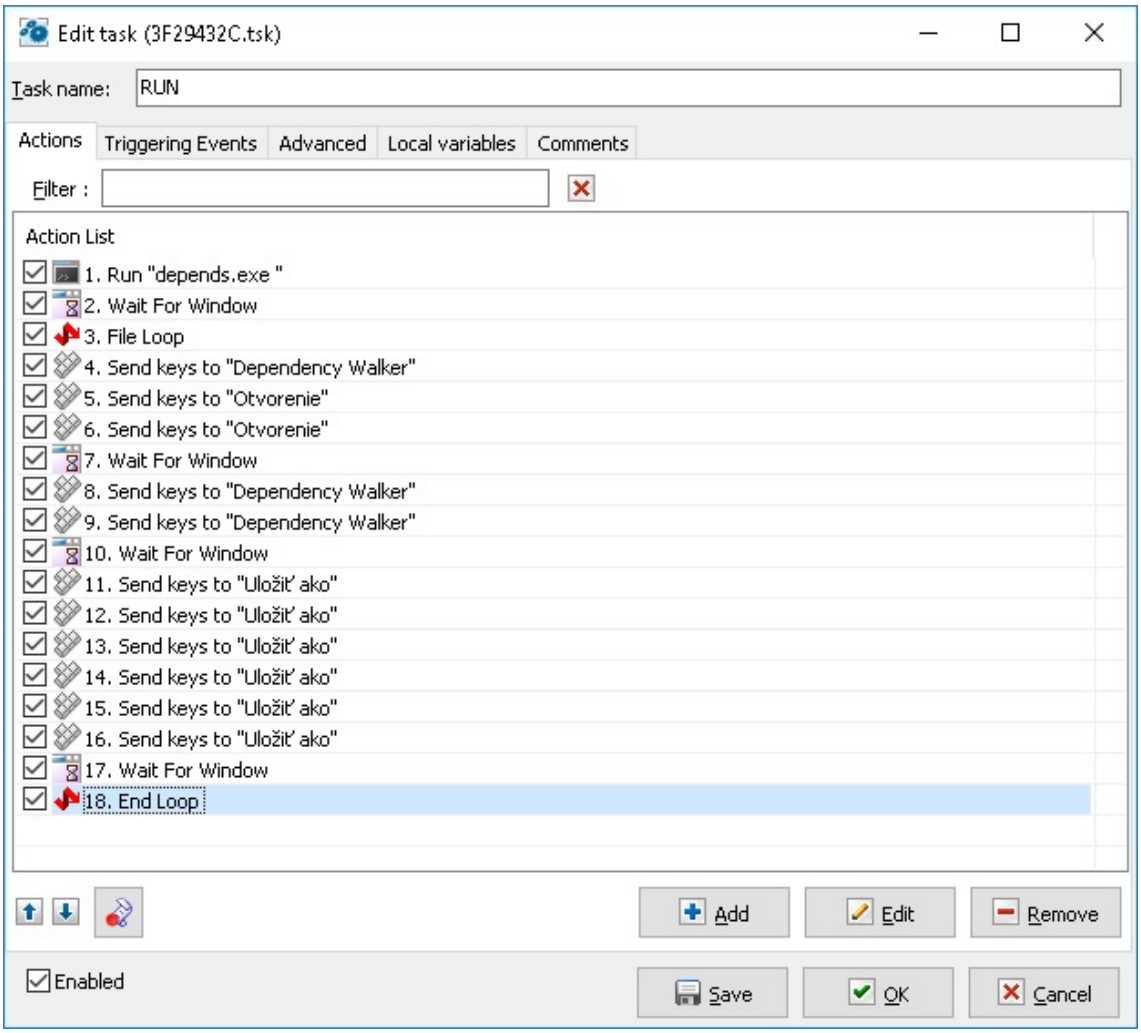
Po zozbieraní potrebných vzoriek sme potrebovali získať údaje o každej jednej vzorke. Na túto úlohu sme použili voľne dostupný nástroj *Dependency Walker*, ktorý analyzuje pripojené knižnice a použité funkcie v nich. Po analýze vzorky týmto nástrojom sme uložili získané informácie do textového súboru. Vzhľadom nato, že sme potrebovali analyzovať vyše 6000 vzoriek, manuálne to nepripadalo do úvahy. Využili sme na to nástroj Robotask, ktorý spustil program *Dependency*

Walker a následne doň posielal inštrukcie. V cykle prešiel všetky vzorky v priečinku a poslal tieto inštrukcie:

- CRTL + O - spustenie dialógu pre otvorenie súboru
- absolútna cesta – charakterizovala súbor, ktorý sa má analyzovať
- ENTER – potvrdenie vybratého súboru

Nasledovala analýza po ktorej Robotask poslal ďalšie inštrukcie:

- CTRL + S – uloženie získaných súborov
- absolútna cesta – charakterizovala súbor, kde sa majú údaje uložiť
- 3 x TAB – výber formátu uloženia, kde sme zvolili možnosť "Text so zoznamom importovaných/exportovaných funkcií"
- ENTER – potvrdenie uloženia



Obr. 4.1: Definované úlohy v nástroji Robotask

Takto sme získali 6130 textových súborov v ktorých boli informácie o jednotlivých vzorkách. Vzorky sme nespúšťali, ale kôli bezpečnosti sme tento postup na škodlivých vzorkách vykonali v bezpečnom prostredí virtuálneho stroja vytvoreného pomocou programu *Oracle VM VirtualBox*.

4.2.2 Spracovanie dát

Získané textové súbory obsahovali obrovské množstvo údajov o vzorkách. My sme sa však chceli zamerať iba na podstatné detaily na základe ktorých by sme vedeli rozlíšiť škodlivú vzorku od čistej vzorky. Verejne dostupné Windows API dáva do rúk tvorcom malvéru obrovské možnosti [30] a práve na to sme sa chceli zamerať. V knihe [31] je opísané, ktoré z funkcií používa škodlivý softvér najčastejšie. Zamerali sme sa preto práve na tieto funkcie, ktoré sú podrobnejšie opísane v kapitole 4.3. Ďalším skriptom napísaným v jazyku *Python* sme postupne prechádzali získané textové súbory s informáciami o vzorkách a hľadali v nich spomínané funkcie. Do úvahy sme brali iba tie výskyty, kedy bola funkcia skutočne použitá, teda prehľadávali sme iba importované funkcie. Výsledky vyhľadávania sme zapisovali do ďalšieho textového súboru. Na úvod tohto súboru sme vložili hlavičku s položkami „TARGET“(informácia, či vzorka je škodlivá alebo nie), „filename“(názov vzorky) a následne názvy všetkých vybratých funkcií. Rozhodli sme sa vytvoriť dve rôzne verzie. Prvá obsahovala počet výskytov každej z vybratých funkcií a druhá obsahovala iba binárnu informáciu o tom, či daná funkcia bola aspoň raz použitá. Pre každú vzorku sme vytvorili jeden riadok, kde boli informácie vo formáte: čistý súbor – 0, škodlivý súbor – 1, názov súboru, a následne počet volaní resp. binárna informácia pre každú vybratú funkciu. Tieto údaje sme oddeľovali čiarkou, a tým sme vytvorili .cvs súbor, s ktorým vedia knižnice použité v strojovom učení dobre pracovať.

```
# Find number of occurrences for each defined function in malware samples
f = open('D:\\results.txt', 'w')

#header
f.write("TARGET,")
f.write("filename,")
for function in functions:
     f.write(function + ",")

#remove last comma
```

```
f.truncate(f.tell()-1)
f.write("\n")

#data
for filepath in glob.iglob('D:\\logs1\\*.txt'):
      #target 1 - malware
      f.write("1,")

      #filename
      f.write(ntpath.basename(filepath) + ",")

      for function in functions:
            pattern = re.compile(function + "\s*Not Bound")
            count = 0
            for i, line in enumerate(open(filepath)):
                  for match in re.finditer(pattern, line):
                        count +=1

            f.write(str(count) + ",")

      # remove last comma
      f.truncate(f.tell()-1)
      f.write("\n")

f.close()
```

4.2.3 Výber modelu

Keďže sme dopredu nevedeli, ktorý z klasifikačných modelov bude pre náš problém najlepší, rozhodli sme sa použiť viacero modelov a porovnať ich výsledky. Použili sme tieto klasifikátory:

- *Rozhodovací strom*,
- *K najbližších susedov*,
- *Neurónovú sieť*,
- *Bayesovskú sieť*,
- *Náhodné lesy*,

- *Metóda podporných vektorov*.

4.2.4 Trénovanie modelu

Súbor všetkých našich vzoriek sme pomocou funkcie *train_test_split* rozdelili na trénovacie dáta a testovacie dáta v pomere 80:20 %, teda z celkového počtu 6130 vzoriek bolo 4904 trénovacích vzoriek 1226 testovacích vzoriek. Následne sme definovali množiny hodnôt pre jednotlivé parametre klasifikátorov. Pre každý z klasifikátorov sme pomocou metódy *mriežkové vyhľadávanie (grid search)* našli optimálne parametre a použili ich na natrénovanie. Na záver sme vykonali predikciu pomocou natrenovaného klasifikačného modelu na testovacej vzorke. Taktiež sme si uložili náš natrénovaný model, aby sme v prípade potreby vedeli predikovať výsledky na neznámych dátach. Tento krok sme mohli vynechať, no nakoľko je natrénovanie modelu časovo náročné, zaradili sme ho do nášho programu.

```
# Train and predict using Decision Tree algorithm

scoring = {'roc_auc', 'accuracy', 'precision', 'recall', 'f1'}
f = open('predictionsDTree.txt', 'w')

def DTree_estimate(X_train, X_test, Y_train, Y_test):
      parameters = {
      'criterion': ['entropy', 'gini'],
      'min_samples_split': [2, 5, 10],
      'min_samples_leaf': [1, 2, 5, 10],
      'presort': [True],
      }
      clf = GridSearchCV(DecisionTreeClassifier(), parameters, n_jobs = -1,
          scoring = scoring, cv=5, refit = 'accuracy')
      clf.fit(X_train, Y_train)
      predict = clf.best_estimator_.predict(X_test)
      printScore(clf, Y_test, predict)

      filename = 'DTREEmodel.sav'
      pickle.dump(clf.best_estimator_, open(filename, 'wb'))


def main():
```

```
    print ("----------------------START----------------------------")
    # load data
    df = pd.read_csv('total.txt', header=0, encoding='utf-8-sig')

    X_public = df.loc[:, "Accept":].values
    y_public = df.loc[:, "TARGET"].values

    X_train, X_test, Y_train, Y_test = train_test_split(X_public,
        y_public, test_size=0.2)

    DTree_estimate(X_train, X_test, Y_train, Y_test)
    print ("----------------------DONE----------------------------")
    f.close()

    return

if __name__ == '__main__':
    main()
```

4.3 Sledované funkcie

V tejto kapitole si predstavíme vybrané funkcie, na ktoré sme sa pri statickej analýze zamerali. Každú z funkcií aj v krátkosti opíšeme.

Accept: Táto funkcia slúži na počúvanie prichádzajúcich spojení. Indikuje, že program počúva prichádzajúce spojenia. Škodlivý softvér ho väčšinou používa na komunikáciu so svojim serverom.

AdjustTokenPrivileges: Táto funkcia sa používa na povolenie alebo zakázanie špecifických prístupových práv. Škodlivý softvér ju používa na získanie ďalších povolení a prístupov, ktoré mu nepatria.

AttachThreadInput: Táto funkcia pripojí spracovanie vstupov z jedného vlákna do druhého, takže aj druhé vlákno prijíma vstupné udalosti, ako sú napríklad vstupy z myši alebo klávesnice. Najčastejšie túto funkciu využívajú špionážne softvéry (napr. keylogger).

Bind: Táto funkcia slúži na priradenie lokálnej adresy k socketu, aby bolo možné počúvať prichádzajúce pripojenia.

BitBlt: Táto funkcia sa používa na kopírovanie grafických údajov z jedného zariadenia na druhé. Spyware niekedy používa túto funkciu na zachytenie snímok obrazovky.

CertOpenSystemStore: Táto funkcia sa používa na prístup k certifikátom uloženým v lokálnom systéme.

Connect: Táto funkcia sa používa na pripojenie k vzdialenému socketu. Malvér často používa funkciu s nízko úrovňovým pripojením k serveru.

ConnectNamedPipe: Táto funkcia slúži na vytvorenie serverovej rúry pre medziprocesovú komunikáciu, ktorá čaká na pripojenie klientskej rúry. Škodlivý softvér používa túto funkciu na zjednodušenie pripojenia k serveru.

ControlService: Táto funkcia slúži na spustenie, zastavenie, zmenu alebo odosielanie signálu spustenej službe. Ak malvér používa vlastnú škodlivú službu, je potrebné analyzovať kód, ktorý implementuje službu, aby sa určil účel volania tejto funkcie.

CreateFile: Vytvorí nový alebo otvorí existujúci súbor.

CreateFileMapping: Táto funkcia sa používa na mapovanie súborov, ktoré sú načítané do pamäte a sprístupňuje ich prostredníctvom pamäťových adries. Škodlivý softvér používa túto funkciu na čítanie a úpravu súborov PE.

CreateMutex: Táto funkcia vytvára objekt vzájomného vylúčenia (mutex), ktorý môže byť používaný škodlivým softvérom, aby sa zabezpečilo, že v danom čase beží iba jedna inštancia škodlivého softvéru. Malvér často používa pevné názvy pre mutexy, ktoré môžu byť dobrými indikátormi hostiteľa na zistenie ďalších inštalácií škodlivého softvéru.

CreateProcess: Táto funkcia vytvára a spúšťa nový proces. Ak malvér vytvorí nový proces, musí sa analyzovať aj nový proces.

CreateRemoteThread: Táto funkcia slúži na spustenie vlákna vo vzdialenom procese. Spúšťače a iný škodlivý softvér používa CreateRemoteThread na vkladanie kódu do iného procesu.

CreateService: Táto funkcia sa používa na vytvorenie služby, ktorá sa spúšťa pri zapnutí zariadenia. Malvér používa túto funkciu na zabezpečenie perzistencie,.

CreateToolhelp32Snapshot: Táto funkcia slúži na vytvorenie snímok procesov, pamäte programu, vlákien a modulov. Malvér často používa túto funkciu ako súčasť kódu, ktorý prechádza procesmi alebo vláknami.

CryptAcquireContext: Táto funkcia je často prvou funkciou používanou škodlivým softvérom na inicializáciu používania šifrovania systému Windows.

DeviceIoControl: Táto funkcia odosiela riadiace správy od užívateľa do ovládača zariadenia. Malware v jadre, ktorý potrebuje preniesť informácie medzi užívateľským priestorom a priestorom jadra, často používa túto funkciu.

EnableExecuteProtectionSupport: Táto funkcia slúži na úpravu nastavenia ochrany dátového zabezpečenia (Data Execution Protection - DEP) hostiteľa, čím je náchylnejšie na útok.

EnumProcesses: Táto funkcia slúži na prechádzanie medzi bežiacimi procesmi v systéme. Malvér často prechádza procesy, aby našiel proces, do ktorého sa môže spustiť.

EnumProcessModules: Táto funkcia slúži na prechádzanie načítaných modulov (spustiteľných súborov a DLL) pre daný proces. Malvér prechádza moduly pri pokuse o útok.

FindFirstFile / FindNextFile: Táto funkcia slúži na vyhľadávanie v adresári a prechádzanie súborového systému.

FindResource: Táto funkcia slúži na vyhľadanie zdroja v spustiteľnom alebo načítanom DLL. Malvér niekedy používa prostriedky na ukladanie reťazcov, informácií o konfigurácii alebo iných škodlivých súborov. Ak sa táto funkcia použije, je potrebné skontrolovať sekciu .rsrc v hlavičke PE škodlivého softvéru.

FindWindow: Táto funkcia slúži na hľadanie otvoreného okna na pracovnej ploche.

FtpPutFile: Táto funkcia sa používa na odoslanie súboru na vzdialený FTP server.

GetAdaptersInfo: Táto funkcia slúži na získanie informácií o sieťových adaptéroch v systéme. Škodlivý softvér niekedy volá GetAdaptersInfo vo fáze zhromažďovania dát a získavajú informácie o infikovaných počítačoch. V niektorých prípadoch sa zvyknú zhromažďovať MAC adresy na detekciu nástroja VMware, ktorý je súčasťou techniky antivirtuálneho stroja.

GetAsyncKeyState: Táto funkcia sa používa na určenie, či je stlačené určité tlačidlo. Malvér niekedy používa túto funkciu na implementáciu keyloggeru.

GetDC: Táto funkcia vráti objekt, ktorý je zodpovedný za okno alebo celú obrazovku. Spyware, ktorý zachytáva obrazovky, často používa túto funkciu.

GetForegroundWindow: Táto funkcia vráti obslužný objekt okna, ktoré je aktuálne aktívne. Keyloggeri bežne používajú túto funkciu na určenie, v ktorom okne užívateľ je používaná klávesnica.

Gethostbyname: Táto funkcia slúži na vyhľadávania DNS záznamov pred vytvorením IP pripojenia k vzdialenému hostiteľovi. Názvy hostiteľov, ktoré slúžia ako škodlivé servery, často vytvárajú dobré podpisy založené na sieti.

Gethostname: Táto funkcia slúži na získanie názvu hostiteľa počítača. Škodlivý softvér niekedy používajú názov túto funkciu vo fáze zhromažďovania informácií o stroji obete.

GetKeyState: Táto funkcia sa používa v programoch keyloggerov na získanie stavu konkrétnej klávesy na klávesnici.

GetModuleFilename: Táto funkcia vráti názov súboru modulu, ktorý je načítaný do aktuálneho procesu. Malvér môže túto funkciu používať na úpravu alebo kopírovanie súborov v aktuálnom procese.

GetModuleHandle: Táto funkcia slúži na získanie objektu načítaného modulu. Malvér môže používať túto funkciu na vyhľadanie a zmenu kódu v načítanom module alebo na hľadanie správneho umiestnenia pre vkladanie škodlivého kódu.

GetProcAddress: Táto funkcia slúži na získanie adresy funkcie v DLL načítanej do pamäte. Používa sa na import funkcií z iných DLL okrem funkcií importovaných v hlavičke súboru PE.

GetStartupInfo: Táto funkcia slúži na získanie štruktúry obsahujúcej podrobnosti o tom, ako bol aktuálny proces nakonfigurovaný pri spustení.

GetSystemDefaultLangId: Táto funkcia vráti predvolené nastavenia jazyka pre systém. Tieto nastavenia sú používané škodlivými softvérmi špeciálne navrhnutými pre útoky v určitých geografických regiónoch.

GetTempPath: Táto funkcia vráti dočasnú cestu k súboru. Ak malvér volá túto funkciu, je potrebné skontrolovať, či číta alebo zapisuje akékoľvek súbory do dočasnej cesty.

GetThreadContext: Táto funkcia vráti obsahovú štruktúru daného vlákna. Táto štruktúra vlákna uchováva všetky informácie o vlákne, ako sú napríklad hodnoty registrov a aktuálny stav.

GetVersionEx: Táto funkcia vráti informácie o aktuálnej verzii systému Windows. Môže sa použiť ako súčasť prieskumu obete alebo vybrať medzi rôznymi posunmi

pre nedokumentované štruktúry, ktoré sa zmenili medzi rôznymi verziami systému Windows.

GetWindowsDirectory: Táto funkcia vráti cestu k adresáru systému Windows (zvyčajne C: Windows). Malvér niekedy používa túto výzvu na určenie, do ktorého adresára nainštalujte ďalšie škodlivé programy.

Inet_addr: Táto funkcia konvertuje reťazec adresy IP (napr. 127.0.0.1) tak, aby mohla byť použitá funkciami, ako je Connect. Zadaný reťazec môže byť niekedy použitý ako sieťový podpis.

InternetOpen: Táto funkcia inicializuje funkcie vysokorýchlostného prístupu na Internet z aplikácie WinINet, ako napríklad InternetOpenUrl a InternetReadFile. Hľadanie InternetOpen je dobrý spôsob, ako nájsť funkciu prístupu k internetu. Jedným z parametrov pre InternetOpen je User-Agent, ktorý môže niekedy vytvoriť dobrý sieťový podpis.

InternetOpenUrl: Táto funkcia otvára špecifickú adresu URL pre pripojenie pomocou protokolu FTP, HTTP alebo HTTPS. Ak je URL fixná, môže byť dobrým sieťovým podpisom.

InternetReadFile: Táto funkcia číta údaje z predtým otvorenej adresy URL.

InternetWriteFile: Táto funkcia zapisuje údaje do predtým otvorenej adresy URL.

IsNTAdmin: Táto funkcia skontroluje, či používateľ má privilégiá správcu.

IsWoW64Process: Táto funkcia sa používa 32-bitovým procesom na určenie, či je spustený na 64-bitovom operačnom systéme.

LdrLoadDll: Toto je funkcia nízkej úrovne na načítanie DLL do procesu, rovnako ako LoadLibrary. Normálne programy používajú funkciu LoadLibrary a prítomnosť tohto importu môže naznačovať program, ktorý sa pokúša byť nepríjemným.

LoadResource: Táto funkcia načíta zdroj z PE súboru do pamäte. Malvér niekedy používa zdroje na ukladanie reťazcov, informácií o konfigurácii alebo iných škodlivých súborov.

LsaEnumerateLogonSessions: Táto funkcia sa používa na prechádzanie relácií prihlásenia na aktuálnom systéme, ktoré môžu byť použité ako súčasť autorizačného nástroja.

MapViewOfFile: Táto funkcia sa používa na mapovanie súboru do pamäte a sprístupnenie obsahu súboru prostredníctvom pamäťových adries. Škodlivý softvér

používa túto funkciu na čítanie a úpravu súborov PE. Použitím funkcie MapViewOfFile sa malvér môže vyhnúť používaniu funkcie WriteFile na úpravu obsahu súboru.

MapVirtualKey: Táto funkcia slúži na preklad virtuálneho kľúča do hodnoty znaku. Často ju používajú keyloggere.

Module32First/ Module32Next: Táto funkcia slúži na prechádzanie modulov načítaných do procesu. Škodlivý softvér používa túto funkciu na určenie toho, kam sa má vložiť kód.

NetScheduleJobAdd: Táto funkcia podáva žiadosť o spustenie programu v určený deň a čas. Malware môže použiť funkciu NetScheduleJobAdd na spustenie iného programu. Je to dôležitý ukazovateľ, ktorý definuje program naplánovaný na spustenie v budúcnosti.

NetShareEnum: Táto funkcia slúži na predchádzanie sieťových zariadení.

NtQueryDirectoryFile: Táto funkcia vracia informácie o súboroch v adresári. Malvér obvykle zakrýva túto funkciu, aby skryl súbory.

NtQueryInformationProcess: Táto funkcia slúži na vrátenie rôznych informácií o určenom procese. Niekedy sa používa aj ako technika ladenia, pretože môže vrátiť rovnaké informácie ako CheckRemoteDebuggerPresent.

NtSetInformationProcess: Táto funkcia sa používa na zmenu úrovne privilégií programu alebo na obchádzanie prevencie pred vykonávaním údajov (Data Execution Prevention – DEP).

OpenMutex: Táto funkcia otvára objekt vzájomného vylúčenia, ktorý môže byť používaný škodlivým softvérom, aby sa zaistilo, že v systéme v ľubovoľnom čase beží iba jedna inštancia škodlivého softvéru. Malvér často používa pevné názvy pre mutexy, ktoré môžu byť dobrými indikátormi hostiteľa.

OpenProcess: Táto funkcia slúži na otvorenie obslužného objektu pre iný proces spustený v systéme. Takýto objekt sa môže použiť na čítanie a zapisovanie do pamäte iného procesu alebo na vloženie kódu do iného procesu.

OutputDebugString: Táto funkcia sa používa na výpis reťazca do debuggeru, ak je pripojený. Môže sa použiť ako technika proti ladeniu.

PeekNamedPipe: Táto funkcia sa používa na kopírovanie údajov z pomenovanej rúry bez odstránenia údajov z potrubia.

Process32First / Process32Next: Táto funkcia sa používa na začatie prechádzania procesov od predchádzajúceho volania na CreateToolhelp32Snapshot.

QueueUserAPC: Táto funkcia sa používa na vykonanie kódu pre iné vlákno. Malvér niekedy používa funkciu QueueUserAPC na vkladanie kódu do iného procesu.

ReadProcessMemory: Táto funkcia slúži na čítanie pamäte vzdialeného procesu.

Recv: Táto funkcia slúži na prijímanie údajov zo vzdialeného počítača. Malvér často používa túto funkciu na prijímanie údajov z riadiaceho servera.

RegisterHotKey: Táto funkcia sa používa na registráciu handleru, ktorý má byť upozornený, kedykoľvek používateľ zadá konkrétnu kombináciu klávesov (napr. CTRL-ALT-J) bez ohľadu na to, ktoré okno je aktívne, keď používateľ stlačí kombináciu klávesov. Táto funkcia sa niekedy používa špionážnym softvérom, ktorý zostáva skrytý pre používateľa, kým sa nestlačí kombinácia klávesov.

RegOpenKey: Táto funkcia sa používa na otvorenie popisu kľúča v registroch na čítanie a úpravu. Kľúče v registroch sa niekedy používajú na dosiahnutie perzistencie. Registre tiež obsahuje celý rad informácií o operačnom systéme a nastavení aplikácie.

ResumeThread: Táto funkcia slúži na obnovenie už ukončeného vlákna.

RtlCreateRegistryKey: Táto funkcia slúži na vytvorenie registra z kódu v režimu jadra.

RtlWriteRegistryValue: Táto funkcia sa používa na zápis hodnoty do registrov z kódu jadra.

SamIConnect: Táto funkcia sa používa na pripojenie k správcovi bezpečnosti účtu (Security Account Manager - SAM), aby sa mohli uskutočniť volania, ktoré majú potrebné práva. Hash-dumpingové programy pristupujú k databáze SAM, aby získali hash užívateľských prihlasovacích hesiel.

SamIGetPrivateData: Táto funkcia sa používa na dotazovanie súkromných informácií o konkrétnom používateľovi z databázy správcu bezpečnosti účtu (SAM).

SamQueryInformationUse: Táto funkcia slúži na dotazovanie informácií o konkrétnom používateľovi v databáze správcu bezpečnosti účtu (SAM).

Send: Táto funkcia sa používa na odosielanie údajov na vzdialenom počítači. Je často využívaná škodlivým softvérom na odosielanie dát na vzdialený server.

SetFileTime: Táto funkcia slúži na úpravu času vytvorenia, prístupu alebo posledného zmenenia súboru. Malware často používa túto funkciu na skrytie škodlivých aktivít.

SetThreadContext: Táto funkcia slúži na úpravu kontextu daného vlákna.

SetWindowsHookEx: Táto funkcia sa používa na nastavenie funkcie, ktorá sa volá pri každom vyvolaní určitej udalosti. Poskytuje taktiež aj jednoduchý spôsob načítania DLL do všetkých procesov grafického rozhrania v systéme. Túto funkciu niekedy pridáva kompilátor.

SfcTerminateWatcherThread: Táto funkcia sa používa na vypnutie ochrany súborov systému Windows a úpravu súborov, ktoré by inak boli chránené.

ShellExecute: Táto funkcia sa používa na vykonanie iného programu.

StartServiceCtrlDispatcher: Táto funkcia slúži na pripojenie hlavného vlákna procesu k správcovi riadenia služieb (service control manager). Každý proces, ktorý beží ako služba, musí túto funkciu zavolať do 30 sekúnd od uvedenia do prevádzky. Umiestnenie tejto funkcie v škodlivom kóde hovorí, že funkcia by mala byť spustená ako služba.

SuspendThread: Táto funkcia slúži na ukončenie vlákna. Malvér niekedy pozastavuje vlákno nato, aby ho mohol modifikovať vkladaním škodlivého kódu.

System: Táto funkcia slúži na spustenie iného programu poskytovaného niektorými C runtime knižnicami. V systéme Windows slúži táto funkcia na zaobalenie pre CreateProcess.

Thread32First / Thread32Next: Táto funkcia sa používa na prechádzanie vlákien procesu.

Toolhelp32ReadProcessMemory: Táto funkcia slúži na čítanie pamäte vzdialeného procesu.

URLDownloadToFile: Táto funkcia slúži na stiahnutie súboru z webového servera a jeho uloženie na disk. Táto funkcia je obľúbená u downloaderov, pretože implementuje všetky funkcie downloadera pri volaní jednej funkcie.

VirtualAllocEx: Táto funkcia slúži na prideľovanie pamätí, ktorá môže byť alokovaná vo vzdialenom procese. Malvér niekedy používa VirtualAllocEx ako súčasť vkladania škodlivého kódu.

VirtualProtectEx: Táto funkcia sa používa na zmenu ochrany v oblasti pamäte.

Malvér môže použiť túto funkciu na zmenu časti pamäte iba na čítanie na spustiteľný súbor.

WideCharToMultiByte: Táto funkcia slúži na konverziu reťazca Unicode na reťazec ASCII.

WinExec: Táto funkcia slúži na spustenie iného programu.

WriteProcessMemory: Táto funkcia sa používa na zapisovanie údajov do vzdialeného procesu.

WSAStartup: Táto funkcia sa používa na inicializáciu funkcií siete na nízkej úrovni. Hľadanie volaní WSAStartup môže byť často jednoduchým spôsobom, ako nájsť začiatok funkcií súvisiacich so sieťou.

5 Vyhodnotenie

Naše riešenie vznikalo vo viacerých krokoch, kde boli postupne využívané skripty v jazyku *Python*, nástroje *Dependency Walker* a *Robotask* a jeden hlavný program v jazyku *Python* s využitím knižnice *Scikit-learn*, ktorý vytvoril predikčný model, natrénoval sa na trénovacích dátach, predikoval výsledky na neznámych dátach a uložil natrénovaný model.. Vytvorili sme viacero rôznych modelov využívajúcich rôzne algoritmy. Na vyhodnotenie úspešnosti nášho modelu počas trénovania bola využitá *metrika presnosti (accuracy score)*, ktorej hodnota sa rovná pomeru správne predikovaných vzoriek ku všetkým predikovaným vzorkám, kde najlepšia hodnota je 1 a najhoršia hodnota je 0. Po predikcii neznámych (testovacích) vzoriek sme použili viacero metrík pre popis úspešnosti nášho modelu. Medzi najviac vypovedajúce môžeme zaradiť *maticu zámen (confusion matrix)* a *metriku presnosti(accuracy score)*. Riešenie sme aplikovali na dve rôzne množiny dát. Prvá množina dát obsahuje vzorky s počtom výskytu vybratých volaných funkcií. Druhá množina obsahuje dáta s informáciami, či dané funkcie boli volané alebo nie.

5.1 Výsledky predikcie na dátach s počtom volaní vybratých Windows API funkcií

V prvom experimente sme použili dáta, ktoré reprezentovali počet výskytov vybratých volaných Windows API funkcií v danej vzorke. Dáta rozdelené na trénovacie a testovacie v pomere 80:20%. Použitím týchto dát sme natrénovali veľmi presné modely. Najlepšiu presnosť mali zhodne tri modely, a to *Rozdohovací strom*, *K najbližších susedov* a *Náhodné lesy*. Tieto modely dosahovali úspešnosť predikcie až 99,84% s jedným jediným „false positive“ prípadom a takisto jedným „false negative“ prípadom z 1226 predikovaných vzoriek. Výsledky predikcie týchto modelov zobrazuje aj matica zámen v tabuľke 5.1.

		Skutočné hodnoty	
		Čistá vzorka	Škodlivá vzorka
Predikované hodnoty	Čistá vzorka	614	1
	Škodlivá vzorka	1	610

Tabuľka 5.1: Matica zámen modelov Rozhodovací strom, K najbližších susedov a Náhodné lesy

V procese trénovania modelu sme hľadali aj hyperparametre, pri ktorých natrénovaný model dosahuje najlepšie výsledky. Najlepšie parametre nájdené technikou grid search boli:

- **Rozhodovací strom** {'criterion': 'entropy','min_samples_leaf': 1, 'min_samples_split': 5, 'presort': True}
- **K najbližších susedov** {'leaf_size': 5, 'n_neighbors': 3, 'p': 1}
- **Náhodné lesy** {'criterion': 'entropy', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 10}

Iba o jednú nepresnú predikciu boli horšie algoritmy Metódy podporných vektorov a Neurónová sieť s presnosťou 99,76%, dvoma prípadmi false posivtive a jedným false negative.

		Skutočné hodnoty	
		Čistá vzorka	Škodlivá vzorka
Predikované hodnoty	Čistá vzorka	613	2
	Škodlivá vzorka	1	610

Tabuľka 5.2: Matica zámen reprezentujúca úspešnosť modelov Metódy podporných vektorov a Neurónovej siete

Najlepšie nájdené parametre pre tieto algoritmy boli:

- **Metóda podporných vektorov** {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
- **Neurónová sieť** {'activation': 'identity', 'alpha': 1e-05, 'solver': 'lbfgs'}

Najhoršiu úspešnosť mal algoritmus Bayesovská sieť. Presnosť dosahovala 99.67%. Všetky vzorky označené ako čisté boli naozaj čistými vzorkami. Objavili sa však až štyri prípady false negative.

		Skutočné hodnoty	
		Čistá vzorka	Škodlivá vzorka
Predikované hodnoty	Čistá vzorka	615	0
	Škodlivá vzorka	4	607

Tabuľka 5.3: Matica zámen reprezentujúca úspešnosť modelu Bayesovskej siete

V tabuľke 5.4 sú znázornené hodnoty úspešnosti všetkých použitých modelov. Hodnotenie riešenia je vyjadrené v metrikách presnosť, správnosť, RECALL, F1 skóre a ROC skóre.

Algoritmus	**Presnosť**	**Správnosť**	**RECALL**	**F1 skóre**	**ROC skore**
Rozhodovacie stromy	0.9984	0.9984	0.9984	0.9984	0.9984
K najlbižčích susedov	0.9984	0.9984	0.9984	0.9984	0.9984
Náhodné lesy	0.9984	0.9984	0.9984	0.9984	0.9984
Metóda podporných vektorov	0.9976	0.9967	0.9984	0.9975	0.9976
Neurónová sieť	0.9976	0.9967	0.9984	0.9975	0.9976
Bayesovská sieť	0.9967	1.0	0.9935	0.9967	0.9967

Tabuľka 5.4: Úspešnosť algoritmov na dátach s počtom výskytu funkcií vo vzorkách

5.2 Výsledky predikcie na dátach s binárnou informáciou o volaní vybratých Windows API funkcií

V druhom experimente sme použili dáta, ktoré neobsahovali informáciu o počte volaní vybratých funkcií, ale iba informáciu či bola dana funkcia volaná alebo nie. Použitím týchto dát sme dosiahli ešte o čosi lepšie výsledky, čo nás prekvapilo. Očakávali sme presný opak, vzhľadom nato, že presný počet výskytu volaní funkcií poskytuje viac informácií. Tento fakt znamená, že nezáleží ako často boli funkcie volané, ale na tom či boli vôbec volané. Najlepšie modely, a v tomto prípade to boli všetky okrem *Bayesovskej siete*, dosahovali presnosť 99,93% a predikovali iba jediný prípad false positive a žiaden false negative. To znamená, že modely nedokázali detekovať jednu škodlivú vzorku a označili ju ako čistú. Matica zámen je totožná pre spomínaných päť modelov okrem Bayesovskej siete a je zobrazená v tabuľke 5.5.

		Skutočné hodnoty	
		Čistá vzorka	Škodlivá vzorka
Predikované hodnoty	Čistá vzorka	614	1
	Škodlivá vzorka	0	611

Tabuľka 5.5: Matica zámen reprezentujúca úspešnosť použitých modelov okrem Bayesovskej siete

Aj v tomto druhom experimente sme v procese trénovania hľadali najlepšie parametre pre naše modely. Pomocou techniky grid search sme dospeli k týmto výsledkom:

- **Rozhodovací strom** {'criterion': 'entropy', 'min_samples_leaf': 2, 'min_samples_split': 10, 'presort': True}
- **K najbližších susedov** {'leaf_size': 5, 'n_neighbors': 3, 'p': 1}
- **Náhodné lesy** {'criterion': 'entropy', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 10}
- **Neurónová sieť** {'activation': 'identity', 'alpha': 1e-05, 'solver': 'lbfgs'}
- **Metóda podporných vektorov** {'C': 1, 'gamma': 1e-11, 'kernel': 'linear'}

Algoritmus *Bayesovská sieť* dosiahol opäť najnižšiu úspešnosť. Presnosť a počet nesprávne predikovaných vzoriek boli zhodné s prvým experimentom. Presnosť teda dosiahla 99.67% a algoritmus predikoval štyri false negative prípady. Matica zámen vyzerá presne ako v prvom experimente 5.3. Zaujímavosťou tohto modelu je, že v obidvoch experimentoch detekoval všetky škodlivé vzorky a označil ich správne.

Algoritmus	**Presnosť**	**Správnosť**	**RECALL**	**F1 skóre**	**ROC skore**
Rozhodovacie stromy	0.9993	0.9988	1.0	0.9994	0.9992
K najlbižčích susedov	0.9993	0.9988	1.0	0.9994	0.9992
Náhodné lesy	0.9993	0.9988	1.0	0.9994	0.9992
Metóda podporných vektorov	0.9993	0.9988	1.0	0.9994	0.9992
Neurónová sieť	0.9993	0.9988	1.0	0.9994	0.9992
Bayesovská sieť	0.9972	1.0	0.9952	0.9976	0.9976

Tabuľka 5.6: Úspešnosť algoritmov na binárnych dátach o výskyte funkcií vo vzorkách

6 Záver

Cieľom tejto diplomovej práce bolo zistiť, či je možné pomocou strojového učenia detekovať škodlivý softvér na základe jeho statickej analýzy. Na základe tohto cieľa bolo potrebné získať potrebné dáta, vytvoriť klasifikačný model a zistiť jeho úspešnosť pri predikovaní na neznámych vzorkách.

Analytická časť sa zaoberala dvoma hlavnými oblasťami a to statickou analýzou škodlivého softvéru a strojovým učením. V úvode bolo objasnené čo vlastne statická analýza je a čím sa zaoberá. Následne boli opísané postupy, ktoré sa používajú v základnej statickej analýze a pokročilej statickej analýze. Druhá polovica analytickej časti sa zaoberá strojovým učením. Po základom opise strojového učenia je uvedené aj jeho praktické využitie. Následne je vysvetlený celý proces strojového učenia od získania dát až po výslednú predpoveď. V závere analytickej časti sú opísané niektoré z algoritmov využívané v strojovom učení a metriky používané na hodnotenie úspešnosti natrénovaných modelov.

Jadro práce sa venuje opisu existujúcich riešení a návrhu zvoleného riešenia. Na úvod sú opísané tri práce, ktoré sa venujú detekcii malvéru pomocou strojového učenia. Každá z prác pristupuje k tomuto problému inak. Využíva rozdielne dáta aj rozdielne modely. Uvedená je aj úspešnosť týchto prác. V popise samotného riešenia je detailne vysvetlený každý jeden krok aj s ukážkami použitého kódu a nástrojov. Podrobne je opísaný proces získavania a spracovania dát. Uvedené sú aj všetky funkcie, na ktoré sa riešenie sústredilo a ich krátky opis. Následne je opísaný proces natrénovania modelu. V závere kapitoly je opísaný spôsob vyhodnotenia úspešnosti natrénovaných modelov.

Výsledkom tejto diplomovej práce sú natrénované klasifikačné modely. Použité algoritmy boli *Rozhodovací strom, K najbližších susedov, Náhodné lesy, Metóda podporných vektorov, Neurónová sieť* a *Bayesovská sieť*. Modely boli natrénované na 6130 vzorkách, z ktorých bolo 3064 čistých vzoriek a 3066 škodlivých vzoriek. Všetky z týchto natrénovaných modelov dosahujú vysokú úspešnosť podľa rôz-

nych metrík. Na základe týchto výsledkov považujeme naše riešenie za úspešné. Úspešnosť natrénovaných modelov však môže skresľovať fakt, že škodlivé vzorky boli z jednej vírusovej rodiny. To znamená, že mali podobný kód a podobné volanie funkcií. Zozbieraním väčšieho množstva škodlivých vzoriek z rôznych rodín by mohla úspešnosť aj klesnúť, no natrénované modely by boli všeobecnejšie.

Na základe tejto práce by v budúcnosti mohol byť vytvorený nástroj, ktorý vykoná všetky kroky tohto riešenia automaticky. Takýto nástroj by mohol detekovať neznáme vzorky škodlivého softvéru. Nato však bude potrebné zozbieranie a následné natrénovanie na oveľa väčšom počte vzoriek. Zaujímavým rozšírením riešenia by mohlo byť aj použitie iných modelov, ktoré neboli v tejto práci použité.

Literatúra

[1] Kaspersky Lab. *Kaspersky Security Bulletin 2015. Overall statistics for 2015*. URL: https://securelist.com/kaspersky-security-bulletin-2015-overall-statistics-for-2015/73038/ (cit. 28. 08. 2018).

[2] Juniper Research. *CYBERCRIME WILL COST BUSINESSES OVER 2TRILLION BY 2019*. URL: https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion (cit. 20. 10. 2018).

[3] J. PIACEK a M. KRAVCIK. *OTVORENA FILOZOFICKA ENCYKLOPEDIA*. URL: http://dai.fmph.uniba.sk/~filit/fva/analyza.html (cit. 10. 08. 2018).

[4] J. HAKKARAINEN. "Malware Analysis Environment for Windows Targeted Malware". Dipl. pr. JAMK University of Applied Sciences, máj 2015.

[5] B. J. PRASAD, H. ANNANGI a K. S. PENDYALA. "Basic Static Malware Analysis using open source tools". In: *CYBER SECURITY COMMUNITY* 24 (feb. 2016).

[6] VIRUSTOTAL. URL: https://www.virustotal.com/#/home/upload (cit. 20. 10. 2018).

[7] VIRSCAN. URL: http://virscan.org (cit. 20. 10. 2018).

[8] G ERDELYI. *Reverse Engineering III: PE Format*. URL: http://www.cse.tkk.fi/fi/opinnot/T-110.6220/2010_Spring_Malware_Analysis_and_Antivirus_Tchnologies/luennot-files/Erdelyi-Reverse_engineering_2.pdf (cit. 18. 09. 2018).

[9] E. EILAM. *Reversing: Secrets of Reverse Engineering*. Wiley Publishing, Inc., 2005. ISBN: 978-0-7645-7481-8.

[10] Dependency Walker. *Dependency Walker Help*. URL: http://www.dependencywalker.com/help/html/contents.htm (cit. 07. 09. 2018).

[11] T. M. MITCHELL. *Machine Learning*. McGraw-Hill Science/Engineering/Math, mar. 1997. ISBN: 978-0-07-042807-2.

[12] J. BROWNLEE. *Master machine learning algorithms*. Jason Brownlee, 2016.

[13] D. SONI. *Supervised vs. Unsupervised Learning*. URL: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d (cit. 04. 01. 2019).

[14] Z. XIAOJIN. "Semi-Supervised Learning Tutorial". In: Department of Computer Sciences University of Wisconsin, Madison, USA. 2007.

[15] V. KURAMA. *Reinforcement Learning with Python*. URL: https://towardsdatascience.com/reinforcement-learning-with-python-8ef0242a2fa2 (cit. 03. 12. 2018).

[16] V. SAVOV. *The OpenAI Dota 2 bots just defeated a team of former pros*. URL: https://www.theverge.com/2018/8/6/17655086/dota2-openai-bots-professional-gaming-ai (cit. 08. 12. 2018).

[17] MARR B. *The Top 10 AI And Machine Learning Use Cases Everyone Should Know About*. URL: https://www.forbes.com/sites/bernardmarr/2016/09/30/what-are-the-top-10-use-cases-for-machine-learning-and-ai/#1a38bcdf94c9 (cit. 14. 11. 2018).

[18] Kaspersky Lab. *Kaspersky Lab is Detecting 325,000 New Malicious Files Every Day*. URL: https://www.kaspersky.com/about/press-releases/2014_kaspersky-lab-is-detecting-325000-new-malicious-files-every-day (cit. 14. 11. 2018).

[19] Yara UK Limited. *N-Sensor - to variably apply nitrogen*. URL: https://www.yara.co.uk/crop-nutrition/farmers-toolbox/n-sensor/ (cit. 18. 11. 2018).

[20] J. GRUS. *Data Science from Scratch*. OReilly Media, 2015. ISBN: 978-1-491-90142-7.

[21] A. SMOLA a S.V.N. VISHWANANTHAN. *Introduction to Machine Learning*. Press Syndicate of the University of Cambridge, 2008. ISBN: 0 521 82583 0.

[22] G. BIAU. "Analysis of a Random Forests Model". In: *Journal of Machine Learning Research* 13 (2012).

[23] S. THIRUMURUGANATHAN. *A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm*. URL: https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/ (cit. 08. 12. 2018).

[24] N. FENTON. *Bayes rule*. URL: https://www.eecs.qmul.ac.uk/~norman/BBNs/Bayes_rule.htm (cit. 28. 11. 2018).

[25] S. S. WARREN. "Neural Networks and Statistical Models". In: SAS Institute Inc. Apr. 1994.

[26] C. M. BISHOP. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, 2006. ISBN: 978-0387-31073-2.

[27] J.Z. KOTLER a M. A. MALOOF. "Learning to Detect and Classify Malicious Executables in the Wild". In: *Journal of Machine Learning Research* 7 (jún 2006).

[28] M. G. SCHULTZ, E. ZADOK a S. J. STOLFO. "Data Mining Methods for Detection of New Malicious Executables". In: IEEE Xplore. Apr. 2001.

[29] M. SIDDIQUI, M. C. WANG a J. LEE. "Detecting Internet Worms Using Data Mining Techniques". In: *Journal of Systemics* 6 (2009).

[30] H. TAMADA et al. "Dynamic software birthmarks based on API calls". In: *IEICE Transactions on Information and Systems*. 8. ed. 89 (2006).

[31] M. SIKORSKI a A. HONIG. *PRACTICAL MALWARE ANALYSIS*. William Pollock, 2012. ISBN: 978-1-59327-290-6.