FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Simulace detekčního modelu škodlivého kódu
Student:	Bc. Libor Šlechta
Vedoucí:	Mgr. Martin Jureček
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je ověřit, jestli jsme schopni simulovat neznámý detekční model škodlivého kódu pomocí jiného modelu s odlišnou sadou příznaků a se znalostí výsledků klasifikace neznámého modelu. První model s neznámými příznaky bude použitý pro klasifikaci různých vzorků (EXE souborů). Tyto výsledky budou použity jako označení vzorků pro trénování druhého modelu používajícího jinou sadu příznaků (ale stejné EXE soubory). Pro simulaci prvního modelu se předpokládá využití různých metod strojového učení.

Pokyny:

1) nastudovat metody strojového učení vhodné pro detekci malware
2) vykonat výběr příznaků na předem dané sadě příznaků
3) implementovat uvedené algoritmy a vyhodnotit jejich přesnosti simulací
4) pokusit se vylepšit výsledky využitím dodatečných trénovacích dat obsahujících správně označené vzorky

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

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

V Praze dne 1. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Simulace detekčního modelu škodlivého kódu

Bc. Libor Šlechta

Katedra informační bezpečnosti
Vedoucí práce: Mgr. Martin Jureček

27. června 2019

Poděkování

Rád bych poděkoval především vedoucímu práce Mgr. Martinu Jurečkovi za přínosné konzultace, které mi velmi pomohly při tvorbě diplomové práce. Poděkování patří také celé mé rodině a přítelkyni, kteří mě podporovali a to nejen po dobu této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. června 2019 .....

Odkaz na tuto práci

Šlechta, Libor. *Simulace detekčního modelu škodlivého kódu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Množství škodlivých programů neustále stoupá a útočníci stále přicházejí s novými technikami, kterými se snaží oklamat používané detekční metody. Tato práce se zabývá automatickou detekcí škodlivého kódu pomocí algoritmů strojového učení. Hlavním rozdílem oproti podobným pracím z tohoto oboru je provedený pokus, který se zaměřuje na napodobení naučeného klasifikátoru. K napodobování klasifikátoru byly vybrány čtyři algoritmy strojového učení. Neuronové sítě, K-nejbližších sousedů, Rozhodovací stromy a Naivní Bayesův klasifikátor. Napodobovaný model se podařilo napodobit nejlépe pomocí varianty neuronových sítí. Výsledná vícevrstvá perceptronová síť dosáhla přesnosti 98,68 %.

Klíčová slova automatická klasifikace malware, strojové učení, simulace modelu, výběr příznaků, neuronové sítě, scikit-learn, python

Abstract

The number of harmful programs is still rising, and attackers invent new techniques to avoid detection every day. This thesis focuses on automatic malware classification by using machine learning algorithms. The main difference from other work in this field of study is experiment that mimics behavior of a detection model. Four machine learning algorithms were used for this experiment. Neural networks, K-nearest neighbors, Decision trees and Naive Bayes classifier. The best result was achieved by using a variant of neural network. Multilayer perceptron network simulated the model with accuracy of 98,68 %.

Keywords automated malware classification, machine learning, simulation of model, feature selection, neural networks, scikit-learn, python

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vývoj v oblasti informačních technologií pokračuje v posledních letech neustále velkým tempem kupředu. Elektronika usnadňuje lidstvu dnes již téměř všechny každodenní úkony. Své využití najde ve většině zaměstnání, při finančních transakcích i při poskytování zábavy.

Digitalizace financí a osobních údajů však také přináší svá rizika. Snadný a rychlý přístup spolu s možností nakládat s těmito zdroji bez nutnosti fyzického přístupu láka kriminálníky k zneužití těchto technologií.

V dnešní době počítačová kriminalita není vůbec neobvyklým jevem. Proto je otázka kybernetické bezpečnosti důležitější než kdy jindy. Způsobů, jakými se snaží útočníci uškodit nebo přijít k penězům, je nespočet. Jedním z přístupů je tvorba škodlivých programů, též nazývaných malware.

Ohromný nárůst množství malware, který je umocněn také modifikací a obfuskací škodlivých kódů, přinesl velkou výzvu pro společnosti zabývající se počítačovou bezpečností. Objemy dat, které tyto společnosti sbírají každý den, vysoce přesahují množství, které by se dalo analyzovat ručně.

Řešením tohoto problému je automatické rozhodování. Klasický přístup pomocí předem definované sady pravidel není příliš efektivní. Nalézt přesnou definici není snadné a díky malé modifikaci může být antivirový program snadno oklamán. Alternativně lze klasifikování založit na metodách strojového učení.

Strojové učení je obor umělé inteligence, který je v současné době užíván napříč širokým spektrem oborů. Jednou z nejvýznamnějších oblastí, kde se strojové učení uplatňuje, je oblast rozpoznávání obrazů. Zde díky této technice došlo k výraznému průlomu v úspěšnosti klasifikace. Užívá se však úspěšně i při detekci a třídění malware.

Algoritmy strojového učení pracují nad příznaky získanými z dat. V případě detekce malware existují dva přístupy k extrakci těchto vlastností. Statická analýza a dynamická analýza. Při statické analýze jsou extrahovány informace ze spustitelných souborů, jako jsou sekvence kódu, informace o importo-

vaných knihovnách a spoustu dalších. Dynamická analýza oproti tomu zkoumá skutečné chování programů. S ohledem na bezpečnost bývá pro tyto potřeby použit emulátor virtuálního prostředí, ve kterém se programy spouštějí.

Hlavní myšlenkou strojového učení je, že programy neobsahují od tvůrce přímo definovaná pravidla, jak se mají rozhodovat. Mají pouze stanovený postup, pomocí nějž mají dospět k rozhodovacím pravidlům. Jinými slovy se učí samy rozhodovat. Obdobně jako u lidského učení je klíčová zejména schopnost generalizace, tedy na základě několika příkladů být schopen správně reagovat na podobné, ale doposud neviděné příklady.

Kapitola 1

Strojové učení

Obsahem této kapitoly je úvod do problematiky strojového učení. Představuje důležité pojmy a prezentuje některé významné algoritmy, které se v tomto oboru používají.

1.1 Důležité pojmy

V textu práce jsou užívány pojmy, jak je lze najít ve většině odborné literatury. Následuje přehled a vysvětlení těchto pojmů.

- Příznak – Vlastnost, kterou lze získat z dostupných dat. V anglickém jazyce bývá označován jako feature.
- Model – Po vybrání algoritmu, který je schopný řešit zadaný problém, je zkonstruován model. Ten popisuje konkrétní implementaci algoritmu.
- Trénování modelu – Nebo také učení modelu je proces, při němž dochází k nastavení vnitřních parametrů za pomoci trénovacích dat.
- Testování modelu – Vyhodnocení úspěšnosti modelu probíhá na testovacích datech. U těchto dat jsou známy správné výstupy, které jsou porovnávány s výstupy modelu. Testovací data nesmí být použita při trénování modelu.

1.2 Dělení algoritmů dle řešeného problému

Strojové učení lze uplatnit při řešení více druhů problémů. Následující dělení a příklady jsou převzaty z kurzu [1] od společnosti Google.

- Klasifikace – Zařaď vzorek do jedné z předem daných kategorií.
- Regrese – Předvídej číselné hodnoty.

- Shlukování – Sdruž vzorky do skupin dle vzájemných podobností.
- Řešení asociací – Odhal pravděpodobné souvislosti v získaných datech.
- Strukturovaný výstup – Vytvoř komplexní výstup. Příkladem je tvorba derivačních stromů pro věty z lidského jazyka nebo hledání ohraničujících čtverců při rozpoznávání obrazu.
- Hodnocení – Urči pozici na škále hodnocení. Například hodnocení kvality výsledků vyhledávání.

1.3 Druhy učení

V závislosti na použitém modelu dochází k učení pomocí různých metod. Ayodele [2] popsal ve své práci následující rozdělení pomocí nějž může k učení docházet.

- Učení s učitelem (Supervised learning) – V případě tohoto druhu učení je nutné mít data u kterých známe požadovaný výstup. Metoda je vhodná zejména při řešení problému klasifikace, protože zadaná data chceme rozdělit do předem známých kategorií.
- Učení bez učitele (Unsupervised learning) – Tento příklad učení je přímým protikladem. Při učení programu nepředkládáme žádná označená data. Nejvýznamnější postupy, které lze při tomto druhu učení využít jsou dva. První spočívá v systému odměn, kdy program při správném vyhodnocení dostává lepší hodnocení než v případě neúspěchu. A druhou možností je takzvané shlukování, kdy se program snaží najít v datech souvislosti a data rozdělit do skupin tak, aby skupiny odpovídaly lidské intuici.
- Semi-supervised learning – Tato metoda učení je kombinací prvních dvou přístupů. Využívá označených i neoznačených dat.
- Reinforcement learning – Reinforcement learning je technika při níž je učení ovlivňováno předchozími rozhodnutími. Nejčastěji je uplatňováno pro problémy rozhodování. Model se pak učí na základě zpětné vazby od vnějšího prostředí. Příkladem může být snaha vyvolat určitou událost. Pokud událost nastane v souvislosti s určitými akcemi, dojde k posílení tohoto chování.
- Transduction – Při tomto druhu učení nedochází ke konstrukci explicitní rozhodovací funkce. Model se snaží rozhodovat na základě trénovacích vstupů, jejich označení a nových vstupů.

- Learning to learn – Pro potřeby učení agentů, kteří jsou schopni řešit více problémů a dokáží rychle reagovat na nové situace, bylo třeba vytvořit nový přístup k učení strojů. Při tomto přístupu se program v průběhu učení snaží zlepšit svoji schopnost učit se a generalizovat.

1.4 Vyhodnocení úspěšnosti modelu

Pro potřeby kvantifikování kvality modelů existuje více přístupů. V případě binární klasifikace vychází většina metrik z tabulky záměn. Tato tabulka je vytvořena za pomoci výstupů klasifikátoru a očekávaných výstupů. Popis matice záměn je uveden v tabulce 1.1. Ve vztahu k automatické klasifikaci malware odpovídají pojmy TP, FP, FN a TN následujícím případům. TP: vzorek je malware a klasifikátor ho správně označil jako škodlivý. FP: vzorek není malware, ale klasifikátor ho označil za škodlivý. FN: vzorek je malware, ale klasifikátor ho označil jako benigní. TN: legitimní soubor byl správně označený jako benigní.

Tabulka 1.1: Popis položek tabulky záměn

		Správná klasifikace	
		positive	negative
Výstup klasifikátoru	positive	# True positive (TP)	# False positive (FP)
	negative	# False negative (FN)	# True negative (TN)

1.4.1 Precision a recall

Jeden z přístupů, který se zaměřuje na positivní třídu, spočívá ve spočtení hodnot označovaných jako precision a recall. Precision lze vypočíst pomocí vzorce (1.1). Při klasifikování malwaru, udává precision pravděpodobnost, že vzorek označený jako malware, je skutečně škodlivý. Recall, též označovaný jako citlivost (sensitivity), je definovaný vzorcem (1.2). Tato hodnota odpovídá pravděpodobnosti, že škodlivý soubor bude klasifikátorem detekován.

$$Precision = \frac{TP}{TP + FP} \tag{1.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{1.2}$$

Je důležité přihlížet k výsledkům obou hodnot, protože obě hodnoty samostatně nemají velkou vypovídající hodnotu. Příkladem může být extrémní případ, kdy klasifikátor určuje všechny vzorky jako malware a dosahuje citlivosti 100 %, ale díky vysokému počtu planých poplachů bude mít nízkou precision hodnotu.

1.4.2 Accuracy

Další metrikou, kterou lze k vyjádření úspěšnosti binární klasifikace použít, je označována jako accuracy. Accuracy počítáme pomocí vzorce (1.3) a odpovídá procentu všech správně klasifikovaných vzorků.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.3}$$

Accuracy počítaná pomocí vztahu (1.3) je zkreslená v případě nevyvážené testovací datové sady, a proto bývá doporučováno používat ballanced accuracy, jež lze spočítat pomocí vzorce (1.6). Ballanced accuracy závisí na true positive rate a true negative rate. True positive rate odpovídá hodnotě uvedené vzorcem (1.4) a je totožný s hodnotou recall. True negative rate je vypočten pomocí vztahu (1.5). V případě vyvážená datové sady dává ballanced accuracy stejnou hodnotu jako accuracy počítaná dle vzathu (1.3)

$$TPR = \frac{TP}{TP + FN} \tag{1.4}$$

$$TNR = \frac{TN}{TN + FP} \tag{1.5}$$

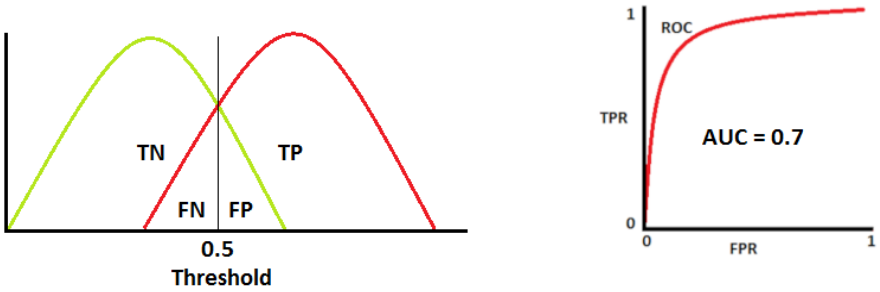
$$BallancedAccuracy = \frac{TPR + TNR}{2} \tag{1.6}$$

1.4.3 Receiver operating characteristic

Většina klasifikátorů predikuje pravděpodobnosti příslušnosti vzorku do dané třídy. Závislost úspěšnosti klasifikace a mezní hodnoty, která odděluje náležitost k jedné z daných tříd, lze vyjádřit pomocí receiver operating characteristic. ROC křivka je graf, který znázorňuje pravděpodobnosti správně klasifikovaných positivních vzorků (1.4) na vertikální ose vůči pravděpodobnosti špatně klasifikovaných vzorků (1.7) na horizontální ose, v závislosti na měnící se hodnotě hraniční hodnoty.

$$FPR = \frac{FP}{FP + TN} \tag{1.7}$$

Výslednou kvalitu modelu lze pak vyjádřit pomocí obsahu plochy pod ROC křivkou, který se označuje AUC a udává separabilitu tříd, viz obrázek 1.1.



Obrázek 1.1: Příklad distribučních funkcí dvou tříd a tomu odpovídající ROC křivka[3].

1.5 Algoritmy vhodné k detekci malware

Detekce malware je problém, při němž je především důležité třídit vzorky do předem známých kategorií. Na základě dělení popsaném v sekci 1.2 se jedná tedy o problém klasifikace. Konkrétně chceme hlavně znát, jestli je vzorek škodlivý, nebo legitimní. Případně škodlivé vzorky lze ještě seskupovat do skupin podle jejich funkce. A to především proto, že většina škodlivého software je modifikována a obfuskována [4]. Tyto skupiny pak nazýváme malwarové rodiny.

S ohledem na studium předchozích prací, zejména práce Malware Detection Using Machine Learning [5], byly vybrány následující algoritmy.

- Neuronové sítě (Neural networks) – Tento algoritmus je inspirovaný biologickými neuronovými sítěmi v lidském mozku. Při učení modelu dochází k úpravě konstant, které reprezentují sílu spojení mezi neurony. Výstup je vybrán podle výsledných hodnot výstupních neuronů. Bližší představení neuronových sítí lze najít v kapitole 2.

- Metoda podpůrných vektorů (Support vector machines) – Model získaný touto metodou reprezentuje hranici mezi daty. Při učení modelu je hledána nadrovina, která nejlépe rozděluje trénovací data. Nejlepší rozdělení dat je takové, při němž je vzdálenost hraničních bodů od hranice největší.

- K-nejbližších sousedů (k nearest neighbors) – Jedná se o jednoduchý algoritmus, který při klasifikaci vzorku najde parametrem daný počet nejbližších sousedů ve více dimenzionálním prostoru. Dimenze prostoru n je rovna počtu příznaků. Jednou z metrik používaných k vypočtení vzdálenosti dvou bodů x, y je Eukleidovská metrika, kterou lze vypočítat dle vzorce (1.8).

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1.8}$$

Vzorek je zařazen do té třídy, jenž se vyskytuje mezi nalezenými sousedy nejčastěji. Pro správnou funkci algoritmu je požadováno, aby počet nejbližších sousedů bylo liché číslo, které není násobkem počtu tříd. Pokud by tomu tak nebylo, mohlo by dojít k nerozhodnému počtu hlasů při rozhodování, do které třídy vzorek zařadit.

- Rozhodovací strom (Decision tree) – Rozhodovací strom je složen z vnitřních uzlů a listů. Při klasifikování vzorku postupuje algoritmus rekurzivně od kořene stromu. Vnitřní uzly obsahují podmínky, podle kterých se algoritmus rozhoduje, do kterého podstromu má pokračovat. Listy pak obsahují označení třídy, do které vzorek patří. Při konstruování stromu se postupuje rovněž od kořene a vždy je vybrán příznak, který nejlépe rozděluje zbývající data. Způsobů, jakými lze kvantifikovat kvalitu rozdělení existuje více. Jedna z používaných funkcí se nazývá Gini Impurity. V případě rozdělení na dvě podmnožiny se vypočte nejprve funkce pro obě podmnožiny zvlášť dle vzorce (1.9).

$$g = \sum_{i=1}^{C} P(i) \times (1 - P(i)) \tag{1.9}$$

Kde C odpovídá počtu všech možných tříd a $P(i)$ označuje pravděpodobnost příslušnosti datového bodu do třídy i v této podmnožině.

Výsledná hodnota kvality rozdělení je následně spočtena podle vztahu (1.10).

$$G = P(A) \times g_A + P(B) \times g_B \tag{1.10}$$

A a B značí podmnožiny, které vznikly rozdělením původní množiny, $P(A)$ a $P(B)$ jsou pravděpodobnosti, že prvek bude náležet do příslušné podmnožiny a g_A společně s g_B odpovídá hodnotám funkce Gini.

- Naivní Bayesův klasifikátor (Naive Bayes classifier) – Klasifikování v případě tohoto přístupu je založeno na použití Bayesova teorému. Silným předpokladem, který tato metoda požaduje, je podmíněná nezávislost všech příznaků. Při trénování modelu se spočítají pravděpodobnosti výskytu každé třídy, pravděpodobnosti všech příznaků a podmíněné

pravděpodobnosti všech příznaků vzhledem ke všem třídám. Při klasifikaci se pak na základě těchto hodnot spočítají pravděpodobnosti, že vzorek patří do třídy y podmíněné hodnotami příznaků x_1 až x_n dle vzorce (1.11).

$$P(y|x_1, \ldots, x_n) = \frac{P(x_1|y) \times P(x_2|y) \times \cdots \times P(x_n|y) \times P(y)}{P(x_1) \times P(x_2) \times \cdots \times P(x_n)} \quad (1.11)$$

Vzorek je pak zařazen do té třídy, u níž je největší hodnota této podmíněné pravděpodobnosti.

Kapitola 2

Neuronové sítě

Kapitola čerpá především ze zdrojů [6, 7].

Uvažovány jsou hlavně dopředné neuronové sítě. Tento typ sítí neobsahuje žádné smyčky a zpětné propagace. Vyhodnocování probíhá od vstupní vrstvy a výsledky se postupně propagují až do výstupní vrstvy.

2.1 Neuron a perceptron

Neuron, jenž je definovaný na obrázku 2.1, je základním stavebním prvkem neuronových sítí. Neuron může být vstupní, výstupní a skrytý. Pokud hodnota neuronu odpovídá hodnotě příznaku, pak jej nazýváme vstupní neuron. Hodnota výstupního neuronu se podílí na určení výstupu sítě. Jako skryté neurony jsou označovány ty, jenž nejsou vstupní ani výstupní. V určitých variantách sítí může neuron plnit funkci vstupního i výstupního neuronu zároveň.

Perceptron je jednou z konkrétních implementací formálního neuronu. Vnitřní potenciál je počítán jako vážená suma vstupů, viz (2.1).

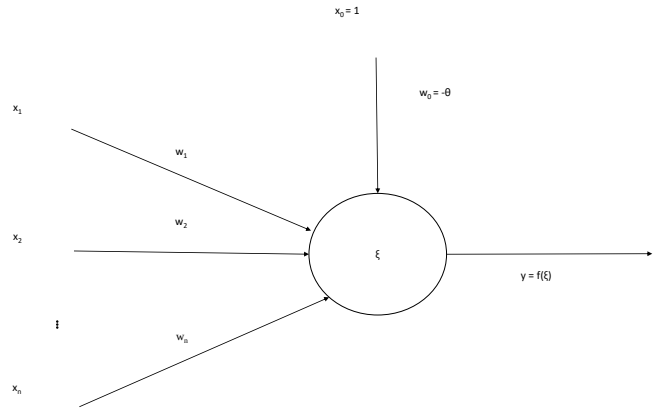
$$\xi = \sum_{i=0}^{n} w_i \times x_i \tag{2.1}$$

Při definování perceptronu byla jako aktivační funkce použita sigmoida, jejíž předpis udává vzorec (2.2), kde λ je parametr strmosti.

$$f(\xi) = \frac{1}{1 + e^{-\lambda \times \xi}} \tag{2.2}$$

V současné době je však velmi populární používat místo sigmoidy funkci ReLU, která se počítá dle vztahu uvedeného vzorcem (2.3).

$$f(\xi) = \max(0, \xi) \tag{2.3}$$



Obrázek 2.1: Formální neuron, $x_1, \ldots, x_n$ jsou vstupy, $w_1, \ldots, w_n$ jsou váhy spojů, $x_0 = 1$ formální vstup, θ práh, $w_0 = -\theta$ bias, ξ vnitřní potenciál a $y = f(\xi)$ výstup neuronu získaný aplikací aktivační funkce na potenciál[6].

2.2 Vícevrstvé perceptronové sítě

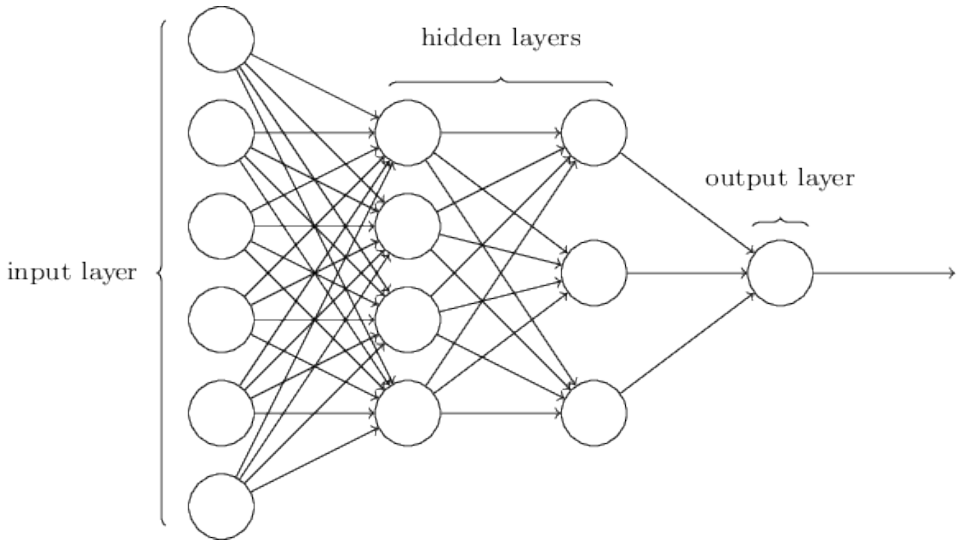
Jednou z nejčastěji používaných variant neuronových sítí jsou více vrstvé perceptronové sítě a jejich varianty. Základní vícevrstvá perceptronová síť se skládá z jedné vrstvy vstupních perceptronů, jedné vrstvy výstupních a volitelného počtu skrytých vrstev. Pro každý perceptron platí, že je spojen se všemi neurony předchozí vrstvy. Krom vstupů je každý neuron ovlivněn ještě odchylkou. Příklad struktury vícevrstvé perceptronové sítě lze vidět na obrázku 2.2.

2.3 Konvoluční neuronové sítě

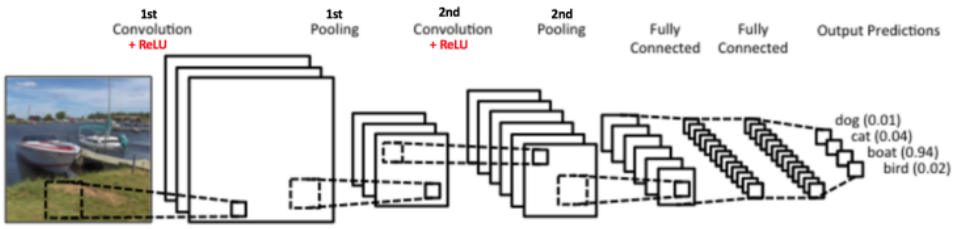
Největší rozvoj použití konvolučních neuronových sítí byl zapříčiněn jejich úspěšným nasazením na problému klasifikace obrazu. V poslední době však byly konvoluční neuronové sítě úspěšně aplikovány i na problém detekce malware. Příkladem může být například práce Empowering convolutional networks for malware classification and analysis [8] a Malware Detection with Neural Network Using Combined Features [9].

Konvoluční neuronové sítě se skládají ze tří hlavních komponent. Z dříve představené vícevrstvé perceptronové sítě, konvoluční vrstvy a sdružovací vrstvy. Konvolučních a sdružovacích vrstev může být i více. Příklad struktury sítě, která by mohla být použita pro rozpoznávání obrazu lze najít na obrázku 2.3.

Konvoluce probíhá pomocí filtrů. Filtr je aplikován pomocí posuvu. Počet filtrů je nazýván hloubka konvolučního kroku. Výsledným produktem aplikování každého filtru je mapa příznaků. Účelem této vrstvy je extrahovat



Obrázek 2.2: Vícevstvá perceptronová síť[7]



Obrázek 2.3: Konvoluční neuronová síť[10]

příznaky. Součástí je také aplikace nelineární funkce ReLU na každou výslednou mapu.

Sdružování, anglicky pooling, snižuje dimenzi map příznaků. Existuje více druhů sdružování. Maximální, průměrové, součtové a další.

Maximální sdružování probíhá tak, že se nejprve zvolí velikost prostorového sousedství. Mapa se rozdělí na sousedství a z každého je následně vybrán největší prvek.

V případě návrhu neuronové sítě je třeba zvolit počet a pořadí konvolučních a sdružovacích vrstev. Neplatí, že po každé konvoluční vrstvě musí následovat sdružovací vrstva. Dále je třeba určit velikost filtru a hloubku každé konvoluční vrstvy, typ sdružovacích vrstev a navrhnout plně spojenou vícevrstvou neuronovou síť. Hodnoty filtrů a váhy jednotlivých spojení se model učí na základě trénovacích dat.

2.4 Back propagation

Algoritmus, který bývá použit k trénovaní neuronových sítí se nazývá back propagation. V prvním kroku tohoto algoritmu jsou parametry sítě vygenerovány náhodně. Poté je provedena klasifikace na trénovacích datech a spočtena chyba od očekávaných hodnot. Parametry sítě jsou upraveny tak, aby došlo k největšímu snížení chyby. O kolik a který parametr se má upravit je určeno pomocí gradientu. Gradient je vektor, který udává směr nejrychlejšího růstu funkce. Záporný gradient určuje, jaké proměnné se mají o kolik pozměnit, aby došlo k největšímu zlepšení.

2.5 Schopnost napodobit libovolnou funkci

Výpočetní síla neuronových sítí je zkoumána již delší dobu. Shrnutí výzkumů z oblasti teorie aproximace funkcí lze najít v knize Umělá inteligence [11]. Jsou zde uvedeny matematické důkazy, že více typů dopředných neuronových sítí je schopno aproximovat libovolné zobrazení v podmnožinách vícedimenzionálních euklidovských prostorů, a to dokonce s požadovanou přesností. Toto tvrzení je podmíněno tím, že síť obsahuje dostatečný počet neuronů.

2.5.1 Aproximace v normovaných lineárních prostorech

Aproximace funkcí bývá provedena na vhodných normovaných lineárních prostorech funkcí. Existují dva druhy aproximací. Lineární aproximace a nelineární aproximace. Teorie lineární aproximace se zabývá takovými aproximacemi, kde množina aproximujících funkcí tvoří lineární podprostor. V opačném případě mluvíme o teorii nelineární aproximace.

Podmnožinu normovaného lineárního prostoru označujeme pojmem *hustá*, pokud jsme schopni pomocí této podmnožiny aproximovat všechny funkce z původního prostoru s libovolnou přesností.

2.5.2 Rychlost aproximace

Rychlost aproximace se obvykle udává v počtu potřebných výpočetních jednotek, tedy počtu neuronů sítě. Při lineární aproximaci nastává exponenciální závislost na počtu proměnných pro mnoho množin funkcí. V případě nelineární aproximace bývá složitost znatelně nižší.

2.5.3 Aproximace pomocí neuronových sítí

Aproximování funkcí pomocí neuronových sítí spadá do teorie nelineární aproximace.

Třída neuronových sítí má *univerzální aproximační vlastnost*, pokud jsme schopni pomocí této třídy neuronových sítí aproximovat s požadovanou přesností

libovolnou funkci, která se může vyskytovat v aplikacích. Univerzální aproximační vlastnost lze matematicky definovat pomocí pojmu hustoty.

Univerzální aproximační vlastnost byla mimo jiné dokázána pro vícevrstvé perceptronové sítě obsahující jednu skrytou vrstvu neuronů. Předchozí vlastnost platí pro libovolnou aktivační funkci s výjimkou polynomů.

Kapitola 3

Výběr příznaků

Při sběru údajů o pozorovaném jevu se často stává, že jsou určité informace duplicitní nebo mají velmi malou až nulovou vypovídající hodnotu. Algoritmy zaměřené na výběr příznaků se snaží snížit dimenzi reprezentovaných dat odstraněním těchto duplicitních a irelevantních informací. Přínosem snížení dimenze příznaků je kromě snížení objemu dat také snížená komplexnost modelu, který se pomocí těchto dat učí. To vede k tomu, že se model učí rychleji a je schopný lépe generalizovat. Schopnost generalizovat je velmi důležitá, protože díky ní je model schopný správně reagovat na doposud neoznačená data, což je hlavním smyslem použití strojového učení.

Algoritmy pro výběr příznaků jsou děleny do dvou skupin podle přístupu, pomocí kterého se snaží dosáhnout redukce dimenze. První skupina algoritmů se nazývá selekce příznaků a druhá extrakce příznaků.

Dělení algortimů bylo spolu s jejich popisy nastudováno z knihy Statistical Pattern Recognition [12].

3.1 Selekce příznaků

Výsledkem selekce je nejlepší podmnožina velikosti d z p příznaků. Metrika, podle které je určena kvalita podmnožiny, se nazývá účelová funkce (criterion function). Existují dva přístupy k definování této funkce.

1. Experimentální vytvoření klasifikátoru. Výsledná sada příznaků je závislá na zvoleném klasifikátoru. Hodnota funkce, pak může být například závislá na očekávané chybovosti.

2. Statistické vyhodnocení kvality příznaků.

K provedení selekce jsou známy jak optimální, tak sub optimální algoritmy. Příkladem optimálního algoritmu je Branch and bound procedure.

Algoritmus lze použít, pokud pro funkci kritérií platí vlastnost monotonie, tedy je pravda, že platí následující výrok. Pokud pro dvě množiny příznaků X, Y platí $X \subset Y$, pak hodnota účelové funkce pro X je menší než pro Y.

Algoritmus nejprve vytvoří strom všech možností podmnožin. V kořenu stromu je množina všech příznaků. V každé další úrovni stromu jsou pak podmnožiny jejichž velikost je o jedna menší než předchozí velikost množiny. Strom má takovou hloubku, aby v listech byly obsaženy podmnožiny požadované velikosti.

Hledané řešení je nalezeno postupným průchodem vytvořeného stromu a postupného vyhodnocování funkce kritérií. Při volbě potomka je volen takový uzel, který obsahuje nejméně potomků. Jakmile dojde k vyhodnocení prvního listu, uloží se tato hodnota jako dosavadní maximum. Při následném procházení vnitřních uzlů se porovnává získaná hodnota s dosavadním maximem. V případě, že je tato hodnota menší, nemá v této větvi smysl pokračovat díky vlastnosti monotonie.

Pokud neplatí vlastnost monotonie nebo jsou nadměrné výpočetní nároky na provedení algoritmu Branch and bound, musí být zvolen sub optimální algoritmus. Tyto algoritmy výměnou za sníženou časovou náročnost negarantují dosažení nejlepšího možného řešení.

- Best individual N – V prvním kroku tohoto algoritmu je spočtena účelová funkce zvlášť pro každou proměnou. Následně jsou příznaky seřazeny podle těchto hodnot a je vybrán požadovaný počet příznaků s nejvyššími hodnotami. Tento postup dává kvalitní výsledky pouze v případě, že vlastnosti nejsou vzájemně korelované.

- Sequential forward selection – Algoritmus je inicializován s prázdnou množinou řešení. V každém kroku je do množiny řešení přidán doposud nepřidaný příznak, který maximalizuje hodnotu účelové funkce. Běh algoritmu končí pokud by přidání dalšího příznaku zhoršilo hodnotu účelové funkce, nebo pokud je dosažena maximální velikost podmnožiny.

- Generalised sequential forward selection – Běh algoritmu je totožný se sequential forward selection. Jedinou výjimkou je, že v každém kroku se do množiny řešení přidává místo jedné proměnné podmnožina doposud nezvolených příznaků. Maximální velikost přidávané podmnožiny je předem definovaná.

- Sequential backward selection – Princip algoritmu je podobný jako u sequential forward selection. Rozdílem je pouze počáteční stav množiny řešení, která v tomto případě obsahuje všechny příznaky. V každém kroku pak je odebírán příznak, díky kterému je dosáhnuto nejlepšího průběžného výsledku. Tedy ten, díky kterému dojde k nejmenšímu poklesu účelové funkce.

- Generalised sequential backward selection – Zobecněná varianta algoritmu sequential backward selection. Stejně jako v dříve představené variantě zobecněného algoritmu se liší pouze tím, že na místo odebírání jedné proměnné v každém kroku, lze odebrat podmnožinu příznaků až o předem definované maximální velikosti.

- Plus l – take away r selection – Algoritmus je kombinací sequential forward selection a sequential backward selection. V každém kroku je přidáno l a odebráno r proměnných. Počáteční stav řešení a pořadí, zda je nejdříve přidáváno, nebo odebráno záleží na velikosti parametrů. Pokud platí, že $l > r$ počáteční množina je prázdná. V každém kroku je nejprve přidáno l příznaků a následně jich je odebráno r. V případě, že $r > l$, pak počáteční množina obsahuje všechny příznaky. V každém kroku se nejdříve příznaky odebírají a teprve poté přidávají.

- Generalised plus l – take away r selection - Generalizovaná metoda přechozího algoritmu, která se opět liší pouze tím, že lze příznaky přidávat a odebírat ve větším počtu najednou během jednoho kroku. Metoda může být ještě rozšířena. Přidávání a odebírání lze realizovat ve více krocích. Konstantu l lze rozdělit na n sčítanců. Pro každý sčítance l_i je pak množina řešení rozšířena o l_i příznaků. Analogicky lze toto použít i při odebírání vlastností.

- Floating search methods - Tyto metody fungují podobně jako plus l – take away r selection algoritmus. I zde dochází k přidávání a odebírání příznaků. Tyto akce se však nestřídají po vykonání předem stanoveného počtu akcí. Přidávání a odebírání se střídá, pokud poslední akce přidání nebo odebírání splní určité podmínky. Například pokud je zlepšení menší než předem definovaná konstanta.

3.2 Extrakce příznaků

Extrakce příznaků je transformace, která bere v úvahu všechny původní příznaky. Po aplikování transformace je získán menší počet nových příznaků.

- Analýza hlavních komponent – Výsledkem této metody je nový ortogonální souřadný systém. Osy systému jsou seřazeny dle toho, jak se podílí na rozptylu v původních datech. V případě, že rozptyl není rozdělen rovnoměrně, lze data reprezentovat v novém systému bez významné ztráty informací. Tato metoda je dle Webba [12] používána nejčastěji. Podrobnější popis této metody lze najít v kapitole 4.

- Analýza faktorů – Statistická metoda, která hledá korelace mezi proměnnými. Výstupem je sada faktorů jejichž lineární kombinace popisují původní pozorované proměnné.

Kapitola 4

Analýza hlavních komponent

Analýza hlavních komponent je algoritmus, který provádí lineární dimensionální redukci příznaků. A to pomocí hledání nového ortogonálního systému souřadnic. Při sestavování nového systému souřadnic je vždy hledána taková nadrovina, která je kolmá na doposud vytvořené nadroviny a zároveň má maximální velikost sumy vzdáleností bodů zobrazených na tuto nadrovinu od počátku. Tím je v každém kroku nalezena taková osa, která má pro daná data největší rozptyl. Zároveň jsou díky tomuto postupu komponenty seřazeny sestupně podle jejich rozptylu. Dále při hledáni nadrovin jsou uvažovány pouze takové, které obsahují počátek původního souřadného systému. A to z toho prostého důvodu, že je třeba počítat vzdálenost zobrazených bodů od počátku původního souřadného systému.

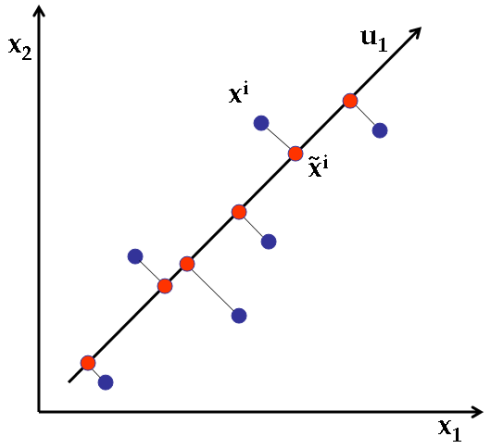
Obrázek 4.1 ilustruje nalezení jedné hlavní komponenty v dvoudimenzionálním prostoru. A transformování dat z dvoudimenzionálního prostoru do jednodimenzionálního při zachování maximální možné informace o rozptylu původních dat.

4.1 Standardizace dat

Pro správnou funkci tohoto postupu je nezbytné, aby data byla standardizovaná. A to především proto, že jednotlivé příznaky obvykle mívají velmi rozdílné hodnoty. Často se liší jak v průměru, tak v rozptylu dat. Ilustrativní příklad důležitosti standardizace dat je uveden v dokumentaci knihovny scikit[1]. Pokud sledujeme váhu a výšku lidí je intuitivně zřejmé, že rozdíl mezi vzorky lišící se o jedno kilo není tak významný, jako rozdíl jednoho metru.

Pro standardizaci dat existuje velké množství algoritmů. Jedním z velmi často používaných postupů je transformace dat tak, aby každý příznak měl hodnoty centrované okolo 0 s rozptylem 1. Popis tohoto postupu lze najít

[1] `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html`



Obrázek 4.1: Ukázka snížení dimenze dat použitím analýzy hlavních komponent[13].

v dokumentaci knihovny scikit[2]. Pro každý příznak je na základě jeho hodnot spočten průměr a směrodatná odchylka. Pokud označíme průměr příznaku u a směrodatnou odchylku s, pak transformace z hodnoty x je spočítána následovně $z = (x - u)/s$.

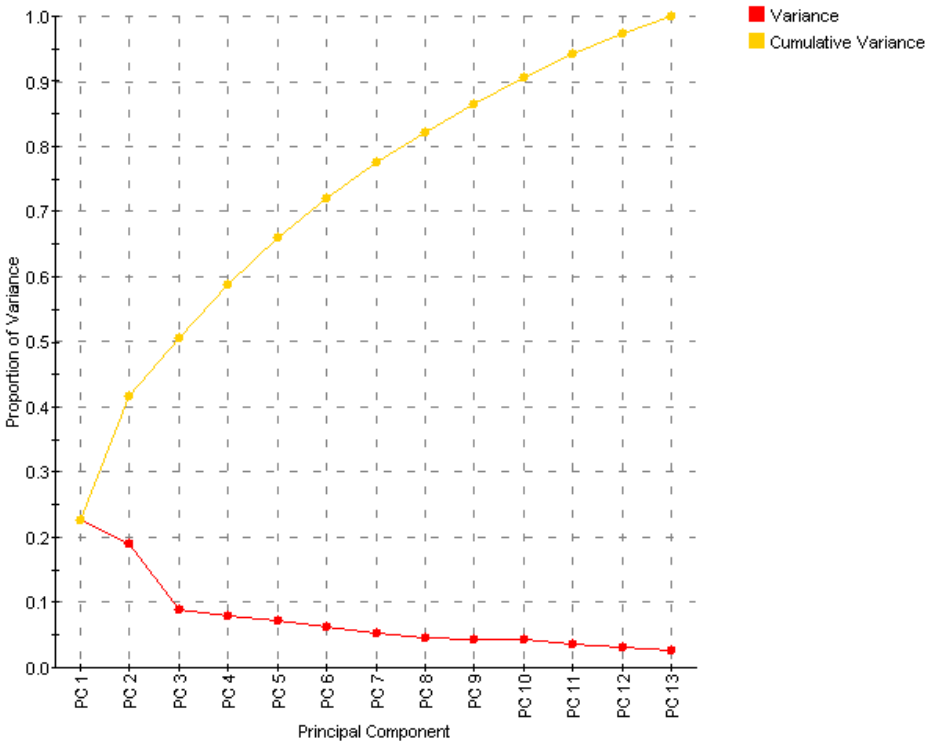
4.2 Určení počtu komponent

Kolik komponent se má použít pro transformaci dat je velmi důležitý parametr při provádění dimenzionální redukce za pomoci analýzy hlavních komponent. Při stanovení tohoto parametru se postupuje tak, že se nejprve vytvoří tolik hlavních komponent jako je dimenze původního systému. A spočte se variance všech komponent. Protože komponenty, které vznikly analýzou hlavních komponent, jsou seřazeny sestupně podle jejich variance, lze jednoduše vizualizovat kolik informace o původních datech zůstane zachováno při použití určitého počtu komponent.

Graf znázorňující tyto informace se nazývá scree plot. Příklad takového grafu lze najít na obrázku 4.2.

Tento přístup funguje optimálně v případě, že původní příznaky nejsou vzájemně korelované. V opačném případě může docházet k neoprávněnému přiřazení většího významu určitým komponentám.

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`



Obrázek 4.2: Příglad scree plotu[14].

Kapitola 5

Návrh experimentu

5.1 Napodobovaný model

K napodobení byl zvolen model, který používali autoři použité datové sady. Konkrétně se jednalo o variantu rozhodovacího stromu z knihovny lightGBM, která používá techniku zesílení gradientu. Při trénování modelu však nebyly použity všechny dostupné příznaky, ale byla provedena selekce 1400 příznaků pomocí algoritmu Best individual N, který byl popsán v sekci 3.1. Účelem tohoto kroku nebyla snaha o zlepšení napodobovaného modelu. Šlo především o to, aby napodobovaný model a modely, které se jej snaží napodobit, měli co nejvíce odlišné sady příznaků, na nichž byly natrénovány.

V dalším textu je tento model označován jako *model 1*.

5.2 Napodobující modely

Modely, které napodobují *model 1*, jsou v této práci označované jako *model 2*.

Při simulaci *modelu 1* byla k extrakci příznaků použita analýza hlavních komponent. Extrakce příznaků byla provedena vícekrát s různým počtem výsledných příznaků, aby bylo možno pozorovat, jak jejich počet bude ovlivňovat schopnost napodobit první model.

Algoritmy, které byly vybrány k tomuto pokusu, odpovídají těm, jenž byly představeny v sekci 1.5 věnované algoritmům vhodným k detekci malware. S výjimkou metody podpůrných vektorů, která byla vyřazena na základě údajů uvedených v dokumentaci používané knihovny. Dle dokumentace[3] není tato metoda vhodná v případě vysokého počtu trénovacích vzorků. Složitost učení modelů je více než kvadratická vůči počtu vzorků a doporučená maximální velkost trénovací sady je 10000. Používaná datová sada obsahovala sta tisíce vzorků a její bližší představení lze najít v sekci 6.1.1.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

Na doporučení vedoucího práce bylo zvažováno také použití konvolučních neuronových sítí, které byly popsány v sekci 2.3. Z časových důvodů však tato varianta neuronových sítí nakonec implementována nebyla.

5.3 Postup experimentu

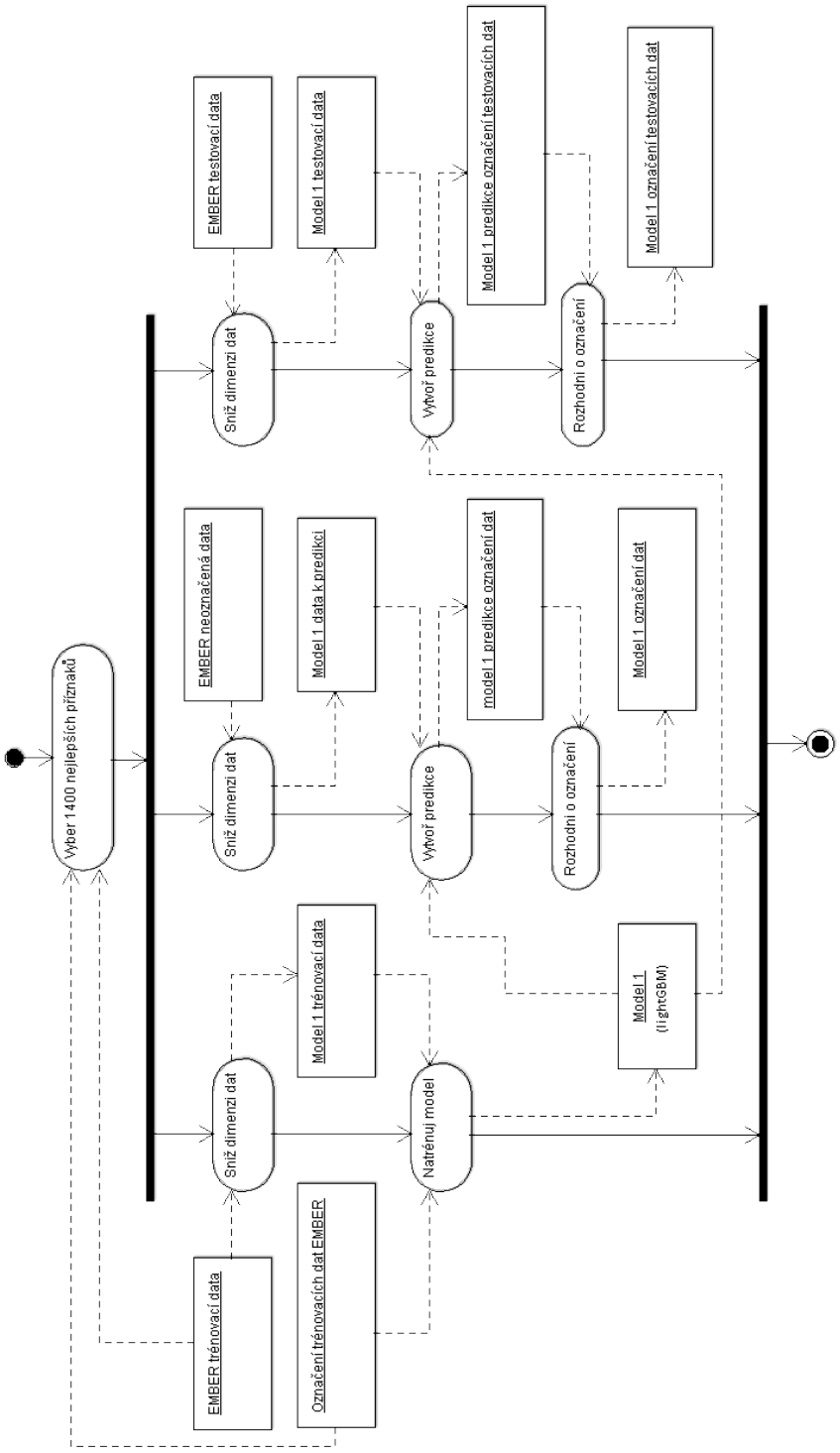
Postup experimentu je zachycen na UML diagramech aktivity 5.1 a 5.2. Pojmem EMBER se rozumí použitá datová sada viz 6.1.1.

První diagram popisuje postup, kterým byl vytvořen *model 1* a jeho výstupy. V prvním kroku této části pokusu byl nejprve vytvořen transformátor, který vybíral 1400 nejlepších příznaků. Následně došlo ke snížení dimenze všech dat z datové sady. Označená trénovací data byla následně použita k naučení *modelu 1*. Naučený *model 1* byl poté použit k predikci neoznačených dat a testovacích dat.

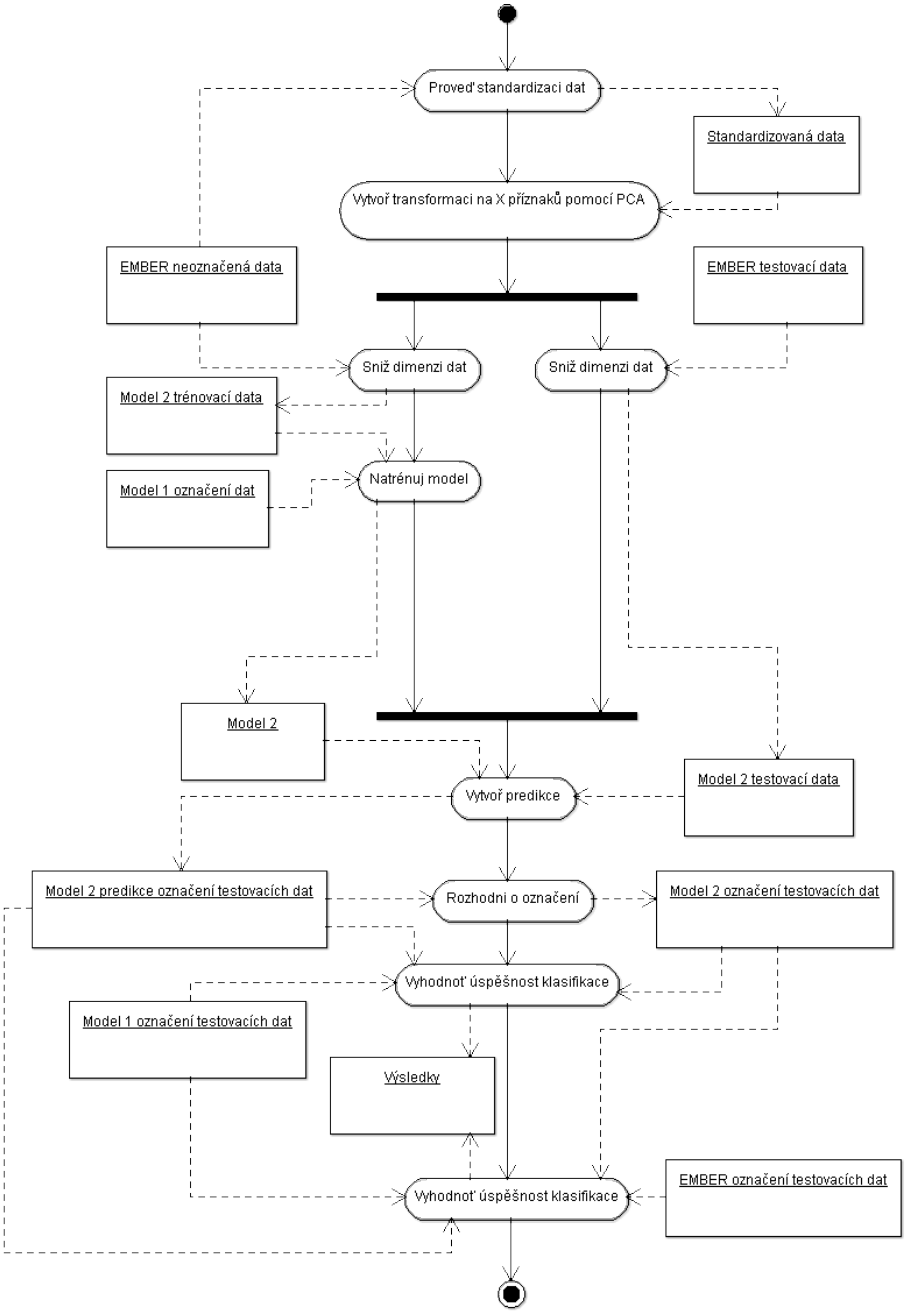
Jak bylo zmíněno dříve, extrakce příznaků byla prováděna vícekrát. Diagram 5.2 uvádí průběh napodobování *modelu 1* pro jednu hodnotu výstupní dimenze analýzy hlavních komponent. Tato hodnota je v diagramu označovaná jako X.

Napodobování *modelu 1* začínalo transformováním neoznačených trénovacích dat z datové sady EMBER. *Model 2* byl následně natrénován pomocí těchto transformovaných dat a ohodnocení, která vznikla použitím *modelu 1* na odpovídající vzorky v první části pokusu.

Vyhodnocení úspěšnosti *modelu 2* probíhalo na transformovaných testovacích datech. Výstupy *modelu 2* byly porovnávány s výstupy *modelu 1* a správnými označeními, které byly součástí použité datové sady.



Obrázek 5.1: Postup vytvoření modelu 1 a jeho výstupů



Obrázek 5.2: Postup vytvoření modelu 2 a vyhodnocení výsledků experimentu

Kapitola 6

Realizace experimentu

6.1 Použité nástroje

6.1.1 Datová sada

V počátečních fázích realizace této práce byla předběžně domluvena spolupráce s firmou ESET, která by mimo jiné znamenala i poskytnutí datové sady. K uzavření dohody bohužel nedošlo, a proto bylo třeba hledat alternativu.

Řešením se ukázala být datová sada EMBER [15]. Podle tvrzení autorů se jedná o jedinou volně dostupnou datovou sadu tohoto druhu. Toto tvrzení potvrdil i průzkum dostupných zdrojů. Bylo objeveno několik datových sad obsahující malware[4], ale žádná, která by obsahovala také vzorky neškodných programů.

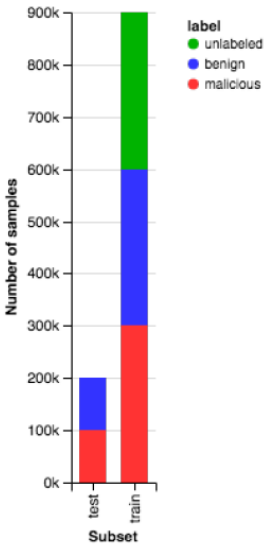
Autoři EMBER vytvořili kolekci obsahující informace o 1,1 milionech PE souborů. Každý vzorek je reprezentován hash otiskem původního souboru a vlastnostmi souboru. Většina vzorků má označení, zda je škodlivá, ale je zde i tři sta tisíc vzorků bez označení. Datová sada je již také předrozdělena na trénovací a testovací vzorky. Testovacích vzorků je v datové sadě dvě stě tisíc. Strukturu datové sady popisuje obrázek 6.1

Vlastnosti souborů jsou uloženy v textové podobě. K datové sadě je však dostupný program implementovaný v jazyce Python, který převádí tyto hodnoty na numerické a díky tomu lze snadno vytvořit numerický vektor příznaků, který je potřebný k trénování modelů.

6.1.2 Jazyk a důležité knihovny

Pro implementaci pokusu byl zvolen jazyk Python verze 3.6.5 a to především z důvodu dostupnosti knihoven vhodných pro práci na projektech obsahujících strojové učení. Dalším přínosem volby tohoto jazyka bylo také snadné napo-

[4] https://www.kaggle.com/c/malware-classification/data a https://virusshare.com



Obrázek 6.1: Počet vzorků v datové sadě EMBER[15]

jení na program EMBER a využití autory připraveného virtuálního prostředí vytvořeného pomoci technologie Conda.

Nejdůležitější použitou knihovou byla knihovna scikit-learn[16]. Tato volně dostupná knihovna pro jazyk Python obsahuje širokou řadu algoritmů strojového učení. Lze v ní najít také algoritmy užívané k předzpracování a dimenzionální redukci dat.

Druhou důležitou knihovou byla knihovna numpy. Jedná se také o volně dostupnou knihovnu jazyka Python. Tato knihovna je součástí SciPy ekosystému [17] a je určena pro numerické výpočty. V rámci této práce usnadnila především práci s velkými objemy dat, které se nevešly do operační paměti a bylo nutné je nahrávat z disku pouze průběžně po částech.

6.1.3 Použitý systém

Experiment byl proveden na stroji, který byl virtualizován za pomoci programu VirtualBox. Parametry tohoto stroje jsou uvedeny v tabulce 6.1.

Zprovoznit přiložené skripty by však mělo být možné i na méně výkonném stroji. Hlavním omezením k provádění výpočtů je velikost operační paměti. V případě selhání výpočtů kvůli nedostatku operační paměti lze ve skriptu snížit hodnoty používané u proměnné chunk_size, která nastavuje velikost použitých sekcí datových souborů.

Skripty byly napsány tak, aby nebyly závislé na použití Unixového operačního systému. Z nedostatku času však tato vlastnost nebyla otestována.

Tabulka 6.1: Parametry použitého stroje

Operační systém	Ubuntu 18.04
Velikost operační paměti	11264 MB
Procesor	Intel Core i7-7700HQ
Maximální frekvence procesoru	3.80 GHz
Počet jader procesoru	4

6.2 Výběr příznaků

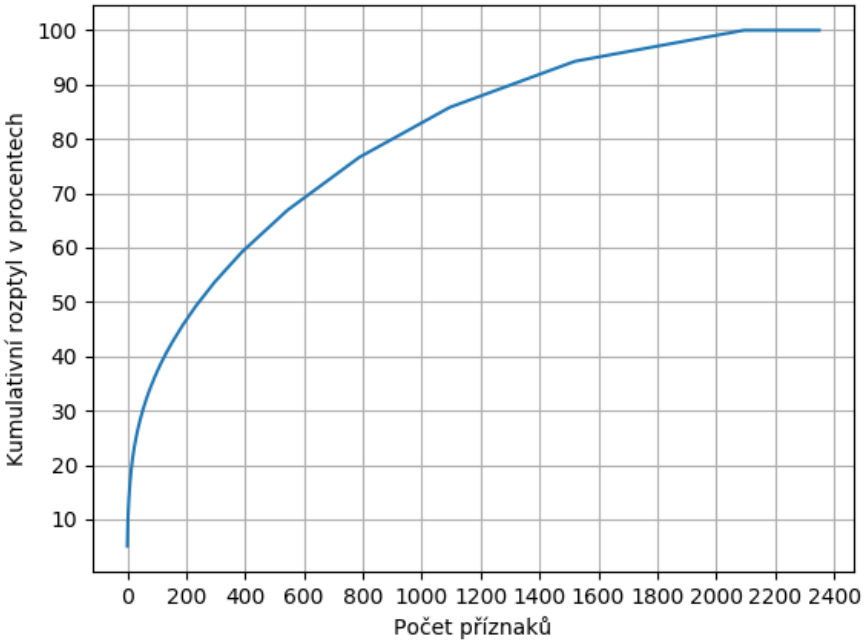
Pro vybraní příznaků použitých při vytváření *modelu 1* byla použita selekce příznaků. Konkrétně byly nejprve vyfiltrovány příznaky s nulovým rozptylem a následně byla použita třída SelectKBest, která odpovídá metodě Best individual N popsané v sekci 3.1. Kvůli paměťovým nárokům při učení transformátorů a chybějící metodě postupného učení bylo k učení nutno použít pouze polovinu, tedy 300 000 vzorků z trénovací datové sady. Pro výpočet účelové funkce byla ponechána přednastavená funkce f_classif, která zkoumá kvalitu příznaku pomocí Analýzy rozptylu.

Příznaky druhého modelu byly vytvořeny za pomocí Analýzy hlavních komponent, která byla představena v kapitole 4. Před provedením analýzy byly data normalizovány za pomoci metod ze třídy StandardScaler, jejíž funkce a význam byl blíže představen v sekci 4.1. Jak třída pro škálování dat, tak pro analýzu hlavních komponent obsahují metodu pro postupné učení, a proto bylo možné použit všechna trénovací data.

Výsledek analýzy hlavních komponent lze vidět na obrázku 6.2 a hodnoty kumulativního rozptylu pro vybrané počty výsledných příznaků jsou zachyceny v tabulce 6.2.

Tabulka 6.2: Poměr zachované informace o původním rozptylu dat pro vybrané počty příznaků.

Příznaků	Kumulativní rozptyl [%]
25	23,37
50	29,41
100	36,93
200	46,59
400	59,67
600	69,08
800	76,94
1000	82,94
1200	87,87
1400	91,87
1800	97,05
2200	99,98



Obrázek 6.2: Scree plot pro analýzu hlavních komponent.

6.3 Implementace klasifikátorů

Po transformování datových sad byly vybrané modely nastaveny následovně.

- Neuronové sítě – Při vytváření těchto modelů byla použita třída MLP-Classifier, která implemetuje vícevrstvou perceptronovou síť. Struktura sítě je ovlivněna parametrem hidden_layer_sizes. Pro tento experiment bylo použito celkem 300 skrytých uzlů. Ty byly rozděleny do tří vrstev a každá obsahovala sto uzlů. Tato struktura sítě byla zvolena s ohledem na studium materiálů zabývajících se doporučeními ke konstrukci sítí. Zejména pak obecných pravidel, které popsal Heaton ve své knize Artificial Intelligence for Humans [18].

- K nejbližších sousedů – Modely byly implementovány za pomoci použití třídy KNeighborsClassifier. Oproti přednastaveným hodnotám byl změněn pouze parametr n_jobs, který ovlivňuje počet paralelních výpočtů při klasifikaci, na hodnotu 4. Modely při klasifikaci hledaly pět nejbližších sousedů a k určení vzdáleností mezi vzorky používaly Eukleidovskou metriku.

- Rozhodovací stromy – K vytvoření modelů, které používají rozhodovací stromy byla použita třída DecisionTreeClassifier. Oproti výchozím hodnotám byla změněna pouze hodnota parametru random_state na hodnotu 0, aby byl experiment co nejlépe znovu opakovatelný. Pokud je parametr random_state poskytnut, je jeho hodnota použita při učení modelu namísto náhodného počátečního stavu pro používaný pseudonáhodný generátor. Jako funkce, která měří kvalitu rozdělení, byla použita Gini impurity, jejíž předpis byl definován vzorcem (1.10).
- Naivní Bayesův klasifikátor – K implementování modelů byla zvolena třída GaussianNB. Tato třída předpokládá, že pravděpodobnosti hodnot příznaků mají Gaussovo rozdělení.

Kapitola 7

Výsledky experimentu

Pro přehlednost následuje stručné shrnutí experimentu.

Napodobován byl *model 1*, který byl implementován pomocí knihovny lightGBM. *Model 1* byl naučen na 600 000 vzorcích s 1 400 příznaky.

Napodobující modely, označované jako *model 2*, byly učeny na datech čítajících 300 000 vzorků, které původně nebyly označené a označení bylo provedeno pomocí *modelu 1*. Napodobující modely byly učeny opakovaně, pokaždé s jiným počtem příznaků. Testovací sada obsahovala 200 000 označených vzorků.

Pro implementování *modelu 2* bylo vybráno více algoritmů. Vyhodnocení úspěšnosti těchto algoritmů lze nalézt v odpovídajících sekcích této kapitoly.

Tabulka 7.1 uvádí výsledky *modelu 1*. Tyto hodnoty byly uvedeny, aby bylo možné výsledky napodobujících modelů lépe pochopit.

Tabulka 7.1: Výsledky napodobovaného modelu vůči správným označením.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
1400	98,78	98,28	98,53	99,90

7.1 Vícevrstvé perceptronové sítě

Jak je patrné z tabulek 7.2 a 7.3, vícevrstvé perceptronové sítě dosahovaly velmi dobrých výsledků přes všechny počty příznaků. Struktura sítí nebyla vylepšována experimentováním s hodnotami parametrů. Dobré výsledky tak byly u této metody o to příznivější, protože existoval silný předpoklad, že vytvořená síť není pro daný problém optimální.

Tabulka 7.2: Výsledky vícevrstvých perceptronových sítí vůči označením prvního modelu.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	97,20	97,71	97,46	99,36
50	97,73	98,52	98,13	99,67
100	97,88	98,77	98,33	99,69
200	98,02	98,90	98,46	99,76
400	98,18	98,96	98,57	99,80
600	98,51	98,84	98,68	99,78
800	98,20	99,06	98,63	99,77
1000	97,88	**99,10**	98,49	99,80
1200	98,49	99,03	**98,76**	**99,83**
1400	**98,63**	98,87	98,75	99,74
1800	98,31	99,01	98,66	99,82
2200	98,54	98,85	98,70	99,82

Tabulka 7.3: Výsledky vícevrstvých perceptronových sítí vůči správným označením.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	97,02	97,03	97,03	99,21
50	97,44	97,72	97,58	99,52
100	97,59	97,97	97,77	99,51
200	97,72	98,08	97,90	99,55
400	97,82	98,08	97,95	99,64
600	98,18	98,00	98,09	99,61
800	97,90	**98,25**	98,07	99,62
1000	97,54	98,24	97,88	99,64
1200	98,09	98,13	98,11	**99,65**
1400	**98,27**	98,00	**98,14**	99,54
1800	97,96	98,15	98,06	**99,65**
2200	98,16	97,96	98,06	99,63

7.2 K-nejbližších sousedů

Kvalitních výsledků dosahovala také metoda K-nejbližších sousedů. Vyhodnocení úspěšnosti těchto modelů lze najít v tabulkách 7.4 a 7.5.

Bohužel časová výpočetní náročnost klasifikování vzorků znemožnila provést experiment pro větší počet příznaků.

Tabulka 7.4: Výsledky modelů používajících algoritmus K-nejbližších sousedů vůči označením prvního modelu.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	95,92	98,26	97,05	99,04
50	96,52	98,42	97,45	99,13
100	96,71	**98,49**	97,58	99,16
200	96,86	98,47	**97,65**	99,23
400	**96,89**	98,42	97,64	**99,24**
600	96,79	98,47	97,62	**99,24**

Tabulka 7.5: Výsledky modelů používajících algoritmus K-nejbližších sousedů vůči správným označením.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	95,86	97,70	96,74	98,70
50	96,53	97,93	97,20	98,84
100	96,72	**97,99**	97,33	98,87
200	96,78	97,88	97,31	98,86
400	**96,84**	97,86	97,34	98,91
600	96,81	97,98	**97,38**	**98,92**

7.3 Rozhodovací stromy

Rozhodovací stromy dosahovaly v porovnání s předchozími algoritmy horších výsledků. Se stoupajícím počtem příznaků docházelo k mírnému snižování úspěšnosti. Tento jev byl nejspíš zapříčiněn tím, že docházelo k přeučení modelu a klasifikátor tak ztrácel schopnost správně reagovat na doposud neviděná data.

Výsledky tohoto klasifikátoru jsou vedeny v tabulkách 7.6 a 7.7.

Tabulka 7.6: Výsledky rozhodovacích stromů vůči označením prvního modelu

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	94,36	96,20	95,25	95,25
50	**94,44**	**96,43**	**95,40**	**95,40**
100	94,41	96,01	95,19	95,19
200	94,16	96,02	95,05	95,06
400	93,93	95,75	94,81	94,81
600	93,98	95,79	94,85	94,86
800	93,75	95,08	94,40	94,40
1000	93,89	95,62	94,72	94,73
1200	93,75	95,63	94,66	94,66
1400	93,57	95,70	94,59	94,60
1800	93,43	94,89	94,14	93,66
2200	93,22	95,18	94,16	94,17

Tabulka 7.7: Výsledky rozhodovacích stromů vůči správným označením.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	94,06	95,40	94,69	94,69
50	**94,20**	**95,69**	**94,90**	**94,90**
100	94,11	95,21	94,63	94,63
200	93,93	95,29	94,56	94,56
400	93,64	94,95	94,25	94,25
600	93,71	95,03	94,33	94,33
800	93,55	94,39	93,94	93,94
1000	93,62	94,86	94,20	94,20
1200	93,55	94,94	94,19	94,19
1400	93,28	94,92	94,04	94,04
1800	93,43	94,89	94,14	94,14
2200	93,05	94,51	93,73	93,73

7.4 Naivní Bayesův klasifikátor

Jak lze vyčíst z tabulek 7.8 a 7.9, Bayesův naivní klasifikátor nedosahoval v této úloze dobrých výsledků. Pro vyšší počty příznaků model označoval většinu vzorků jako malware a měl tak vysoké procento planých poplachů. Špatné výsledky byly s největší pravděpodobností zapříčiněny tím, že nebyl splněn předpoklad podmíněné nezávislosti příznaků.

Tabulka 7.8: Výsledky naivního bayesova klasifikátoru vůči označením prvního modelu

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	**82,79**	79,02	**81,40**	**86,59**
50	53,06	**93,77**	55,64	71,65
100	51,37	88,89	52,61	53,81
200	50,89	87,72	51,79	51,73
400	52,40	92,56	54,48	55,03
600	52,67	92,70	54,93	55,33
800	52,88	92,09	55,25	55,58
1000	53,02	91,60	55,45	55,80
1200	53,22	91,38	55,75	56,04
1400	53,27	89,60	55,73	53,05
1800	53,67	90,44	56,42	56,68
2200	53,86	89,82	56,66	56,90

Tabulka 7.9: Výsledky naivního bayesova klasifikátoru vůči správným označením.

Příznaků	Precision [%]	Recall [%]	Accuracy [%]	AUC [%]
25	**82,87**	78,69	**81,21**	**86,30**
50	53,29	**93,68**	55,78	71,23
100	51,54	88,74	52,66	53,56
200	51,09	87,62	51,87	51,63
400	52,61	92,45	54,58	54,94
600	52,89	92,60	55,05	55,28
800	53,13	92,04	55,42	55,55
1000	53,26	91,54	55,60	55,75
1200	53,43	91,27	55,86	55,97
1400	53,43	89,41	55,74	52,99
1800	53,81	90,20	56,39	56,47
2200	53,94	89,50	56,54	56,61

Kapitola 8

Pokus o vylepšení výsledků

Na základě výsledků uvedených v kapitole 7, byly při pokusech o dosažení lepších výsledků použity pouze vícevrstvé perceptronové sítě. Tento algoritmus byl zvolen, protože dosahoval nejlepších výsledků a používá velké množství parametrů, které lze ladit pro daný problém.

První pokus o dosažení lepších výsledků spočíval v hledání co nejlepších parametrů sítě. Druhý pokus pak zahrnoval vylepšování klasifikátoru učením na správně klasifikovaných datech.

V této práci je vylepšený model označovaný též jako *model 3*.

8.1 Experimenty se strukturou vícevrstvé perceptronové sítě.

Při hledání nejlepších parametrů vícevrstvé perceptronové sítě byla použita třída GridSearchCV. V této třídě je implementována metoda, která vytvoří všechny možné kombinace uživatelem zadaných hodnot parametrů. Následně tato metoda určí kvalitu klasifikátoru pro každou vytvořenou kombinaci. Kvalita každé kombinace parametrů je určena pomocí křížové validace (cross-validation). Více informací o křížové validaci lze najít například v dokumentaci knihovny scikit[5].

Při vytváření instance této třídy byl oproti výchozím hodnotám změněn pouze parametr n_jobs na hodnotu 4, který ovlivňuje počet paralelních výpočtů. Použití výchozích hodnot znamenalo, že při křížové validaci byla trénovací datová sada rozdělena na 3 podmnožiny.

Hledání nejlepší struktury sítě bylo provedeno za použití dat obsahujících 400 příznaků. Tento počet byl zvolen, protože dosahoval v prvním experimentu dobrých výsledků a učení modelu s tolika příznaky trvalo přijatelnou dobu, průměrně okolo dvanácti minut. V případě použití více příznaků trvalo učení modelu téměř hodinu. Přijatelná doba učení modelu byla důležitá,

[5] `https://scikit-learn.org/stable/modules/cross_validation.html`

protože učeno bylo tolik modelů, jako byl počet vzniklých kombinací. A navíc byl každý model učen třikrát, pokaždé za použití jiné kombinace trénovacích a testovacích dat.

Experimentováno bylo s hodnotami parametrů hidden_layer_sizes, activation, max_iter, solver, learning_rate a alpha. Parametr hidden_layer_sizes definuje strukturu skryté vrstvy, activation funkci použitou při výpočtu hodnoty neuronu, max_iter maximální počet iteraci při učení, solver algoritmus použitý při optimalizaci vah spojení neuronů, learning_rate strategii při aktualizaci vah spojení neuronů a alpha udává L2 penalizaci.

Kombinací parametrů bylo vyzkoušeno více než 80. Jejich konkrétní hodnoty a výsledky lze najít v příloze. Nejlepší kombinace parametrů dosáhla při křížové validaci přesnosti 97 %. Neuronová síť s parametry z prvního pokusu měla přesnost při křížové validaci 96,5 %.

Konkrétní parametry *modelu 3* byly následující: $activation = relu$, $alpha = 1e-05$, $hidden\_layer\_sizes = (200, 200, 200)$, $learning\_rate = constant$, $max\_iter = 600$.

Zlepšení struktury sítě potvrdil i test na testovací sadě dat. Jak lze vidět v tabulkách 8.1 a 8.2, ke zlepšení došlo při napodobování prvního modelu i při porovnávání vůči správným označením.

Tabulka 8.1: Přesnost klasifikace vůči označením prvního modelu.

	Původní model	Vylepšený model
Precision [%]	98,18	98,24
Recall [%]	98,96	99,11
Accuracy [%]	98,57	98,68
AUC [%]	99,80	99,78

Tabulka 8.2: Přesnost klasifikace vůči správným označením.

	Původní model	Vylepšený model
Precision [%]	97,82	97,92
Recall [%]	98,08	98,28
Accuracy [%]	97,95	98,10
AUC [%]	99,64	99,58

8.2 Dodatečné učení pomocí správně označených dat.

Dodatečné učení bylo provedeno na *modelu 3*, viz kapitola 8.1.

K získání dodatečných dat k učení byl vybrán stejný postup, který použil Ács ve své bakalářské práci Static detection of malicious PE files [19].

Legitimní programy byly shromážděny z čisté instalace Windows 10, která obsahovala navíc několik aplikací od ověřených vydavatelů. Malware byl získán ze stránky `https://virusshare.com`. Výsledná datová sada obsahovala 14 000 legitimních vzorků a 14 000 příkladů malware. Unikátnost každého vzorku byla zajištěna vypočtením hodnoty hash funkce SHA-256 a porovnáním s metadaty z datové sady EMBER

Příznaky byly extrahovány ze získaných dat za pomoci třídy PEFeature-Extractor, která je součástí programu EMBER. Tato třída obsahuje metodu feature_vector, která z binárních dat aplikace získává numerické příznaky.

Po vytvoření vlastní datové sady bylo nutno data ještě škálovat a provést transformaci na 400 příznaků za pomoci analýzy hlavních komponent. Pro oba tyto kroky byly použity stejné transformátory jako při transformování dat pro *model 2*.

Učení proběhlo na 328 000 vzorcích, z čehož 28 000 odpovídalo nově přidaným vzorkům a 300 000 vzorků bylo stejných jako v původním experimentu. Pro dodatečné učení byly použity dva přístupy.

První spočíval v učení na datové sadě vzniklé spojením původních a nových dat. Tento postup je dále v textu označován jako *Dodatečné učení 1*.

Druhý postup je nazýván *Dodatečné učení 2*. Při *Dodatečném učení 2* byl nejprve načten naučený *model 3* a následně byl model doučen na nových datech. Použití předchozích výsledků z původního učení *modelu 3* bylo vynuceno nastavením proměnné warm_start na hodnotu True.

Jak lze vidět z tabulek 8.3 a 8.4, ani jeden z přistupů k vylepšení výsledků nevedl. Druhý postup dosáhl úplně stejných výsledků jako původní *model 3*. První postup dosáhl mírně horších výsledků při napodobování *modelu 1*, ale došlo k mírnému zlepšení vůči správným označením.

Tabulka 8.3: Přesnost klasifikace vůči označením prvního modelu.

	Model 3	Dodatečné učení 1	Dodatečné učení 2
Precision [%]	98,24	98,12	98,24
Recall [%]	99,11	99,06	99,11
Accuracy [%]	98,68	98,59	98,68
AUC [%]	99,78	99,80	99,78

Tabulka 8.4: Přesnost klasifikace vůči správným označením.

	Model 3	Dodatečné učení 1	Dodatečné učení 2
Precision [%]	97,92	97,90	97,92
Recall [%]	98,28	98,41	98,28
Accuracy [%]	98,10	98,15	98,10
AUC [%]	99,58	99,65	99,58

Stejné výsledky u druhého postupu byly nejspíše zapříčiněny tím, že dodatečné učení nijak neovlivnilo váhy spojení. Z jakého důvodu tomu tak bylo lze z provedeného pokusu jen odhadovat. Jako nejpravděpodobnější se jeví varianty, že model zůstal uvízlý v předchozích minimu z důvodu špatné konfigurace modelu, nebo z nedostatku nových informací v dodatečné datové sadě.

O příčinách mírného zhoršení po prvním postupu lze na základě dostupných informací také jen spekulovat. Pravděpodobnou příčinou zhoršení mohlo být to, že učení vícevrstvých neuronových sítí probíhá randomizovaně a výsledný model tak může i na stejných datech pokaždé skončit v jiném lokálním minimu chybové funkce.

Pokus, který by více experimentoval s poměry správně označených dat vůči datům označených pomocí *modelu 2* je součástí návrhů na možné pokračování výzkumu v závěru této práce.

Závěr

Hlavním cílem práce bylo provedení experimentu, který by odpověděl na otázku, zda a pomocí jakých kroků jsme schopni napodobit neznámý model detekující škodlivý kód. Dílčí kroky pro provedení tohoto pokusu zahrnovaly nastudování problematiky automatické detekce malware, provedení výběru příznaků na vybrané datové sadě, implementování zvolených algoritmů strojového učení a pokusu o vylepšení výsledků za použití dodatečných správně označených dat.

Všechny zadané cíle práce se podařilo splnit. Výsledky experimentu naznačily, že simulování modelu, který detekuje škodlivý kód, je možné. Z vybraných algoritmů dosáhly nejlepších výsledků neuronové sítě. Výsledný model dosáhl přesnosti 98,68 %.

Úvodní část textu diplomové práce obsahuje shrnutí problematiky strojového učení, a to především se zaměřením na problém binární klasifikace. Při představování algoritmů byl největší prostor věnován neuronovým sítím, u nichž na základě matematické teorie, která byla představena v sekci 2.5, existoval silný předpoklad, že budou pro tuto úlohu nejvhodnější.

V textu práce následuje kapitola 3, která popisuje algoritmy používané k výběru příznaků. Blíže byla představena analýza hlavních komponent. Tato metoda byla použita v praktické části práce k výběru příznaků používaných k učení napodobujících modelů.

Experiment byl proveden na datech z datové sady EMBER. Napodobován byl model, který používali autoři vybrané datové sady. Jedinou modifikací tohoto modelu bylo snížení počtu používaných příznaků. Snížení počtu příznaků bylo provedeno proto, aby příznaky používané u napodobovaného modelu byly co nejodlišnější od příznaků používaných napodobujícími modely. Napodobující modely byly vybrány na základě studia teorie automatické klasifikace malware.

Následný výzkum této problematiky by se mohl zaměřit na použití dalších algoritmů strojového učení. Existuje množství algoritmů, které v práci z časových důvodů použity nebyly.

Další možnou oblastí zájmu by mohlo být experimentování s velikostmi datových sad použitých k učení napodobujícího modelu. Zejména zajímavým pokusem by mohlo být kombinování vzorků označených napodobovaným modelem a správně označených vzorků a sledovat, jaký poměr vede k vytvoření nejpřesnějšího klasifikátoru. Tento pokus by mohl být přínosný proto, že by mohl zjistit, zda se vyplatí při vytváření klasifikátoru přidávat vzorky označené za pomoci automatického detektoru škodlivého kódu.

Možným pokračováním této práce by také mohlo být napodobování jiných modelů. Například by šlo zvolit za napodobované modely některé z komerčních antivirových programů.

Literatura

[1] Common ML Problems — Introduction to Machine Learning Problem Framing. Nov 2018, [Online]. Dostupné z: https://developers.google.com/machine-learning/problem-framing/cases

[2] Ayodele, T. O.: In *New Advances in Machine Learning*, editace Y. Zhang, kapitola Types of Machine Learning Algorithms, Rijeka: IntechOpen, 2010, doi:10.5772/9385. Dostupné z: https://doi.org/10.5772/9385

[3] Narkhede, S.: Understanding AUC - ROC Curve. Jun 2018, [Online]. Dostupné z: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[4] Barat, M.; Prelipcean, D.-B.; Gavriluţ, D. T.: A Study on Common Malware Families Evolution in 2012. *J. Comput. Virol.*, ročník 9, č. 4, Listopad 2013: s. 171–178, ISSN 1772-9890, doi:10.1007/s11416-013-0192-5. Dostupné z: http://dx.doi.org/10.1007/s11416-013-0192-5

[5] Gavrilut, D.; Cimpoesu, M.; Anton, D.; aj.: Malware Detection Using Machine Learning. In *Proceedings of the International multiconference on computer science and information technology*, ročník 4, Dec 2009, ISBN 978-83-60810-22-4, ISSN 1896-7094, s. 735–741, doi:10.1109/ACT.2010.33.

[6] Jiřina, M.: *Umělá inteligence*, ročník 4, kapitola Vybrané partie z neuronových sítí. Academia, 2003, ISBN 80-200-1044-0.

[7] Nielsen, M. A.: *Neural Networks and Deep Learning*. Determination Press, 2015.

[8] Kolosnjaji, B.; Eraisha, G.; Webster, G.; aj.: Empowering convolutional networks for malware classification and analysis. In *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, s. 3838–3845.

[9] Zhou, H.: Malware Detection with Neural Network Using Combined Features. In *China Cyber Security Annual Conference*, Springer, 2018, s. 96–106.

[10] Karn, U.: An Intuitive Explanation of Convolutional Neural Networks. Aug 2016, [Online]. Dostupné z: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[11] Kůrková, V.: *Umělá inteligence*, ročník 4, kapitola Aproximace funkcí neuronovými sítěmi. Academia, 2003, ISBN 80-200-1044-0.

[12] Webb, A. R.: *Statistical Pattern Recognition*. John Wiley and Sons, 2011.

[13] Ungar, L.: PCA. [Online]. Dostupné z: https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=Lectures.PCA

[14] Creating a Scree Plot. [Online]. Dostupné z: http://www.improvedoutcomes.com/docs/WebSiteDocs/PCA/Creating_a_Scree_Plot.htm

[15] Anderson, H. S.; Roth, P.: EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*, Duben 2018, 1804.04637.

[16] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[17] Jones, E.; Oliphant, T.; Peterson, P.; aj.: SciPy: Open source scientific tools for Python. 2001–, [Online]. Dostupné z: http://www.scipy.org/

[18] Heaton, J.: *Artificial Intelligence for Humans*, ročník 3. Createspace Independent Publishing Platform, deep learning and neural networks vydání, 2015, ISBN 978-1505714340.

[19] Ács, J.: *Static detection of malicious PE files*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2018.

Příloha **A**

Seznam použitých zkratek

ACC – Accuracy

AUC – Area under curve

FN – False negative

FP – False positive

FPR – False positive rate

PE – Portable Executable

ReLU – Rectified Linear Unit

ROC – Receiver Operator Characteristic

TN – True negative

TNR – True negative rate

TP – True positive

TPR – True positive rate

UML – Unified Modeling Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `results` ..... adresář obsahující naměřené výsledky a výstupy skriptů
- `DP_Slechta_Libor_2019.pdf` ..... text této práce ve formátu pdf
- `src` ..... adresář obsahující zdrojové kódy implementace
  - `README.txt` ..... návod k provedení experimentu
- `text` ..... adresář obsahující zdrojovou formu práce ve formátu LaTeX