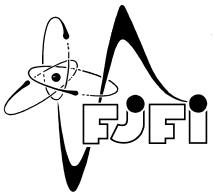
České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Identifikace operačních systémů na základě chování

Behavioral Identification of Operating Systems

Diplomová práce

Autor: **Bc. Benjamín Páterek**

Vedoucí práce: **Mgr. Jan Kohout, Ph.D.**

Akademický rok: 2020/2021

Katedra: matematiky Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Benjamín Páterek
Studijní program:	Aplikace přírodních věd
Studijní obor:	Matematické inženýrství
Název práce (česky):	Identifikace operačních systémů na základě chování
Název práce (anglicky):	Behavioral Identification of Operating Systems

Pokyny pro vypracování:

1) Nastudujte metody používané k identifikaci operačních jednotlivých zařízení v počítačové síti na základě záznamů jejich aktivity.
2) Navrhněte metodu (popř. metody) pro identifikaci operačních systémů, která bude založena na zpracování záznamů síťové komunikace jednotlivých zařízení. Při návrhu metody se zaměřte na schopnost metody pracovat i s velmi omezenými zdroji informací (např. v případě šifrované komunikace).
3) Vyhodnoťte výsledky navržené metody na datech z různých počítačových sítí a diskutujte její vlastnosti.

Doporučená literatura:

1) Q. Zhu et al. A Survey on Network Traffic Identification. In: International Conference on Artificial Intelligence and Security. Springer, 2019, pp. 91-100.

2) B. Anderson et al. OS fingerprinting: New techniques and a study of information gain and obfuscation. In: 2017 IEEE Conference on Communications and Network Security (CNS). IEEE, 2017, pp. 1-9

3) M. Husák et al. Network-based HTTPS client identification using SSL/TLS fingerprinting. In: 2015 10th International Conference on Availability, Reliability and Security. IEEE, 2015, pp. 389-396

Jméno a pracoviště vedoucího diplomové práce:

Mgr. Jan Kohout
Cisco Systems, Karlovo náměstí 10, Praha 2, 120 00

Jméno a pracoviště konzultanta:

Datum zadání diplomové práce: 31.10.2019

Datum odevzdání diplomové práce: 4.5.2020

Doba platnosti zadání je dva roky od data zadání.

Poděkování:

Chcel by som sa poďakovať svojmu školiteľovi Mgr. Janovi Kohoutovi, Ph.D. za vedenie pri tejto práci, ochotu a tiež za cenné rady a pripomienky. Rovnako by som chcel poďakovať spoločnosti Cisco Systems, s.r.o. za poskytnutie zázemia pre vznik tejto práce. Záverečná, avšak nemenej dôležitá vďaka patrí mojim rodičom, súrodencom a priateľom, ktorí mi boli pri písaní tejto práce výraznou oporou.

Čestné prohlášení:

Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 3. května 2021 Benjamín Páterek

Název práce:
Identifikace operačních systémů na základě chování

Autor: Bc. Benjamín Páterek

Obor: Matematické inženýrství

Druh práce: Diplomová práce

Vedoucí práce: Mgr. Jan Kohout, Ph.D., Cisco Systems, Karlovo náměstí 10, Praha 2

Abstrakt: Identifikácia operačného systému zariadenia zohráva v počítačovej bezpečnosti dôležitú úlohu. Súčasné trendy šifrovania sieťovej komunikácie však podstatným spôsobom obmedzujú zdroje informácií, ktoré môžu byť za týmto účelom použité. Táto práca navrhuje metódu, ktorá identifikuje operačný systém na základe analýzy šifrovanej komunikácie zariadenia s tzv. indikatívnymi doménami. Za týmto účelom metóda využíva techniky strojového učenia, ktoré sú použité pri samotnej identifikácii, ale zároveň aj pre vyhľadávanie indikatívnych domén. Experimentálna časť práce overuje navrhovanú metódu na dátach z reálnych počítačových sietí. Táto aplikácia viedla k získaniu cenných záverov, ktoré poukazujú napríklad na to, že automatizácia procesu vyhľadávania indikatívnych domén je výrazným prínosom v danej problematike.

Kľúčové slová: identifikácia operačného systému, operačný systém, šifrovaná komunikácia, indikatívne domény, analýza sieťovej komunikácie, strojové učenie

Title:
Behavioral Identification of Operating Systems

Author: Bc. Benjamín Páterek

Abstract: Operating system fingerprinting plays an important role in computer security. However, current trends in communication encryption significantly limit the sources of information that can be used for this purpose. This thesis proposes a method that identifies the operating system based on the analysis of encrypted communication of devices with so-called indicative domains. To achieve this, a novel method was developed, exploiting machine learning techniques for operating system identification and indicative domain search. The experimental part of

the work verifies the proposed method on data from real computer networks. This application has led to valuable conclusions, which indicate, for example, that the automation of the search for indicative domains is a significant contribution in this field of research.

Key words: operating system fingerprinting, operating system, encrypted communication, indicative domains, network communication analysis, machine learning

Obsah

Úvod

Identifikácia operačného systému (OS) zariadenia pripojeného na sieť je v oblasti počítačovej bezpečnosti všeobecne známou úlohou [1]. Zohráva totižto kľúčovú rolu v kybernetickej ochrane a efektívnom spravovaní počítačovej siete [2]. Pasívne metódy pre identifikáciu OS fungujú na princípe analyzovania prirodzenej prevádzky v sieti, pričom sa snažia rozoznať stopy správania, ktoré sú typické pre jednotlivé OS.

Mnohé existujúce metódy identifikujú zariadenie na základe porovnávania s databázou, ktorá obsahuje typické vzory správania pre jednotlivé OS. Udržovanie takýchto databáz však vyžaduje časovo náročnú expertnú prácu, čo nasmerovalo výskum ku snahe automatizovať procesy, napríklad využitím techník strojového učenia [3, 2]. Zásadným spôsobom na vývoj v oblasti identifikácie OS vplýva súčasný trend šifrovania komunikácie, ktorý výrazne obmedzuje použitie niektorých etablovaných metód (napr. metód založených na analýze User-agent reťazca) [1].

Táto práca vo svojej rešeršnej časti poskytuje prehľad existujúcich algoritmov pre identifikáciu OS, pričom následne sa venuje ich porovnaniu. Špecifická pozornosť je pritom venovaná použiteľnosti jednotlivých metód pre analýzu šifrovanej komunikácie. Teoretická časť práce popisuje návrh metódy, ktorá analyzuje šifrovanú komunikáciu zariadení s tzv. indikatívnymi doménami za účelom rozpoznania OS. Súčasťou teoretického návrhu je aplikovanie strojového učenia pre vyhľadávanie indikatívnych domén a takisto pre samotnú identifikáciu OS. V experimentálnej časti je metóda otestovaná na dátach z rôznych počítačových sietí s cieľom odhaliť hlavné výhody, nevýhody a overiť postupný vývoj metódy. Špeciálna pozornosť bola súčasne venovaná porovnaniu automatického a manuálneho vyhľadávania indikatívnych domén.

Práca je štrukturovaná nasledovne. Kapitola 1 má rešeršný charakter a je venovaná existujúcim algoritmom pre identifikáciu OS. Návrh vlastnej metódy a jej postupný vývoj je dokumentovaný v kapitole 2. Obsahom kapitoly 3 je vytvorenie experimentov, ktorých cieľom je overiť praktické použitie navrhovanej metódy. Dosiahnuté výsledky sú prezentované a diskutované v kapitole 4.

Kapitola 1

Identifikácia operačných systémov

1.1 Identifikácia OS vo všeobecnosti

Určenie operačného systému (OS) zariadenia, ktoré je pripojené na sieť, je v oblasti počítačovej bezpečnosti známa úloha [1, 4]. Vo všeobecnosti identifikácia prebieha získavaním informácií zo siete a následným aplikovaním algoritmu, ktorý tieto informácie vyhodnocuje. Každý operačný systém má svoje špecifické nastavenia, ktoré pri komunikácii na sieti zanechávajú stopu (anglicky fingerprint) [1]. Úlohou metód pre identifikáciu OS je tieto stopy pri komunikácii analyzovať a na ich základe rozhodovať o OS zariadenia.

Väčšina existujúcich techník je založená na manuálnom generovaní signatúr, ktoré sa následne porovnávajú s databázou, ktorá obsahuje signatúry typické pre jednotlivé OS. Manuálna aktualizácia veľkého množstva signatúr a správa databáz nových OS však vyžadujú značné množstvo času a úsilia. Z tohto dôvodu dochádza ku prípadom, kedy signatúry pre OS nie sú aktuálne [2]. Napríklad, ako sa uvádza v [1], posledné aktualizácie databáz pre nástroje *Ettercap* [5] a *p0f* [6] sú z roku 2011, respektíve z roku 2014. Táto problematika viedla vo vedeckej obci logicky ku snahe jednotlivé procesy automatizovať, napríklad využitím techník strojového učenia [2, 3]. V nasledujúcej časti kapitoly sa na jednotlivé techniky identifikácie OS pozrieme podrobnejšie.

1.2 Techniky identifikácie OS

Vo všeobecnosti môžeme rozdeliť techniky identifikácie OS podľa toho, či ku analýze sieťovej prevádzky pristupujú aktívne alebo pasívne.

1.2.1 Aktívny prístup

O aktívnom prístupe hovoríme v prípade techník, ktoré sú založené na aktívnom prenose jedného alebo viacerých špeciálne vytvorených sieťových paketov na zariadenie v sieti s cieľom analyzovať zodpovedajúce (a potenciálne identifikujúce) odpovede [2]. Tieto metódy určujú OS na základe prijatých odpovedí z cieľového zariadenia, skúmaním správania známeho TCP/IP zásobníka [7]. Tento prístup však kvôli generovaniu sieťových paketov vnáša do siete ďalší prenos, čo so sebou nesie isté nevýhody. Prvou nevýhodou je, že dochádza k umelému navýšeniu zaťaženia siete. Okrem toho, toto umelé zaťaženie môže samo o sebe vyvolávať alarmy alebo môže byť zablokované firewallom či prekladačmi sieťových adries (Network address translators) [8].

1.2.2 Pasívny prístup

Na druhej strane, pasívny prístup spočíva v analyzovaní paketov, ktoré v sieti putujú ako prirodzená komunikácia. Tieto techniky teda do siete nepridávajú žiadne špecificky vytvorené pakety. Namiesto toho analyzujú informácie, ktoré sa v sieti už nachádzajú a zameriavajú sa na vzory správania, ktoré sú špecifické pre rôzne operačné systémy [2]. Ako už bolo spomenuté vyššie, tieto vzory správania sa v mnohých prípadoch porovnávajú s databázami, ktoré obsahujú známe signatúry pre rôzne OS. V ďalších častiach práce si však ukážeme, že v súčastnosti existujú aj pokročilejšie techniky analýz než je porovnávanie s databázou. Môžeme si všimnúť, že pasívny prístup má oproti aktívnemu prístupu zrejmé výhody. Narozdiel od aktívneho prístupu neposiela do siete žiadne špecificky vytvorené pakety. Tým pádom nedochádza ku generovaniu ďalšej prevádzky v sieti a zároveň sa znižuje riziko spustenia alarmov či zablokovania firewallom [8]. Aj na základe týchto dôvodov sa v ďalšom priebehu práce zameriame predovšetkým na pasívny prístup.

Pasívne metódy sú teda založené na extrakcii istého druhu informácie zo sieťovej komunikácie. Nasledujúce podkapitoly budú venované zhrnutiu existujúcich algoritmov, pričom bude využité práve delenie na základe zdroja informácií, ktorý je využívaný pri identifikácii OS zariadenia.

1.2.3 TCP/IP parametre

Zariadenia, ktoré pre komunikáciu cez sieť využívajú TCP (Transmission Control Protocol), vytvárajú pakety. Pre komunikáciu cez sieť môže byť používaný napríklad aj UDP protokol,

avšak vzhľadom na bežnosť používania je najviac skúmanou alternatívou práve TCP protokol. V moderných počítačových systémoch je správa paketov TCP/IP zabezpečená jadrom OS [4]. Tieto pakety pozostávajú z dvoch častí: hlavičky a tela. Telo obsahuje údaje, ktoré chce odosielateľ preniesť na príjemcu. V hlavičke sú obsiahnuté údaje o zdroji, cieľovej destinácii a ďalšie užitočné údaje, ktorých cieľom je zabezpečiť správne vlastnosti spojenia ako sú napríklad spoľahlivosť, správne zoradenie paketov, kontrola preťaženia a pod.

Protokol presne definuje hodnoty väčšiny polí v rámci hlavičky, aj keď niektoré z nich majú určitú variabilitu v závislosti od potrieb odosielateľa. Implementácie TCP/IP zásobníka sa medzi rôznymi OS zvyčajne líšia a v dôsledku toho sa líšia takisto počiatočné predvolené hodnoty polí v hlavičke. Z tohto dôvodu môžu spomínané hodnoty slúžiť na identifikáciu OS zariadenia. [4]

Ako uvádzajú autori v [1], väčšina informácií, ktoré slúžia na identifikáciu OS zariadení pochádza z hlavičiek TCP/IP paketov a aplikačných protokolov. Zároveň dodávajú, že prístup s využitím hlavičiek TCP/IP paketov je dobre preskúmaný. Napríklad v práci Lipmanna a kol. [9] je prezentovaný klasifikátor, ktorý je schopný identifikovať 9 tried OS na základe hlavičiek paketov. Ďalšou možnosťou pri detekcii OS je využiť paket inicializujúci TCP spojenie, tzv. SYN paket. Ako príklad je možné uviesť prácu Tyagiho a kol. [10], v ktorej autori využívajú vlastnosti SYN paketu ako sú napríklad dĺžka paketu, TTL (Time to Live), veľkosť TCP okna a pod.

V [1] je autormi uvedené, že pri metódach založených na analýze TCP/IP parametrov je známou výzvou rozhodnúť sa, ktoré údaje z hlavičiek budú použité. Tento proces nazývame výber príznakov (anglicky feature selection). Naprieč rôznymi metódami sa rozsah príznakov mení od používania jedinej hodnoty TTL až po analýzu takmer každého bitu v hlavičke [1]. V práci je ďalej uvedené, že predovšetkým modely strojového učenia pre výber príznakov majú tendenciu vyberať parametre, ktoré závisia skôr od oboch komunikujúcich strán alebo od aplikácie, ako od samotného OS. Ich cieľom bolo naopak využiť údaje, ktoré závisia výhradne na jadre OS a môžu byť použité priamo na jeho identifikáciu. Na základe týchto kritérií vybrali nasledujúce príznaky:

- Veľkosť inicializujúceho SYN paketu (synSize) - veľkosť prvého paketu TCP spojenia je daná špecifickou implementáciou OS.
- Veľkosť TCP okna (winSize) - začiatočná hodnota veľkosti okna určuje počet oktetov, ktoré je prijímajúca strana pripravená prijať. Táto hodnota vychádza z nastavenia OS.

- TTL - táto hodnota, ktorá je nastavená jadrom OS, sa znižuje o jedna na každom elemente siete, aby sa zabránilo nekonečnému putovaniu paketov.

Tieto príznaky boli v rámci práce [1] použité nasledovne. Autori na základe dvojmesačného monitorovania univerzitnej siete našli unikátne trojice (synSize, winSize, TTL). Následne vytvorili databázu zobrazení zo spomínaných trojíc na unikátne OS, resp. ich verzie. Zároveň každému priradili váhu nazývanú spoľahlivosť, ktorá bola spočítaná ako podiel počtu spojení pre daný OS ku celkovému počtu spojení pre zodpovedajúcu unikátnu trojicu. Názornú ukážku z tejto databázy je možné vidieť na 1.1.

synSize	winSize	TTL	OS	Spoľahlivosť
52	8192	128	Windows 10.0	55.2%
52	8192	128	Windows 6.1	31.9%
52	65535	128	Windows 10.0	74.9%
60	65535	64	Android 6.0	48.2%
60	14600	64	Android 4.4	28.4%
60	29200	64	Ubuntu	20.4%
64	65535	64	Mac OS X 10.12	26.5%
64	65535	64	iOS 10.3	10.3%

Tabuľka 1.1: Tabuľka znázorňuje ukážku z databázy zobrazení, ktorá je využitá pri identifikácii OS na základe TCP/IP parametrov. Údaje v tabuľke sú prevzaté z [1].

Záverečným krokom popisovanej metódy je samozrejme samotná identifikácia OS na základe TCP/IP parametrov, ktoré sú zachytené v rámci toku sieťovej komunikácie. Keď algoritmus zachytí tok obsahujúci všetky tri požadované parametre, priradí mu OS, ktorý na základe databázy vykazuje najvyššiu spoľahlivosť. Ako hlavnú nevýhodu danej metódy uvádzajú autori práce [1] fakt, že identifikácia OS je ovplyvnená nerovnomernou distribúciou OS v rámci univerzitnej populácie. Menej používané OS sú tak zatienené tými populárnejšími.

Metóda založená na rovnakom zdroji údajov, ktorá však pri identifikácii OS využíva techniky strojového učenia je prezentovaná v práci [3]. Autori sa v tejto práci rozhodli namiesto vytvorenia databázy zobrazení pozerať na identifikáciu OS ako na klasifikačnú úlohu strojového učenia, pri ktorej ako príznaky využívali totožnú trojicu parametrov - synSize, winSize, TTL. Informácia o OS pre každú danú trojicu bola využitá ako trieda (label). Na dáta tohto tvaru boli následne použité štyri populárne prístupy pre klasifikáciu v strojovom učení - *Naive Bayes* [11], rozhodovací strom (*Decision Tree* [12]), k-NN (*k-Nearest Neighbors* [13]) a SVM (*Support Vector Machine* [14]). V zhodnotení práce autori uvádzajú, že z testovaných techník

sa najviac osvedčili rozhodovacie stromy. Naopak, výrazne slabšie výsledky boli zaznamenané pri využití *Naive Bayes* algoritmu.

1.2.4 Špecifické domény [1, 4]

Ďalšou možnosťou pre identifikáciu OS zariadenia je využitie špecifických domén. Tento princíp je založený na skutočnosti, že moderné OS sú konfigurované tak, aby po pripojení na sieť vykonávali špecifické akcie. Ako príklady týchto akcií sú v práci [1] uvedené testovanie pripojenia (zodpovedajúca doména napr. *connectivity-check-android.com*) alebo kontrola aktualizácií (napr. na doméne *update.microsoft.com*).

Takúto aktivitu je možné monitorovať ako komunikáciu s externými servermi alebo prípadne na úrovni DNS komunikácie, pričom prvá z možností je priamočiarejšia. Dochádza pri nej ku kontrole paketov odoslaných priamo na daný server. V prípade využitia HTTP protokolu je *hostname* (názov domény, na ktorú sa užívateľ pripája) obsiahnutý v jednom z polí hlavičky. V prípade, že je komunikácia šifrovaná pomocou TLS protokolu (*Transport Layer Security* [15]), server musí vedieť, pre ktorú doménu je správa určená bez toho, aby ju dešifroval. Táto záležitosť je vyriešená pridaním SNI (*Server Name Indication* [16]) vo forme textu. [4]

Práve analýzou webovej prevádzky realizovanej použitím HTTP/HTTPS protokolov so zameraním na princíp špecifických domén sa zaoberá v jednej zo svojich častí práca [1]. Ich cieľom je opäť vytvoriť databázu prepojených dvojíc - špecifická doména a zodpovedajúci OS. Pri vyhľadávaní špecifických domén boli použité dva prístupy. Prvý spočíval v ručnom vyhľadávaní. Autori sa zamerali na technické dokumentácie a príručky pre vývojárov od hlavných výrobcov OS, pričom sa venovali hlavne detailom ohľadne aktualizačných procesov a implementácii kontroly pripojenia. Neskôr svoju pozornosť obrátili na blogy a fóra, kde administrátori riešili problémy s konfiguráciou firewallu. Uvádzajú, že tento postup sa veľmi osvedčil, pretože blokovanie istých špecifických domén narušovalo funkčnosť OS a tým pádom odhaľovalo skutočne používané domény.

Druhý prístup odhaľovania špecifických domén spočíval v agregovaní prevádzky na podobnom princípe ako pri vyššie spomínanej metóde pomocou trojice TCP/IP parametrov. Autori zmieňujú, že tento postup nebol úplne priamočiary, pretože komunikácia bola zahltená aktivitou používateľov. Preto si to vyžadovalo manuálnu evaluáciu významu jednotlivých domén a posúdenie, či bola aktivita prepojená s OS alebo nie. Výrazným príspevkom spomínanej práce je určite zverejnenie databáz a ďalších poznatkov z výskumu v rámci ich Github repozitára [17].

Ako je možné usúdiť z uvedeného postupu, vytváranie databázy vyžaduje značné množstvo manuálnej práce. Hoci autori výskumu [1] veria, že zmeny názvov domén nebudú častým javom, manuálna expertná práca a jej časová náročnosť potrebná na udržanie aktuálnosti databázy sa javia ako obmedzenie danej metódy. Rovnako tak je potrebné spomenúť obmedzenie z hľadiska detailnosti identifikácie OS. Autori uvádzajú, že detekcia OS popisovanou metódou je možná na úrovni výrobcu OS alebo na úrovni názvov jednotlivých OS, avšak rozlišovanie medzi rôznymi verziami daných OS možné nie je.

Analyzovanie HTTP/HTTPS prevádzky je z hľadiska výskumu v rámci tejto práce zaujímavé a preto je mu venovaná výraznejšia pozornosť. Ako však už bolo spomenuté v úvode tejto podkapitoly, princíp špecifických domén môže byť využívaný aj pri analýze DNS prevádzky. Týmto smerom sa vyvíjal výskum prezentovaný napríklad v práci [18].

1.2.5 User-agent

HTTP User-agent [19] je jedným z polí hlavičky HTTP/1.1 protokolu [20], ktorý poskytuje serveru informácie o OS a prehliadači. Účelom takejto identifikácie je, že webový server môže poskytnúť používateľovi obsah vo forme prispôsobenej zariadeniu či softvéru. Ako príklad môžeme uviesť dnešný trend, v ktorom majú webové stránky špeciálnu verziu pre mobilné zariadenia. [1]

Vytváranie User-agent reťazca je plne pod kontrolou softvéru aplikácie nezávisle na OS zariadenia, pretože User-agent patrí do aplikačnej vrstvy sieťovej komunikácie. Špecifikácia HTTP/1.1 [20] dokonca vyslovene uvádza, že User-agent by nemal byť generovaný so zbytočnými podrobnosťami, aby sa tak zabránilo identifikácii používateľa. V praxi však môžeme bežne vidieť, že aplikácie vyplnia názov OS, vrátane jeho verzie. [1]

Pre lepšiu predstavivosť si môžeme na obrázku 1.1 všimnúť príklad User-agent reťazca s informáciami, ktoré môže bežne obsahovať.

```
User-Agent:
    Mozilla/5.0 (X11; Linux x86_64; rv:59.0)
    Gecko/20100101
    Firefox/59.0
```

Obr. 1.1: Obrázok znázorňuje príklad User-agent reťazca. Z typografických dôvodov sú medzery nahradené oddelením na nový riadok. Ukážka je prevzatá z [4].

Identifikácia OS na základe User-agent reťazca je pomerne známa metóda v oblasti počítačovej bezpečnosti a existuje pre ňu viacero nástrojov. Napríklad v práci [1] bolo využívané komerčné riešenie Flowmon, ktorého súčasťou je aj nástroj pre analýzu User-agent reťazcov. K dispozícii sú však aj rôzne ďalšie nástroje, medzi ktorými môžeme spomenúť napríklad ua-parser [21], ktorý bol používaný aj v rámci tejto práce.

V poslednej dobe však v počítačovej bezpečnosti rastie trend šifrovania sieťovej prevádzky a správy HTTP protokolu bývajú zabalené v TLS tuneli. Táto skutočnosť účinne zabraňuje využívaniu User-agent reťazca na identifikáciu OS, pretože obsah správy je pre vonkajšieho pozorovateľa komunikácie nečitateľný [4]. Šifrovanie sieťovej prevádzky a jeho vývoj v súčasnej dobe sa v kontexte identifikácie OS javí ako zásadné obmedzenie pri využívaní User-agent reťazca.

1.2.6 DHCP parametre [4]

Aby zariadenie mohlo byť súčasťou počítačovej siete, musí mať vytvorenú IP adresu. Tú je možné vytvoriť buď staticky (manuálnou konfiguráciou) alebo dynamicky pomocou istého mechanizmu pre získavanie adresy od niektorej lokálnej sieťovej autority. Pri využívaní IPv4 protokolu [22] je týmto mechanizmom DHCP (*Dynamic Host Configuration Protocol* [23]).

Siete, ktoré sú spravované DHCP protokolom, obsahujú server, ktorý je zodpovedný za priraďovanie IP adries. Po pripojení sa k sieti s dynamickým priradením IP adresy klient odosiela správu *DHCP DISCOVER*. Server reaguje zaslaním *DHCP OFFER*, ktorý okrem iných informácií obsahuje aj ponúkanú IP adresu. Klient následne môže požiadať o pridelenie adresy pomocou správy *DHCP REQUEST*, na ktorú server odpovedá buď *DHCP ACK* (prijatie žiadosti) alebo *DHCP NAK*, v prípade, že požadovaná IP adresa bola už pridelená alebo jej platnosť vypršala. [4]

Ak je adresa úspešne priradená, môže ju klient využívať určitú dobu, ktorá bola uvedená v správe *DHCP OFFER*. V momente, keď klient danú IP adresu už nepotrebuje, mal by ju vrátiť serveru, aby mohla byť pridelená inému klientovi. Táto akcia je vykonaná pomocou správy *DHCP RELEASE*. Mechanizmy na prácu s DHCP protokolom sú zvyčajne implementované na úrovni OS. To znamená, že DHCP správy obsahujú polia, ktorých hodnoty závisia od konkrétnej implementácie. Z tohto dôvodu je možné pozorovať rozdiely v kontexte správania sa rôznych OS pri spracovávaní DHCP správ a následne ich využiť pri identifikácii OS [24]. Výhoda tohto prístupu spočíva v tom, že užívateľ nemôže jednoducho upravovať charakteristické vlastnosti správ a vykonávanie protokolu. Zároveň, aby sa mohol k dynamicky spravovanej sieti pripojiť,

tak je tento postup povinný. Z tohto dôvodu sú údaje dostupné pre všetky zariadenie, ktoré boli monitorované počas pripojenia. [4]

Z hľadiska výskumu bol tento postup využívaný predovšetkým na získanie tzv. základnej pravdy (anglicky *ground truth*). Tento pojem v našom prípade vyjadruje znalosť skutočného OS daného zariadenia. Táto znalosť je následne použitá na vyhodnotenie, ako správne fungujú identifikácia OS v prípade použitia nejakého algoritmu. Aplikovanie DHCP záznamov týmto spôsobom evidujeme napríklad v prácach [2, 4]. Konkrétne je využívaná informácia o názve zariadenia, ktorú je možné čerpať z DHCP servera. Ako sa uvádza v [1], tento spôsob je mimoriadne efektívny pre zariadenia s operačným systémom od Apple a Google, pretože ich názvy sú prednastavené a je takmer nemožné ich zmeniť.

1.3 Porovnanie metód

Druhá časť tejto kapitoly bude zameraná na porovnanie prístupov k identifikácii OS, ktoré boli popísané vyššie. V rámci porovnávania výsledkov rôznych prác nepovažujeme za prínosné venovať sa konkrétnym hodnotám evaluačných metrík ako sú spoľahlivosť (precision) a úplnosť (recall), pretože práce môžu používať rôzne dáta z hľadiska náročnosti (typ siete, distribúcia rôznych OS, počet tried v klasifikácii a pod.). Naopak, za veľmi prínosné považujem vyhodnotenie dosiahnutých výsledkov v práci [1]. Toto vyhodnotenie porovnáva rôzne typy metód z hľadiska hodnôt evaluačných metrík na dátach z rovnakej siete, čo zaručuje objektívnosť porovnania. Okrem toho poskytuje pohľad na perspektívnosť metód v kontexte súčasných trendov šifrovania komunikácie. Práve táto problematika je pre našu prácu obzvlášť zaujímavá a preto jej bude venovaná náležitá pozornosť.

V nasledujúcej časti teda zhrnieme pozorovania, ktoré publikovali autori práce [1]. Tieto pozorovania zároveň obohatíme závermi z iných prác či subjektívnymi názormi, ktoré môžu byť predmetom skúmania v našej práci.

V spomínanej práci prebiehala identifikácia OS na úrovni jednotlivých Wi-Fi relácii (Wi-Fi sessions). Každá relácia je tvorená tokmi (flows). Tieto toky boli vyhodnocované pomocou techník popísaných vyššie. Konkrétne v rámci tejto publikácie boli použité algoritmy, ktoré sú popísané v predchádzajúcich častiach kapitoly a preto si ich už len pripomenieme v nasledujúcich bodoch.

1. **TCP/IP parametre**: identifikácia prebieha na základe vytvorenej databázy, ktorá je tvorená zobrazeniami z unikátnych trojíc (synSize, winSize, TTL) na OS, pričom každému

zobrazeniu je priradená hodnota spoľahlivosti.

2. **Špecifické domény**: identifikácia opäť prebieha na základe vytvorenej databázy, ktorá v tomto prípade pozostáva z prepojených dvojíc - špecifická doména a zodpovedajúci OS.

3. **User-agent**: reťazec je analyzovaný pomocou vstavanej funkcie komerčného nástroja Flowmon.

4. **Kombinácia metód**: táto metóda je kombináciou predchádzajúcich 3 metód. Každá z metód prebehne nezávisle a výsledný OS je daný princípom väčšinového hlasovania. V prípade rovnosti hlasov majú hlasy nasledovnú dôležitosť (od najväčšej po najmenšiu): User-agent, špecifické domény, TCP/IP parametre.

Porovnanie z hľadiska pokrytia prevádzky

Prvým kritériom pre porovnanie metód je to, akú časť zo všetkých relácii sú schopné vyhodnotiť, tj. priradiť relácii OS. Aby bola relácia vyhodnotená, musí obsahovať aspoň jeden tok nesúci informáciu, ktorú daná metóda potrebuje pre identifikáciu OS. V prípade User-agent metódy je táto požiadavka naplnená v 64.3% relácií. Čo sa týka špecifických domén, tam pokrytie predstavuje 78.1% prípadov. TCP/IP metóda vyžaduje pre identifikáciu OS iba TCP spojenie, z ktorého sa dajú zistiť zodpovedajúce parametre a túto požiadavku naplnilo až 88.4% relácií. Pri použití kombinácie metód bolo možné dosiahnuť pokrytie až 93.4%.

Porovnanie z hľadiska presnosti identifikácie OS

V tejto časti porovnávali autori práce ich metódy na základe evaluačných metrík pre klasifikačnú úlohu. Konkrétne boli využité presnosť (accuracy), spoľahlivosť (precision), úplnosť (recall) a F-skóre (F-score) [25]. Tieto metriky boli napočítané pre jednotlivé triedy (jednotlivé OS) a následne bol použitý mikro-priemer pre výslednú hodnotu. Zhrnutie dosiahnutých výsledkov je možné vidieť v tabuľke na 1.2.

Najlepšie výsledky z pohľadu evaluačných metrík dosiahla jednoznačne metóda založená na User-agent reťazci. Vypichnúť môžeme predovšetkým spoľahlivosť, ktorá presahuje 98%. Ako dôvod takto vysokej spoľahlivosti uvádzajú autori skutočnosť, že aplikácie generujúce HTTP požiadavky sú obvykle priame pri zverejnení OS. Naopak, výrazne nižšia hodnota úplnosti, približne 60% poukazuje na to, že podstatné množstvo User-agent reťazcov je v nepoužiteľnej forme (zašifrované, prípadne obsahujú informáciu len o aplikácii, ale nie o OS). [1]

Metóda	Presnosť	Spoľahlivosť	Úplnosť	F-skóre
User-agent	0.9189	0.9812	0.6063	0.7495
TCP/IP parametre	0.8088	0.5249	0.4643	0.4927
Špecifické domény	0.8402	0.6286	0.4907	0.5512
Kombinácia metód	0.8582	0.6587	0.6041	0.6302

Tabuľka 1.2: Tabuľka zhŕňa dosiahnuté výsledky metód v rámci práce [1].

Z celkového hľadiska nám teda metóda založená na User-agent reťazci pre značné množstvo tokov nevie poskytnúť žiadnu informáciu. Avšak v prípade, že informáciu poskytnúť vie, tak je táto veľmi dôveryhodná, na čo poukazuje vyše 98%-ná spoľahlivosť. Tieto charakteristiky umožňujú danú metódu využívať aj pre určovanie tzv. základnej pravdy (ground truth), kde je vysoká spoľahlivosť veľmi potrebná. Naopak to, že metóda má nižšiu úplnosť, nemusí pri tomto využití znamenať problém. V prípade, že napríklad DHCP parametre nie sú k dispozícii, je určovanie základne pravdy z User-agent reťazcov logickou alternatívou. Zmysluplnosť tohto kroku potvrdzuje napríklad práca [2], v ktorej bol využitý práve tento postup.

Využitie špecifických domén pri identifikácii OS sa na základe výsledkov v práci [1] javí ako veľmi sľubné. V porovnaní s metódou, ktorá je postavená na User-agent reťazcoch sú síce hodnoty evaluačných metrík značne nižšie, čo je očakávaný výsledok. Výrazne pozitívne však treba hodnotiť porovnanie s metódou založenou na TCP/IP parametroch, ktorá je v oblasti identifikácie OS etablovanou záležitosťou. Oproti nej je nová metóda (založená na špecifických doménach) lepšia naprieč všetkými evaluačnými metrikami, čo jednoznačne poukazuje na jej potenciál. Ako jej hlavné obmedzenie pri detekcii OS uvádzajú autori rozlišovanie medzi rôznymi OS od toho istého výrobcu, špeciálne to platí pre Apple produkty. Dôvod spočíva v skutočnosti, že všetky zariadenia od Applu komunikujú s podobnou množinou domén a aj ich aplikácie sťahujú aktualizácie z rovnakých domén. Zo subjektívneho hľadiska považujeme za limitáciu prezentovanej metódy spôsob identifikácie OS, ktorý spočíva na databáze prepojených dvojíc - špecifická doména a zodpovedajúci OS (popísané v prvej časti kapitoly). Táto limitácia však môže byť vnímaná pozitívne ako priestor pre odhalenie plného potenciálu princípu špecifických domén, čo bude predmetom výskumu v našej práci.

Metóda založená na TCP/IP parametroch dosiahla v porovnaní s User-agent metódou výrazne slabšie výsledky. Spoľahlivosť bola na úrovni 52.5%, úplnosť mala hodnotu 46.4%. Ako hlavné 2 dôvody pre takto slabé výsledky uvádzajú autori v [1] nasledovné skutočnosti. Za prvé, že Apple produkty zdieľajú rovnaké parametre, čo spôsobuje, že všetky Apple zariadenia

sú klasifikované ako MAC OS X, pretože metóda medzi nimi nevie rozlišovať. A za druhé, že Windows a Android zariadenia používajú pri komunikácie rôzne konfigurácie parametrov, čo v kombinácii s ich dominantným výskytom v sieti komplikuje klasifikáciu ostatných OS.

Ako sofistikovanejšiu alternatívu k popisovanej metóde môžeme vnímať prácu prezentovanú v [3]. Autori pracujú s rovnakými dátami využívajúc TCP/IP parametre, ale namiesto vytvorenia databázy používajú pre klasifikáciu rôzne techniky strojového učenia. Keďže pre agregáciu dosiahnutých hodnôt spoľahlivosti a úplnosti (pre jednotlivé triedy) bol v tomto prípade použitý makro-priemer, nie je ich možné objektívne porovnať s výsledkami práce [1], v ktorej bol použitý mikro-priemer. Výsledky tak môžeme porovnať jedine na základe hodnoty presnosti. Pri využití rozhodovacieho stromu bola dosiahnutá presnosť 97.6%, zatiaľ čo pri využití databázového prístupu z práce [1] to bolo 80.9%. To naznačuje, že využitie techník strojového učenia môže byť veľmi užitočné. Na druhú stranu je potrebné spomenúť, že hoci práce [3, 1] vychádzali z totožných dát, tak v rámci práce [3] boli dáta pred klasifikáciou vyčistené, čo takisto mohlo zohrať úlohu pri vylepšení presnosti klasifikácie.

Autori publikácie [2] v rešeršnom zhrnutí existujúcich TCP/IP metód tvrdia, že mnohé z nich sú založené len na základných TCP/IP parametroch. Tieto nástroje tak podľa nich nie sú schopné extrahovať všetky možné príznaky, ktoré sú špecifické pre rôzne OS. Na rozdiel od týchto prác navrhujú algoritmus, ktorý prepája využitie základných TCP/IP parametrov so základným TCP variantom, ktorý je pasívne detekovaný použitím strojového učenia. Vo výsledkoch práce deklarujú, že pri využití detekcie TCP variantu a jeho následnom použití pre identifikáciu OS sa presnosť algoritmu zvýšila z 85.6% na 91.3%. Tento výsledok poukazuje na fakt, že využitie techník strojového učenia pri konštrukcii príznakov môže vo výsledku viesť k výraznému zlepšeniu identifikácie OS.

Porovnanie z hľadiska šifrovania komunikácie

Ako už bolo spomínané v priebehu práce, súčasný trend v počítačových sieťach smeruje ku šifrovanej komunikácii. Táto situácia napomáha chrániť súkromie používateľov, avšak bezpečnostným analytikom komplikuje situáciu napr. pri identifikácii OS. Je preto na mieste skúmať, akým spôsobom budú jednotlivé metódy ovplyvnené v kontexte šifrovania komunikácie. Súčasné trendy naznačujú, že novými štandardami v sieťovej komunikácii budú IPv6 protokol [22], šifrovaná komunikácia cez TLS a využívanie HTTP/2.0 [26], resp. QUIC [27] protokolu. [1]

Pri uvedení spomínaných protokolov do bežného používania bude najviac zasiahnutou metóda založená na User-agent reťazci. Šifrovanie dátových prenosov pomocou TLS má za následok, že pole User-agent reťazca nebude počas dátového prenosu čitateľné. V prípade protokolu HTTP/2.0 nie je šifrovanie povinné, ale vývojári prehliadačov uviedli, že podporovaný bude iba šifrovaný formát [1]. QUIC protokol šifrovanie explicitne požaduje a len niektoré implementácie posielajú UAID (ekvivalent User-agent reťazca) v rámci prvej nešifrovanej správy [1].

Tento vývoj naznačuje, že využitie User-agent reťazca sa čoskoro stane takmer nepoužiteľným. Logickým vyústením týchto skutočností môže byť už spomínaná možnosť využiť User-agent reťazec na určenie základnej pravdy (ground truth), ktorá bude využitá pri vývoji iných metód, ktorých použitie nebude limitované šifrovanou komunikáciou.

V prípade TCP/IP parametrov uvádzajú autori práce [1], že šifrovanie danú metódu neobmedzí, avšak jej použitie sa bude musieť adaptovať aj na IPv6 protokol. Základné TCP/IP parametre, sledované pri použití IPv4 protokolu, majú svoje ekvivalenty aj pri použití IPv6 protokolu. Podobne to platí aj pre QUIC protokol, ktorý naväzuje spojenie za pomoci UDP protokolu. Parametre prvého paketu je možné merať, avšak vyžadovalo by to zmeny pri spôsobe monitorovania. Celkovo sa tak metódy založené na tomto princípe budú musieť etablovať na používanie iných protokolov. To môže znamenať napríklad rozšírenie databázy tak, aby pokrývala obe verzie IP protokolov, prípadne aby okrem TCP parametrov pokrývala aj UDP parametre. [1]

Špecifické domény narozdiel od predchádzajúcich dvoch metód nebudú nastupujúcimi trendmi v sieťovej komunikácii nijako ovplyvnené. Šifrovanie komunikácie metódu neobmedzí a rovnako tak ani používanie nových protokolov, pretože SNI pole je súčasťou všetkých z nich. Aj na základe týchto dôvodov autori článku [1] veria, že daná metóda sa v najbližšej dobe stane najviac spoľahlivou a presnou. Ako podmienku pre tento cieľ uvádzajú monitorovanie zmien vo špecifických doménach a udržovanie aktuálnosti ich databázy.

Kapitola 2

Navrhovaný prístup

V nasledujúcej kapitole sa budeme venovať prístupu k identifikácii OS zariadenia, ktorý bol navrhnutý v rámci tejto práce. Na začiatku bude poskytnutá motivácia k navrhovanej metóde, zatiaľ čo ďalšie podkapitoly budú popisovať 3 verzie navrhovaného algoritmu. Tieto verzie na seba naväzujú a postupne zaznamenávajú vývoj metódy v priebehu výskumu. Cieľom novej verzie algoritmu je jeho vylepšenie, prípadne reakcia na výzvy či obmedzenia, ktoré vyplývajú z predchádzajúcej verzie. Rovnako tak je zámerom postupne reagovať na komplexnejšie výzvy objavujúce sa v rešeršnej časti, ako je napr. automatizácia istých procesov. Takáto štruktúra nasledujúcej kapitoly takisto pripravuje pôdu pre experimentálnu časť práce, kde bude cieľom porovnať medzi sebou jednotlivé verzie metódy alebo v rámci možností vykonať porovnanie s existujúcim výskumom.

2.1 Motivácia

Identifikácia OS v počítačovej bezpečnosti

V oblasti počítačovej bezpečnosti zohráva znalosť OS zariadení, ktoré sa pohybujú v sieti, dôležitú úlohu z pohľadu bezpečnosti a ochrany súkromia [2]. Vedomosť o tom, aké zariadenie je pripojené do siete umožňuje bezpečnostným analytikom vykonávať správne rozhodnutia a adekvátne reagovať na bezpečnostné incidenty [28]. Medzi konkrétne možnosti využitia môžeme zaradiť identifikáciu kritických útokov a zraniteľností siete, odhaľovanie nových informácií o používateľovi či používanie nepodporovaného OS s bezpečnostnými rizikami [2, 1]. V našom prípade je znalosť OS zariadenia využívaná ako podporná informácia pre komplexnej-

šie bezpečnostné systémy. Z tohto dôvodu je potrebný výskum rôznych metód, ktoré umožnia detekciu OS na základe aktuálne dostupných dát zo sieťovej prevádzky.

Princíp navrhovaného prístupu

Na začiatku výskumu danej problematiky sme boli, podobne ako autori práce [1], inšpirovaní komunikáciou zariadení s internetovými doménami, ktorých navštívenie môže slúžiť ako indikátor pre identifikáciu OS zariadenia. Domény s touto vlastnosťou sme sa rozhodli nazývať *indikatívne domény*. Hoci tento princíp pôvodne vznikol nezávisle od existujúcich prác, pri rešeršnej činnosti sme narazili na článok autorov Lastovičku a kol. [1], ktorí ako prví skúmali podobný princíp v kontexte identifikácie OS. Z pohľadu nášho výskumu je spomínaná práca veľkým prínosom, pretože poskytuje závery, ktoré nášmu výskumu dodávajú ešte väčšie opodstatnenie a zmysluplnosť. Spolu s ostatnými závermi, ktoré vyplynuli z rešeršnej časti práce, si ich zhrnieme v nasledujúcich bodoch.

- Identifikácia OS na základe špecifických domén dosiahla na rovnakej vzorke zariadení presnejšie výsledky klasifikácie než metóda založená na etablovanom princípe využitia TCP/IP parametrov. [1]
- V kontexte šifrovania komunikácie a zavedenia nových protokolov sa metóda založená na špecifických doménach javí ako najperspektívnejšia možnosť. [1]
- Metóda založená na User-agent reťazci sa kvôli šifrovaniu komunikácie stane čoskoro neefektívna, avšak vysoká spoľahlivosť pri klasifikácii umožňuje využiť ju pre určovanie tzv. základnej pravdy. [1, 2]
- Techniky strojového učenia sa ukázali ako efektívny spôsob vylepšovania existujúcich algoritmov pri samotnej klasifikácii OS [3, 2], alebo aj pri vytváraní nových príznakov [2].

Logickým vyústením týchto bodov je metóda založená na princípe indikatívnych (špecifických) domén s využitím techník strojového učenia. Vzhľadom na charakter záznamov o prevádzke v sieti bude určovanie základnej pravdy prebiehať na základe User-agent reťazca. Daná metóda dostala z praktických dôvodov skratku *IDB-OSId* z anglického Indicative Domain Based Operating System Identification. V nasledujúcich častiach kapitoly sa budeme venovať jej teoretickému popisu, pričom sa zameriame na postupný vývoj metódy.

2.2 IDB-OSId 1.0

Popisovaná metóda pozostáva z nasledujúcich základných krokov, ktoré si detailne popíšeme v jednotlivých podkapitolách.

1. Vybudovať slovník indikatívnych domén.
2. Vytvoriť na základe slovníka dáta vo formáte klasifikačnej úlohy strojového učenia.
3. Vykonať klasifikačnú úlohu.

2.2.1 Slovník indikatívnych domén

Podobne ako v práci [1], aj naša úvodná úloha spočíva vo vyhľadávaní indikatívnych domén. Našim cieľom je vytvoriť zoznam domén, ktoré napomáhajú identifikácii OS. Narozdiel od práce [1], v ktorej bola vytvorená databáza prepojených dvojíc doména - OS, sa v našom prípade nebudeme obmedzovať na prípad, že daná doména je indikatívna len pre jeden OS. Ako jednoduchý príklad môžeme uviesť Apple domény, s ktorými komunikujú ako zariadenia s Mac OS X, tak aj zariadenia s operačným systémom iOS. Indikatívnosť domény môže spočívať takisto aj v tom, že ju navštevujú predovšetkým mobilné zariadenia. Z týchto dôvodov sa na indikatívnosť domény budeme pozerať všeobecnejšie, čo samozrejme zahŕňa prípad indikatívnosti pre konkrétny OS, ale takisto aj iné formy indikatívnosti.

Záznamy sieťovej prevádzky

Vyhľadávanie indikatívnych domén prebiehalo v našej práci na základe analýzy záznamov sieťovej prevádzky. Tieto záznamy obsahujú trojicu údajov: hostname reťazec, User-agent reťazec a zodpovedajúci počet tokov. Z týchto údajov sme mohli nasledovným spôsobom získať potrebné informácie.

- **Doména:** k jej získaniu bol využitý reťazec hostname. Doména bola braná ako časť reťazca, ktorá sa nachádza za prvou bodkou. V prípade, že reťazec obsahoval iba jednu bodku, tak bola doména braná ako celý reťazec.
- **OS:** priraďovanie jednotlivých záznamov k OS prebiehalo na základe analýzy User-agent reťazca. V tomto bode je prvýkrát využitý User-agent reťazec pre určenie tzv. základnej pravdy, čo sme avizovali v predchádzajúcich častiach práce. Pre tento krok bol použitý nástroj *ua-parser* [21].

Po aplikovaní popísaných dvoch krokov sme mali záznamy o sieťovej prevádzke v tvare (doména, OS, počet tokov). Na základe záznamov v tomto tvare bolo možné agregáciou získať nasledovné štatistiky:

- *tot_fl*... celkový počet tokov (flows) v záznamoch,
- *os_tot_fl(OS)*... celkový počet tokov pre daný OS,
- *fl(doména, OS)*... počet tokov na danej doméne pre daný OS.

Tieto štatistiky boli napočítané pre všetky OS a domény, ktoré sa nachádzali v záznamoch sieťovej prevádzky.

Dôležitosť OS

V ďalšom kroku bola zavedená dôležitosť OS. Tento krok bol zásadný vzhľadom na to, že jednotlivé OS nie sú medzi zariadeniami zastúpené rovnomerne. V praxi to môže znamenať, že aktivita menej častých OS je zatienená aktivitou častejšie používaných OS. Dôležitosť, resp. váha pre daný OS je definovaná ako:

$$w(OS) = \frac{tot\_fl}{os\_tot\_fl(OS)}. \tag{2.1}$$

Môžeme si všimnúť, že dôležitosť je definovaná na základe distribúcie OS v celých dátach. Pritom čím menšie zastúpenie daný OS v záznamoch má, tým je jeho výskyt dôležitejší. Hodnoty dôležitosti OS boli následne aplikované na počty tokov pre OS na jednotlivých doménach. Po aplikácii sme teda dostali vážený počet tokov na danej doméne pre daný OS:

$$w\_fl(dom, OS) = fl(dom, OS).w(OS), \tag{2.2}$$

kde *dom* je skrátené označenie pre doménu.

Ako už bolo spomenuté, naším cieľom v tomto kroku metódy je nájdenie indikatívnych domén. Túto úlohu je možné formulovať tak, že keď neznáme zariadenie navštívi indikatívnu domény, malo by nám to poskytnúť informačný zisk v kontexte OS zariadenia. Za týmto účelom bola pre každú doménu napočítaná entropia z hodnôt váženého počtu tokov pre jednotlivé OS. Myšlienka spočíva v tom, že čím menšia je entropia pre danú doménu (inak povedané, menšia miera neurčitosti), tým väčší je informačný zisk. Prečo počítať entropiu na základe váženého počtu tokov bude vysvetlené na nasledujúcom motivačnom príklade.

Predstavme si situáciu, že v dátach zo sieťovej prevádzky sa nachádzajú 3 operačné systémy - OS1, OS2 a OS3, pričom celkový počet tokov pre tieto OS je v pomere $10 : 1 : 7$. Z tohto údaju je možné napočítať váhy pre OS:

$$w(OS\,1) = 18/10,$$

$$w(OS\,2) = 18/1,$$

$$w(OS\,3) = 18/7.$$

Ďalej si predstavme, že pre doménu *xy.com* máme pre OS1, OS2 a OS3 počet tokov 100, 10 a 70. To znamená, že pre danú doménu máme rovnaké rozloženie OS ako je rozloženie v celých dátach, čiže v pomere $10 : 1 : 7$. Po aplikovaní dôležitostí dostávame nasledujúce hodnoty vážených tokov:

$$w\_fl(xy.com, OS\,1) = 100.(18/10) = 180,$$

$$w\_fl(xy.com, OS\,2) = 10.(18/1) = 180,$$

$$w\_fl(xy.com, OS\,3) = 70.(18/7) = 180.$$

Keďže všetky 3 hodnoty sú rovnaké, pri výpočte entropie dostaneme maximálnu hodnotu, čo odpovedá minimálnemu informačnému zisku. Tento výsledok je logický, pretože rozloženie OS na danej doméne je totožné s celkovým rozložením a v tomto zmysle nám doména poskytuje minimálnu informáciu. Naopak v prípade domén, kde by sa rozloženie, resp. distribúcia OS líšila od celkového rozloženia, by sme dostávali nižšiu hodnotu entropie a teda väčší informačný zisk. Rozdielnosť distribúcie OS na danej doméne od celkovej distribúcie OS môže byť braná práve na základe hodnoty entropie. Z tohto dôvodu môže byť entropia, ktorá je počítaná z vážených počtov tokov, braná ako miera vyjadrujúca informačný zisk z domény. Jednoduchšie povedané, táto miera vyjadruje, ako zaujímavá je doména z pohľadu OS zariadení, ktoré ju navštevujú.

Výber domén

Pri výbere domén do slovníka indikatívnych domén boli stanovené 2 kritériá, ktoré má doména zo slovníka spĺňať.

- **Indikatívnosť:** táto vlastnosť je meraná na základe entropie vážených počtov tokov pre jednotlivé OS na danej doméne. Motivácia k tomuto kroku bola predstavená vyššie.
- **Návštevnosť:** druhá vlastnosť, ktorú požadujeme od domén v slovníku je to, aby ju zariadenia navštevovali relatívne často. Táto požiadavka je logická, pretože ak chceme identifikovať OS na základe komunikácie s danou doménou, musí táto komunikácia prebiehať v dostatočnej miere. Meranie tejto vlastnosti je možné na základe počtu tokov, ktorý bol na danej doméne zaznamenaný.

Hodnoty týchto kritérií boli napočítané pre všetky domény. Pre výber domény do slovníka chceme, aby hodnota entropie bola čo najnižšia a zároveň, aby návštevnosť, teda počet tokov na doméne, bol čo najvyšší. V praxi boli tieto kritéria využité tak, že boli stanovené isté hranice pre obe kritériá a na základe toho boli vyfiltrované domény, ktoré boli potenciálne zaujímavé. Tie boli manuálne vyhodnotené na základe zaznamenaných štatistík tokov a takisto z hľadiska logického významu domény.

Hranice pre hodnoty návštevnosti a indikatívnosti bolo potrebné meniť či dopĺňať. Napríklad pri hľadaní domén, ktoré sú zaujímave z hľadiska Linux zariadení, bolo potrebné znížiť hranicu návštevnosti, pretože výskyt Linux zariadení bol výrazne nižší než napr. výskyt zariadení s OS Windows. Naopak, v tomto prípade sme mohli doplniť podmienku na istý minimálny počet tokov konkrétne pre OS Linux. Takýchto príkladov by bolo možné nájsť mnoho a z časového hľadiska išlo o náročnú úlohu, ktorá si vyžadovala množstvo manuálnej práce. Na túto skutočnosť pri podobnom postupe poukazovali aj v článku [1]. Tejto problematike budeme v ďalšom priebehu práce ešte venovať pozornosť, pretože v danej oblasti výskumu zohráva dôležitú úlohu. Každopádne, po vykonaní popisaného postupu sme vytvorili slovník 122 indikatívnych domén. Ten bol následne využitý v ďalšom kroku, ktorého cieľom bolo formulovať identifikáciu OS zariadenia ako klasifikačnú úlohu strojového učenia.

2.2.2 Identifikácia OS ako klasifikačná úloha

Strojové učenie a klasifikácia

V súčasnosti sa dá hovoriť o dvoch hlavných typoch strojového učenia. Sú nimi **učenie s učiteľom** a **učenie bez učiteľa**. V prípade učenia s učiteľom sú k dispozícii vstupné dáta vo forme príkladov (*samples*) s výstupmi (*labels*). Jednotlivé príklady sú bežne vyjadrené vo forme vektorov príznakov s konštantnou dĺžkou. Cieľom je naučiť sa na základe vstupných dát

priradiť výstup novému príkladu, ku ktorému výstup nie je známy. Pokiaľ je tento výstup z konečnej diskrétnej množiny, hovoríme o **klasifikácii**.

Vytvorenie dát pre klasifikáciu

Našou úlohou pri prepojení klasifikácie s identifikáciou OS je vyjadriť aktivitu zariadenia ako vektor príznakov s konštantnou dĺžkou, pričom zodpovedajúci výstup pre daný príklad (pre dané zariadenie) obsahuje informáciu o OS zariadenia. Hlavným cieľom pri vyjadrení aktivity je zachytiť mieru komunikácie s indikatívnymi doménami, ktoré sa nachádzajú v slovníku. Z tohto dôvodu jednotlivé domény zodpovedajú jednotlivým zložkám vo vektore príznakov, pričom hodnota príznaku odzrkadľuje mieru komunikácie daného zariadenia so zodpovedajúcou doménou. Konkrétny spôsob, akým bol získaný vektor príznakov vyjadrujúci aktivitu zariadenia je popísaný v nasledujúcich bodoch.

1. Komunikácia zariadenia je pozorovaná istý časový úsek, pričom tento úsek je rozdelený do 5-minútových intervalov.
2. Jednotlivé zložky vektora príznakov odpovedajú jednotlivým indikatívnym doménam v slovníku.
3. Hodnota i-teho príznaku je daná počtom 5-minútových intervalov, v ktorých priebehu dané zariadenie aspoň raz navštívilo i-tu doménu v slovníku.
4. Postup z predchádzajúceho bodu je aplikovaný na všetky príznaky.
5. Na záver sú všetky hodnoty príznakov znormované hodnotou, ktorá odpovedá počtu 5-minútových intervalov, počas ktorých bolo zariadenie aktívne, tj. bola pre neho zaznamenaná nejaká aktivita. Tento krok zaručí, že hodnoty vo vektore príznakov sú na škále od 0 do 1.

Posledným krokom pre vytvorenie dát na klasifikáciu je určenie výstupov pre jednotlivé príklady. To znamená, priradiť OS zariadeniu. Za týmto účelom bola opäť použitá analýza User-agent reťazcov pre dané zariadenie. Zdôraznime, že User-agent reťazce sú využívané len pre stanovenie základnej pravdy, teda pre určenie výstupov. Takisto dôležité je spomenúť, že v tejto verzii navrhovaného algoritmu boli uvažované 4 triedy: Windows, Apple, Android a Linux zariadenia. V prípade, že OS nespadal ani do jednej kategórie, tak mu bola priradená trieda Iné.

2.2.3 Klasifikačná úloha

V momente, keď máme k dispozícii dáta v správnej forme, nasleduje záverečný krok algoritmu IDB-OSId 1.0, ktorým je samotná klasifikácia. Za týmto účelom bol zvolený klasifikačný algoritmus k-NN (k-Nearest Neighbors [13]). Detaily ohľadne klasifikácie budú rozobraté v rámci experimentálnej časti práce.

2.3 IDB-OSId 2.0

Druhá verzia navrhovaného algoritmu pre identifikáciu OS zariadenia IDB-OSId veľmi priamo naväzuje na prvú verziu. Konkrétnejšie povedané, vychádza z rovnakého slovníka indikatívnych domén a rovnaký zostáva aj postup vytvorenia klasifikačných dát. Ako už bolo popísané, vektor príznakov v našom prípade reprezentuje sieťovú komunikáciu zariadenia. Predmetom záujmu v rámci druhej verzie nášho algoritmu je vytvorenie robustnejšej reprezentácie (napr. pomocou transformácie pôvodnej reprezentácie), ktorej použitie povedie k vylepšeniu klasifikácie v nasledujúcom kroku.

2.3.1 Inšpirácia

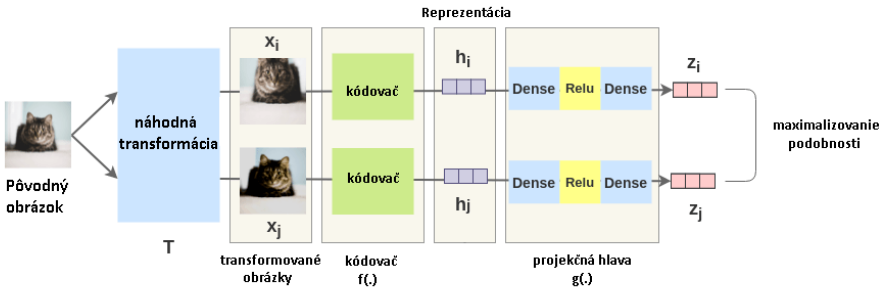
Inšpiráciou k tomuto prístupu bol článok [29], ktorý sa zaoberá učením reprezentácií obrázkov. V nasledujúcej časti budú zhrnuté základné myšlienky z tejto práce, pričom sa sústredíme predovšetkým na časti, ktoré sú použiteľné v našom výskume. Zhrnutie je inšpirované blogom [30], ktorý výborne objasňuje základné myšlienky z práce [29].

Kontrastné učenie

Nosnou myšlienkou popisovaného algoritmu Sim-CLR je princíp kontrastného učenia. V ňom ide o to, naučiť stroj rozoznávať medzi podobnými a odlišnými vecami. Aby sme tento problém formulovali vo forme, ktorej počítač rozumie, je potrebné nasledovné (uvažujeme oblasť klasifikácie obrázkov).

1. Príklady podobných a odlišných obrázkov: potrebné pre trénovanie modelu.
2. Kódovač: mechanizmus, ktorý reprezentuje obrázok vo forme, ktorej počítač rozumie.
3. Kvantifikovanie podobnosti: mechanizmus, ktorý odmeria podobnosť dvoch obrázkov.

Sim-CLR metóda



Obr. 2.1: Na diagrame je znázornený princíp fungovania Sim-CLR metódy [29]. Ukážka je prevzatá z [30] a upravená.

V Sim-CLR metóde prepojili autori tieto body nasledovným spôsobom. Na začiatku sa vezme obrázok a sú na neho paralelne aplikované dve nezávislé náhodné transformácie, čím vznikne pár obrázkov označených x_i a x_j. Oba obrázky následne prechádzajú kódovačom, ktorý obrázky vyjadrí pomocou reprezentácií h_i a h_j. Na tieto reprezentácie je aplikovaná nelineárna plne prepojená vrstva (*nonlinear dense layer*), čoho výsledkom sú reprezentácie z_i a z_j. Optimalizačnou úlohou je maximalizovať podobnosť medzi týmito reprezentáciami, ktoré vychádzajú z rovnakého obrázku. Celý tento proces je názorne zobrazený na diagrame 2.1.

Myšlienka teda spočíva v tom, naučiť kódovač, aby aj pre rôzne transformácie rovnakého obrázka vytváral podobné reprezentácie. Zároveň však chceme, aby odlišné obrázky mali aj odlišné reprezentácie. Učenie prebieha po šaržiach (*batches*), ktoré mali v článku [29] veľkosť $N = 8192$. Pre jednoduchosť si však jednotlivé kroky učenia prejdeme pre $N = 2$. To znamená, že máme 2 obrázky, napríklad obrázok mačky a obrázok slona.

Na každý obrázok je následne dvakrát aplikovaná náhodná transformácia (náhodná kombinácia posunutia, výrezu, otočenia, zmeny odtieňu a pod.), čím získame pár obrázkov. Keďže šarža obsahuje v tomto ilustratívnom príklade 2 obrázky, tak dostaneme 2 páry, čiže 4 obrázky. Pár transformovaných obrázkov, ktoré sú na 2.1 označené x_i a x_j, prechádza kódovačom $f(.)$, čím sú získané reprezentácie h_i a h_j. Pre kódovač v [29] je použitá architektúra ResNet-50 [31]. Reprezentácie h_i a h_j následne prechádzajú sériou nelineárnych $Dense \rightarrow Relu \rightarrow Dense$

vrstiev, čoho výsledkom sú reprezentácie z_i a z_j. Túto časť metódy nazývajú v práci [29] projekčnou hlavou a na 2.1 je označená ako $g(.)$.

Stratová funkcia

Učenie modelu spočíva v tom, že chceme maximalizovať podobnosť reprezentácií v rámci jednotlivých párov. V ilustračnom príklade uvažujeme dva páry, prvý z nich nech je x_1, x_2 a druhý x_3, x_4. Popísaným postupom sú získané zodpovedajúce reprezentácie z_1 a z_2, resp. z_3 a z_4. Podobnosť dvoch reprezentácií je vyjadrená pomocou kosínovej podobnosti, ktorá je definovaná ako:

$$s_{i,j} = \frac{z_i^T z_j}{\tau \left\|z_i\right\|_2 \left\|z_j\right\|_2}, \tag{2.3}$$

kde τ je nastaviteľný parameter teploty a $\|.\|_2$ označuje euklidovskú normu.

Stratová funkcia použitá v práci [29] sa nazýva **NT-Xent loss** z anglického *Normalized Temperature-Scaled Cross-Entropy Loss*. Intuícia za touto funkciou je nasledovná. Najprv sú napočítané vzájomné podobnosti pomocou definície 2.3. Páry, ktoré vznikli transformáciami z jedného obrázka (teda napr. pár x_1, x_2) nazývame pozitívne páry. Ostatné páry nevychádzajúce z rovnakého obrázka, ako napr. pár x_1 s x_3, nazývame negatívne páry. Pre vyjadrenie pravdepodobnosti, že 2 obrázky sú podobné, je použitá softmax funkcia. Napr. pre pár x_1, x_2 je toto vyjadrené ako:

$$p\_sim(x_1, x_2) = \frac{exp(s_{1,2})}{exp(s_{1,2}) + exp(s_{1,3}) + exp(s_{1,4})}. \tag{2.4}$$

Strata pre pár je vyjadrená ako záporná hodnota logaritmu z pravdepodobnosti podobnosti, čo je formálne možné vyjadriť nasledovne:

$$l(i, j) = -log \frac{exp(s_{i,j})}{\sum_{k=1}^{2N} l_{[k!=i]} exp(s_{i,k})}. \tag{2.5}$$

Vzhľadom na skutočnosť, že daný výraz nie je symetrický z pohľadu argumentov, strata je napočítaná pre daný pár aj s vymeneným poradím argumentov. Celková strata je daná ako priemer cez všetky páry v šarži.

$$L = \frac{1}{2N} \sum_{k=1}^{N} [l(2k-1, 2k) + l(2k, 2k-1)] \tag{2.6}$$

Optimalizácia tejto stratovej funkcie zaručuje vylepšovanie reprezentácií, ktoré sú produkované v priebehu metódy. Ako jeden zo záverov uvádzajú autori práce [29], že po ukončení kontrastného učenia môžu byť naučené reprezentácie úspešne použité napríklad pre klasifikačnú úlohu. Zároveň poukazujú na zaujímavú skutočnosť, že pri ďalšom použití v klasifikačnej úlohe sa viac osvedčilo využitie reprezentácie označenej ako h_i než reprezentácie z_i.

2.3.2 Aplikovanie kontrastného učenia

Z pohľadu nášho výskumu bolo zaujímavé práve využitie kontrastného učenia za účelom vytvorenia robustnejších reprezentácií, ktoré sa následne použijú pri klasifikácii. Aplikovanie myšlienok z článku [29] na náš výskum však nie je priamočiare. Najväčší rozdiel spočíva vo forme dát. V spomínanom článku pracujú autori s obrázkami, ktoré sú vyjadrené pomocou tenzoru s rozmermi napr. $(224, 224, 3)$. V našom prípade ide o zariadenia, ktorých aktivita je vyjadrená pomocou 1-dimenzionálneho vektora.

Zásadným krokom v SimCLR metóde je náhodná transformácia, kedy sú na obrázok aplikované dve náhodné nezávislé transformácie. Kontrastné učenie potom smeruje k tomu, aby reprezentácie týchto dvoch transformovaných obrázkov boli čo najpodobnejšie. Ide teda o učenie sa robustnosti voči transformáciám, ktoré sú typické pre obrázky - posunutie, výrez, zmena odtieňu a pod. Celkovo má teda výsledná reprezentácia vyjadrovať, čo sa na obrázku nachádza a to nezávisle na forme.

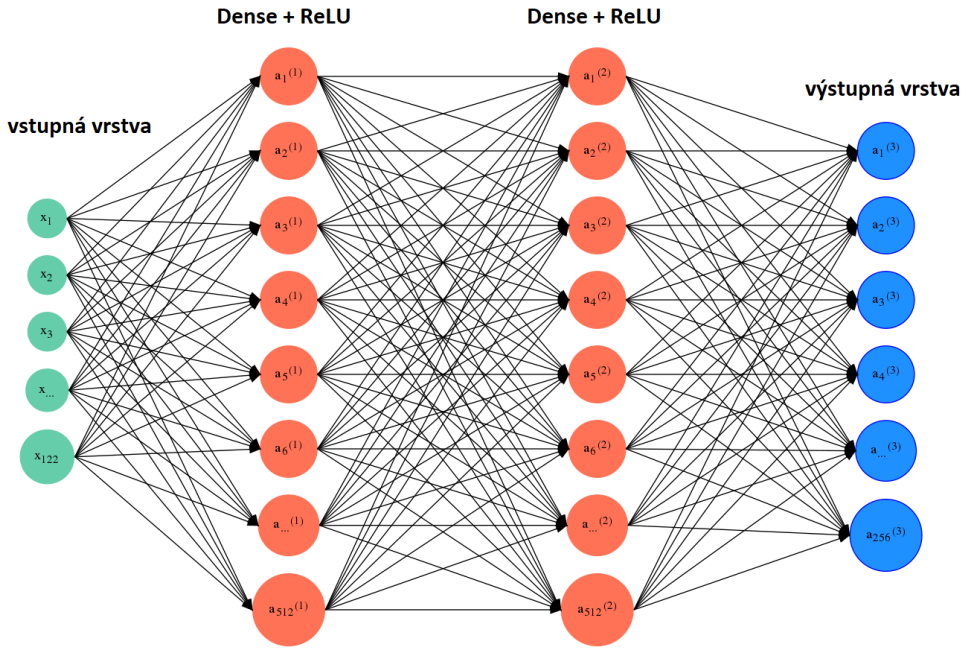
V našom prípade transformácie ako posunutie, otočenie a zmena odtieňu zrejme nedávajú žiadny zmysel. Preto je potrebné krok transformácie istým spôsobom modifikovať tak, aby kontrastné učenie bolo zmysluplné v kontexte našej úlohy. Rovnako tak je potrebné modifikovať aj architektúru kódovača. V rámci práce [29] je použitá architektúra ResNet-50. Táto architektúra je špeciálne navrhnutá pre reprezentovanie obrázkov. Niektoré typy vrstiev, resp. ich kombinácie sú zamerané na detekciu hrán, tvarov a pod. Okrem toho samozrejme očakáva vstup vo forme obrázka, čiže tenzor. Z týchto dôvodov je potrebné nájsť vhodnejšiu alternatívu pre našu úlohu.

Kontrastné učenie

Náš cieľ v kontexte kontrastného učenia môžeme formulovať tak, že chceme, aby reprezentácie rôznych zariadení s rovnakým OS boli podobnejšie. Konštrukcia pozitívnych párov teda môže v našom prípade znamenať náhodný výber dvoch zariadení s rovnakým OS. Optimalizácia stratovej funkcie tým pádom smeruje k zvyšovaniu podobnosti v pozitívnych pároch, teda

k zvyšovaniu podobnosti reprezentácií, ktoré odpovedajú zariadeniam s rovnakým OS. Takáto formulácia sa dá objektívne zhodnotiť ako zmysluplná. Zároveň sa týmto spôsobom nahradí krok transformácie, ktorý sa javil ako problematický. Stratová funkcia zostáva rovnaká ako v prípade práce [29], keďže intuícia popísaná v 2.3.1 je konzistentná aj v kontexte nášho cieľu.

Architektúra



Obr. 2.2: Architektúra použitá pri kontrastnom učení robustnejších reprezentácií v rámci nášho výskumu.

Ako už bolo popísané vyššie, architektúra ResNet-50 nie je v našom prípade vhodnou voľbou. Okrem už spomenutých dôvodov môžeme uviesť aj fakt, že ide o veľmi komplexnú architektúru. Vzhľadom na skutočnosť, že v našom prípade sú dáta výrazne menej komplexné (histogram popisujúci aktivitu 122 zložkami vs. obrázok vyjadrený ako tenzor s rozmermi $(224, 224, 3)$), považujeme za zmysluplné voliť takisto výrazne jednoduchšiu architektúru. Na základe tejto úvahy bola zvolená neurónová sieť zložená z *Dense* vrstiev s *ReLU* prenosovou funkciou, ktorej ilustráciu je možné vidieť na obrázku 2.2. Zjednodušenie v našej práci bude spočívať takisto aj v zjednotení kódovača a projekčnej hlavy do jednej komponenty. Berúc do úvahy zvolenú architektúru je tento krok prirodzený.

Postup učenia

Pri aplikovaní popisovaných myšlienok zohráva dôležitú úlohu logický postup. Je potrebné vziať do úvahy, že v našom prípade pri vytváraní pozitívnych párov využívame výstupy (labels). To znamená, že za týmto účelom môžeme využívať iba tréningovú časť dát pre klasifikáciu. Postup, ktorý uvažuje túto skutočnosť, zhrnieme pomocou nasledujúcich bodov.

1. Rozdelíme dáta pre klasifikačnú úlohu na tréningovú a testovaciu časť.
2. Tréningové dáta sú využité pri kontrastnom učení. Výsledkom je natrénovaná neurónová sieť, ktorú môžeme vnímať ako transformátor. Tento transformátor berie ako vstup pôvodnú reprezentáciu a jeho výstupom je nová, potenciálne robustnejšia reprezentácia.
3. Transformáciu aplikujeme na tréningovú aj testovaciu časť dát, čím dostávame dáta pre klasifikáciu, kde sú jednotlivé príklady vyjadrené pomocou naučenej reprezentácie.

Ako už bolo spomenuté, autori práce [29] uvádzajú, že v prípade použitia naučenej reprezentácie pre klasifikáciu, sa im osvedčilo použiť reprezentáciu h, ktorá je výstupom kódovača namiesto reprezentácie z, ktorá je výstupom z projekčnej hlavy. Inšpirovaný týmto zaujímavým záverom, aj v našom prípade otestujeme ako potenciálnu reprezentáciu nie len poslednú výstupnú vrstvu neurónovej siete, ale takisto aj predchádzajúce vrstvy.

Vzhľadom na implementačnú zložitosť pôvodného algoritmu sa ukázalo ako veľmi užitočné hľadanie minimálnej implementácie Sim-CLR metódy [32]. Adaptovanie tejto implementácie na náš prístup bolo efektívne, pričom boli zachované všetky potrebné aspekty metódy.

2.4 IDB-OSId 3.0

Nosnou myšlienkou tretej verzie navrhovaného algoritmu je automatizácia procesu tvorby slovníka indikatívnych domén. Ako už bolo spomenuté v priebehu práce (viď 2.2.1), proces výberu indikatívnych domén zahŕňa výrazne množstvo manuálnej práce. Tento krok je teda z časového hľadiska veľmi náročný, čo môže byť značne nevýhodné v kontexte udržovania aktuálnosti slovníka. Na podobný fenomén bolo možné naraziť aj v rámci práce [1], kde tvorba databázy špecifických domén rovnako vyžadovala rozsiahlu manuálnu expertnú prácu. Okrem toho autori tejto práce v diskusnej časti uvádzajú, že udržovanie aktuálnosti databázy je nevyhnutné. V prípade, že to bude zabezpečené, veria, že práve daná metóda sa v blízkej budúcnosti stane najspoľahlivejšou a najpresnejšou. Z týchto dôvodov môžeme usudzovať, že automatizácia tvorby slovníka môže byť kľúčovým faktorom pri vylepšení našej metódy.

2.4.1 Výber domén ako klasifikačná úloha

Pri automatizácii nejakého manuálneho procesu zohráva zásadnú úlohu formulovať daný problém spôsobom, ktorému počítač rozumie. Práve to nás motivovalo k tomu, formulovať výber domén do slovníka ako klasifikačnú úlohu strojového učenia. Keď vezmeme do úvahy proces, pri ktorom sme na základe istých štatistík pre doménu rozhodovali o tom, či ju zaradiť alebo nezaradiť do slovníka, tak podobnosť s klasifikačnou úlohou je zrejmá.

Pre realizáciu klasifikačnej úlohy je potrebné mať dáta vo formáte príkladov (samples) s odpovedajúcimi výstupmi (labels), pričom jednotlivé príklady sú vyjadrené pomocou vektorov príznakov (feature vectors). V našom prípade sú jednotlivými príkladmi domény a odpovedajúci výstup vyjadruje, či je doména zaradená do slovníka indikatívnych domén, alebo nie. V nasledujúcich odsekoch bude objasnené, ako dáta tohto formátu vznikli.

Vektory príznakov

Prvým bodom pri formulácii spomínanej klasifikačnej úlohy bolo vyjadrenie domén pomocou vektorov príznakov. Požiadavkou na tieto príznaky je to, aby zachytávali informácie relevantné z hľadiska posúdenia ich indikatívnosti. Samozrejmou podmienkou bolo aj to, aby sme hodnotu príznakov pre všetky domény boli schopní určiť na základe dostupných záznamov o sieťovej komunikácii.

Prvé 2 príznaky mali za úlohu vyjadriť, ako často bola doména navštevovaná zariadeniami. Túto vlastnosť môžeme merať na základe počtu tokov, ktorý bol na danej doméne zaznamenaný, čo bolo spomenuté už v 2.2.1. Druhý príznak, ktorý vyjadruje návštevnosť domény je počet unikátnych zariadení, ktoré doménu navštívili. To je užitočné najmä v prípadoch, kedy jedno zariadenie vo veľkej miere komunikuje s nejakou konkrétnou doménou, čím generuje na tejto doméne veľký počet tokov. Pokiaľ je však táto komunikácia typická len pre jedno zariadenie, prípadne malú skupinu zariadení, tak sa to ukáže práve na hodnote príznaku, ktorý zachytáva počet unikátnych zariadení. Hodnoty týchto príznakov sú napočítané pre všetky domény a následne sú obe hodnoty znormované maximálnymi hodnotami, ktoré boli zaznamenané naprieč všetkými doménami.

Druhá skupina príznakov popisovala rozloženie tokov na doméne z hľadiska rôznych OS. Za týmto účelom sme opäť využili vážený počet tokov na danej doméne pre daný OS, ktorý sme definovali v 2.2. Jeden príznak teda odpovedal jednému konkrétnemu OS. Vzhľadom na zastúpenie OS v záznamoch sieťovej prevádzky išlo o tieto OS (resp. ich skupiny): Windows, Android, Linux (všetky distribúcie okrem Ubuntu), Mac OS X, iOS, Ubuntu a iné. Vážený

počet tokov na danej doméne pre daný OS (teda hodnota príznaku) bol znormovaný súčtom vážených tokov na danej doméne, čím sme dosiahli relatívne vyjadrenie a hodnoty popisovaných príznakov boli na škále od 0 do 1.

Posledným príznakom bola entropia napočítaná z hodnôt váženého počtu tokov pre jednotlivé OS. Motivácia k tejto hodnote už bola takisto objasnená v 2.2.1. Dá sa povedať, že ide o istú mieru, ktorá vyjadruje indikatívnosť domény.

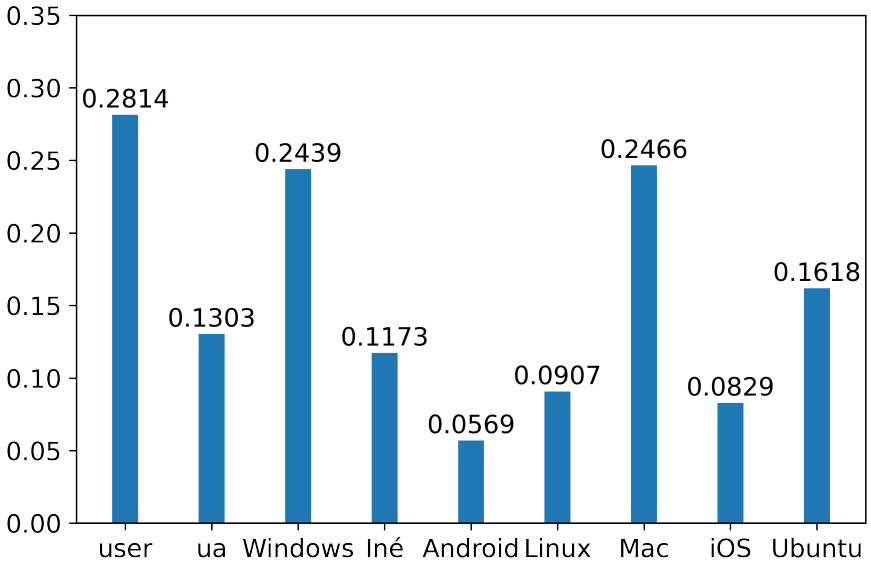
Popísané príznaky, resp. ich hodnoty sú napočítané pre všetky domény nachádzajúce sa v dátach. Tým pádom je možné danú doménu vyjadriť pomocou vektora príznakov. Ich celkový počet je 11, pričom prvé 2 majú za úlohu popísať návštevnosť domén, nasledujúcich 8 vyjadruje rozloženie tokov medzi OS na danej doméne, zatiaľ čo posledný príznak odzrkadľuje mieru zaujímavosti tohto rozloženia.

Získanie výstupov

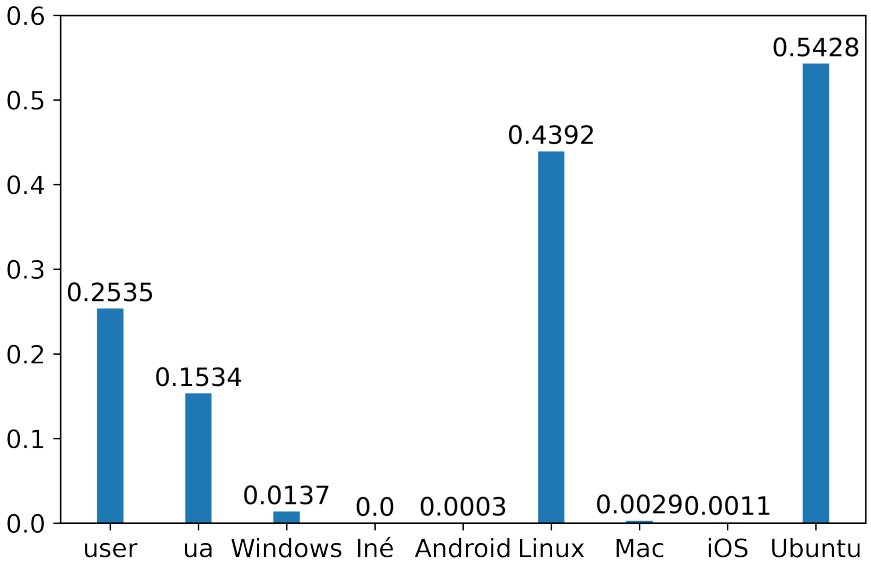
Zásadnú úlohu pri klasifikácii majú výstupy (labels), na základe ktorých sa klasifikačný algoritmus učí. V našom prípade má výstup vyjadrovať, či je daná doména indikatívna alebo nie, čo je ekvivalentné tomu, či ju zaradíme do slovníka alebo nie. Takýto typ výstupov samozrejme nie je dostupný priamym spôsobom a je potrebné ich získať. Naším cieľom je teda získať výstupy pre nejakú množinu domén, pričom táto množina bude obsahovať aj pozitívne domény (indikatívne, zaradené do slovníka), aj negatívne domény (nezaradené do slovníka).

Za týmto účelom bol navrhnutý postup, ktorý je založený na vizualizácii vektora príznakov danej domény. Vizualizácia v tomto prípade napomáha jednoduchšiemu a rýchlejšiemu rozhodnutiu sa ohľadne výstupu pre danú metódu. Ako tento proces prebiehal, si ukážeme na ilustratívnych príkladoch.

Prvý takýto ilustratívny príklad je možné vidieť na obrázku 2.3. Stĺpce s označením *user* a *ua* zodpovedajú 2 príznakom, ktoré popisujú návštevnosť. V tomto prípade hodnoty poukazujú na vysokú návštevnosť danej domény (na posúdenie tejto skutočnosti je samozrejme potrebná skúsenosť s danou problematikou a dátami). Na druhej strane je možné si všimnúť, že rozloženie vážených tokov pre jednotlivé OS je pomerne rovnomerné a zastúpené sú všetky triedy. Z tohto pohľadu je indikatívnosť domény nízka. Na základe nízkej indikatívnosti bol doméne pridelený negatívny výstup.



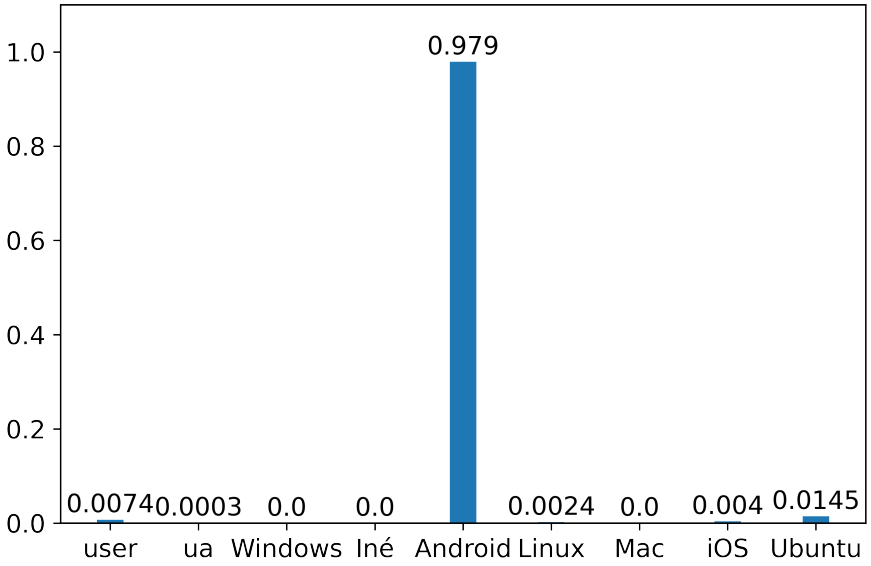
Obr. 2.3: Ilustratívny príklad vizualizácie vektora príznakov. Tento druh vizualizácie bol použitý v procese získavania výstupov pre domény.



Obr. 2.4: Ilustratívny príklad vizualizácie vektora príznakov.

V druhom príklade, ktorý je ilustrovaný na grafe 2.4, si opäť môžeme všimnúť, že návštevnosť je na vysokej úrovni (viď stĺpce *user* a *ua*). Okrem toho z pohľadu rozdelenia vážených tokov môžeme vidieť, že výrazne prevládajú dve súvisiace triedy Linux a Ubuntu. To znamená, že okrem vysokej návštevnosti je doména aj indikatívna. Preto j bol pridelený pozitívny výstup.

Na poslednom príklade (viď 2.5) je možné registrovať vizualizáciu vektora príznakov pre doménu, na ktorej je v porovnaní s predchádzajúcimi príkladmi výrazne menšia prevádzka. Napriek tomu je návštevnosť na takejto úrovni dostatočná na to, aby sme ju brali do úvahy (pre posúdenie je opäť potrebný istý náhľad do problematiky). Dôležitú úlohu v tomto samozrejme zohráva indikatívnosť, ktorá je pri pohľade na graf očividná. Z týchto dôvodov bol doméne pridelený pozitívny výstup.

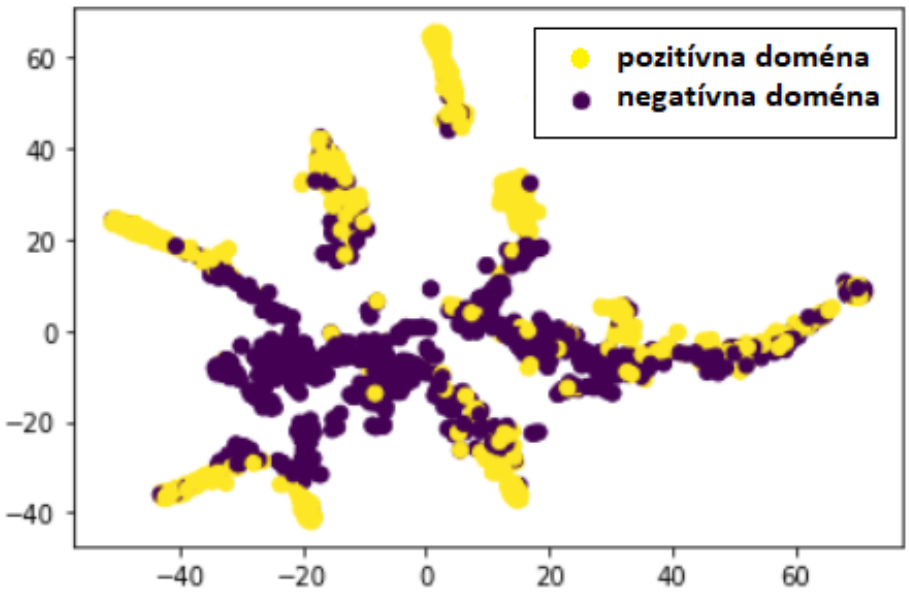


Obr. 2.5: Ilustratívny príklad vizualizácie vektora príznakov.

Postup, ktorý bol názorne vysvetlený na predchádzajúcich 3 príkladoch, bol uplatnený na získanie výstupov pre 1257 domén, pričom pozitívnych výstupov bolo 539. Tento postup si spočiatku vyžadoval zoznámenie sa s danými dátami a takisto dôležité bolo nájsť efektívny spôsob, ktorým by bolo možné získať výstupy pre dostatočný počet domén za rozumný časový úsek. Po nájdení takejto metodiky už bolo možné získať výstupy pre 100 domén za približne 15 minút expertnej manuálnej práce.

Vizualizácia

Veľmi zaujímavú vizualizáciu týkajúcu sa procesu získavania výstupov pre domény môžeme sledovať na obrázku 2.6. Táto vizualizácia vznikla použitím *t-sne* algoritmu [33], ktorý bol aplikovaný na vektory príznakov odpovedajúce doménam, pre ktoré máme výstupy získané postupom popísaným vyššie. Tento algoritmus sa využíva pre vizualizáciu dát s vyššou dimenziou. Následne boli využité získané výstupy, na základe ktorých boli jednotlivým bodom (doménam) udelené farby. Žlté body na vizualizácii odpovedajú doménam s pozitívnym výstupom a fialová farba odpovedá negatívnym doménam. Zdôraznime, že vizualizácia dát v 2-dimenzionálnom priestore vznikla iba na základe vektorov príznakov, pričom príslušné výstupy boli použité až následne pre farebné odlíšenie.



Obr. 2.6: Vizualizácia domén a ich výstupov vytvorená použitím algoritmu tsne.

Na obrázku je možné všimnúť si, že jednotlivé body tvoria zhluky a to predovšetkým mimo ťažiska dát (vizuálne na okrajoch). Okrem toho je možné pozorovať, že práve v týchto mimoťažiskových zhlukoch sa nachádza väčšina pozitívnych domén. Toto je možné interpretovať tak, že domény popísané príznakmi, ktoré sme v tejto časti práce vytvorili, tvoria isté skupiny. Okrem toho je možné vidieť súvislosť medzi týmito skupinami a našim chápaním indikatívnosti domén. To sa ukázalo tak, že väčšina pozitívnych domén sa na vizualizácii nachádzala v

zhlukoch, ktoré sa tvorili mimo ťažiska. Na základe popísaných pozorovaní sme danú vizualizáciu považovali za sľubný indikátor zmysluplnosti tohto postupu.

Klasifikačná úloha

V momente, keď sme mali k dispozícii dáta v požadovanom formáte (vektory príznakov s výstupmi), nasledovala samotná klasifikačná úloha. Cieľom tohto kroku bolo overiť, či klasifikačný algoritmus bude schopný riešiť danú úlohu s rozumnou presnosťou. Z povahy danej klasifikačnej úlohy sme sa rozhodli pre použitie náhodného lesu, pričom bola využitá implementácia *RandomForestClassifier* z modulu scikit-learn [34]. Jediným optimalizovaným parametrom bol počet stromov, inak bolo použité základné nastavenie odpovedajúce uvedenej implementácii. Výsledky dosiahnuté týmto spôsobom boli výborné. Spoľahlivosť (precision) bola na úrovni 93%, zatiaľ čo úplnosť (recall) dosahovala 87%.

2.4.2 Konštrukcia nového slovníka

V tomto štádiu už teda máme k dispozícii všetky potrebné nástroje k tomu, aby bolo možné skonštruovať nový slovník indikatívnych domén automatizovaným spôsobom. Za týmto účelom sú samozrejme opäť potrebné záznamy o sieťovej prevádzke, ktoré pochádzajú z nového časového intervalu. Z týchto záznamov sú pre všetky domény napočítané vektory príznakov rovnako, ako bolo uvedené v časti 2.4.1. Celkový počet domén bol 28491, pričom po odfiltrovaní domén, na ktorých sme registrovali iba zanedbateľnú úroveň prevádzky, sme sa dostali na 14317 domén. V nasledujúcom kroku bol natrénovaný klasifikátor, v tomto prípade náhodný les, k čomu boli použitých 1257 domén s výstupmi, ktoré boli získané postupom popísaným vyššie. Aplikovaním natrénovaného klasifikátora na novú sadu 14317 domén, sme pre tieto domény získali výstupy, pričom 1902 z nich bolo pozitívnych. Práve týchto 1902 domén s pozitívnym výstupom spoločne vytvorilo nový slovník indikatívnych domén, ktorý budeme v ďalšom priebehu práce nazývať *naučený slovník*.

Jeho použitie bude analogické ako v prípade pôvodného slovníka. To znamená, že bude využitý pre vytvorenie klasifikačných dát pre úlohu identifikácie OS, a to rovnakým spôsobom, aký bol detailne popísaný v časti 2.2.2. Porovnanie kvality nového, naučeného slovníka s pôvodným slovníkom bude predmetom skúmania v experimentálnej časti práce. Rovnako zaujímavé bude aj porovnanie so slovníkom, ktorý vznikol v rámci práce [1].

2.4.3 Potenciálne vylepšenia

Okrem nosnej myšlienky v 3. verzii navrhovaného algoritmu, ktorou je automatizácia tvorby slovníka indikatívnych domén, boli zavedené aj ďalšie potenciálne vylepšenia.

Extrakcia domény

Prvé z množiny týchto navrhovaných vylepšení spočívalo v spôsobe, ktorým sa z hostname reťazca extrahovala doména. Ako je uvedené v časti 2.2.1, v 1. verzii algoritmu bola doména braná ako tá časť reťazca, ktorá sa nachádza za prvou bodkou v tomto reťazci. Hoci tento jednoduchý spôsob fungoval vo väčšine prípadov až prekvapivo dobre, existoval tam výrazný priestor na vylepšenie. Už od začiatku výskumu sme sa zaoberali otázkou, koľko úrovní majú mať domény v slovníku indikatívnych domén.

Pojem počet úrovní domény si môžeme vysvetliť na jednoduchom príklade. Napríklad doména *com* je doména prvej úrovne, *google.com* je príklad mena domény druhej úrovne, *www.google.com* je meno domény tretej úrovne a tak podobne. Každopádne sa ukázalo, že správny počet úrovní pre domény v slovníku neexistuje a môže sa líšiť príklad od príkladu. Napríklad doména *google.com* je druhej úrovne a je navštevovaná veľmi často, avšak z hľadiska OS zariadenia nie je indikatívna. Narozdiel od toho, doména *android.clients.google.com* má 4 úrovne a je indikatívna pre Android zariadenia. Avšak nájdeme aj príklady ako *apple.com*, kde doména s 2 úrovňami je silno indikatívna. Motivovaní týmito a mnohými ďalšími príkladmi sme sa rozhodli, že z hostname reťazca budeme extrahovať doménu 2., 3. a 4. úrovne, pokiaľ je to možné. Rozsah 2. až 4. úrovne je daný tým, že domény 1. úrovne považujeme za príliš všeobecné, zatiaľ čo domény 5. úrovne za príliš špecifické.

Takáto zmena prístupu k extrakcii domény z hostname reťazca smeruje k tomu, že máme k dispozícii viac domén, pre ktoré môžeme skúmať ich indikatívnosť. Zároveň to prispieva k tomu, aby sa v slovníku indikatívnych domén vyskytovali domény so správnym počtom úrovní.

Počet aktívnych 5-minútoviek

Ďalším navrhovaným vylepšením v rámci novej verzie algoritmu je použitie počtu aktívnych 5-minútoviek ako príznaku pri klasifikácii. Ako už bolo popísané v časti 2.2.2, vektor príznakov, ktorý vyjadruje aktivitu zariadenia počas nejakého časového intervalu sa skladá zo zložiek, ktoré odpovedajú jednotlivým doménam v použitom slovníku indikatívnych domén. V 5. bode postupu, ktorý objasňuje, ako daný vektor príznakov vzniká (viď 2.2.2), je uvedené, že hodnoty

príznakov sú znormované počtom 5-minútových intervalov, počas ktorých bolo zariadenie aktívne. Navrhované vylepšenie spočíva práve v myšlienke využiť počet aktívnych 5-minútových intervalov ako nový príznak pri identifikácii OS. Hodnota tohto príznaku môže napr. zachytávať informáciu o tom, či bolo zariadenie aktívne počas celého dňa alebo iba v istých časových intervaloch a podobne.

2.4.4 Výber príznakov

Posledným potenciálnym vylepšením, ktorého použitie bolo zavedené v rámci 3. verzie algoritmu *IDB-OSId* bol tzv. výber príznakov. V strojovom učení je táto oblasť známa pod anglickým výrazom *feature selection*. V kontexte klasifikačnej úlohy ide o súčasť predspracovania dát (*data preprocessing*), kde pointou je pripraviť dáta pre klasifikačnú úlohu v takej forme, aby bolo riešenie úlohy čo najefektívnejšie z hľadiska presnosti klasifikácie, výpočtovej zložitosti a pod. Využívanie techník, ktoré súvisia s predspracovaním dát, je v strojovom učení bežnou praxou, preto je možné nájsť aj rôzne publikácie, ktoré poskytujú prehľad metód pre výber príznakov [35, 36, 37].

V našom prípade boli použité základné a pomerne jednoduché techniky predspracovania dát, resp. výberu príznakov, ktoré budú zhrnuté v nasledujúcom texte. Pokiaľ uvažujeme našu konkrétnu úlohu, tak vieme, že jednotlivé príznaky odpovedajú doménam v slovníku. Proces výberu príznakov je teda analógiou k výberu domén, ktoré budú použité už pri samotnej klasifikačnej úlohe. Z opačnej strany sa dá tento proces vnímať aj ako odfiltrovanie domén, ktoré sa pri klasifikácii používať nebudú. Takýto pohľad sa javí ako zaujímavý predovšetkým v kontexte automatického generovania slovníka. Pri tvorbe takéhoto slovníka samozrejme môže dochádzať ku nepresnostiam, ktoré vedú k tomu, že je do slovníka zaradená aj doména, ktorá nie je indikatívna. Proces výberu príznakov teda môže byť vnímaný aj ako identifikácia a odfiltrovanie domén, ktoré do slovníka nepatria.

L-1 normalizácia

Úvodným krokom pri predspracovaní dát je v našom postupe L-1 normalizácia vektorov príznakov. V tomto kroku je teda každá zložka vektora príznakov vydelená L-1 normou celého vektora. Tento krok môže byť zaujímavý napr. z hľadiska zariadení, ktoré s indikatívnymi doménami komunikovali v malej miere. To by totižto znamenalo, že L-1 norma ich vektoru príznakov je menšia v porovnaní s normou vektora príznakov pre zariadenie, ktorého komunikácia s indikatívnymi doménami prebiehala vo väčšej miere. Týmto spôsobom teda môže byť zvý-

raznená prevádzka menej aktívnych zariadení, čo teoreticky môže napomôcť ich identifikácii. Je vhodné spomenúť, že v tomto smere sme špecificky pristupovali k príznaku počtu aktívnych 5-minútoviek, ktorý sme týmto spôsobom nenormalizovali a zároveň nebol započítaný ani do normy vektora príznakov. Keďže na tento príznak môžeme pozerať ako na akúsi všeobecnú mieru aktivity zariadenia, chceme jeho hodnotu nechať zachovanú. Aby bol overený vplyv L-1 normalizácie v popísanej forme na výslednú identifikáciu OS, tak proces predspracovania dát bol otestovaný s aj bez použitia tohto kroku.

Rozptyl príznakov

Ďalšou technikou, ktorá bola využitá je výber príznakov na základe ich rozptylu. Hodnotu rozptylu pre daný príznak si zavedieme v nasledujúcej definícii.

Rozptyl príznaku. *Nech X je matica obsahujúca vektory príznakov pre jednotlivé príklady, kde X[i,j] je hodnota j-teho príznaku pre i-ty príklad v dátach. Potom rozptyl j-teho príznaku definujeme ako:*

$$f\_var(j) = var(X[:, j]),$$

kde var(.) označuje výpočet rozptylu.

Motiváciou tohto kroku je odstrániť príznaky, ktorých hodnota je naprieč príkladmi konštantná alebo sa mení len v malej miere. Typickou ukážkou takéhoto príznaku je v našom prípade doména, ktorú navštevuje málo zariadení, čo sa vo výsledku prejaví na nízkom rozptyle hodnôt daného príznaku. V praxi je možné túto metódu pre výber príznakov využiť tak, že je zvolená prahová hodnota. Pokiaľ je rozptyl pre daný príznak nižší než zvolená prahová hodnota, tak príznak už ďalej nie je používaný.

Korelácia príznaku s výstupom

Nasledujúcou metódou, ktorá bola pri výbere príznakov použitá, je korelácia príznakov s výstupom. Pre prehľadnosť a korektnosť zavedieme daný pojem v nasledujúcej definícii.

Korelácia príznaku s výstupom. *Nech X je matica obsahujúca vektory príznakov pre jednotlivé príklady a y je vektor obsahujúci výstupy pre jednotlivé príklady, kde X[i,j] je hodnota j-teho príznaku pre i-ty príklad v dátach a y[i] je výstup zodpovedajúci i-temu príkladu v dátach. Potom koreláciu j-teho príznaku s triedou výstupu l definujeme ako:*

$$corr\_w\_t(j, l) = corr(X[:, j], 1[y == l])$$

kde corr(.) označuje Pearsonovu metódu výpočtu korelácie a $1[y == l]$ *označuje vektor s dĺžkou zodpovedajúcou vektoru y, pričom i-ta zložka má hodnotu 1, pokiaľ y[i] = l (teda i-ty príklad má výstup triedu l), inak má hodnotu 0.*

Korelácia príznaku s výstupom má teda vyjadrovať súvislosť medzi hodnotou daného príznaku a triedou. V tomto zmysle sa dá na koreláciu príznaku s výstupom pozerať ako na akúsi mieru indikatívnosti daného príznaku (resp. danej domény) pre danú triedu. Výraz miera indikatívnosti treba samozrejme v tomto prípade rozumieť intuitívne. Korelácia počítaná Pearsonovou metódou naberá hodnoty z intervalu $[-1, 1]$, pričom z hľadiska indikatívnosti sú pre nás zaujímavé kladné hodnoty poukazujúce na koreláciu, ale takisto aj záporné hodnoty vyjadrujúce anti-koreláciu. Je tomu tak preto, že indikatívnosť chápeme v širšom zmysle. To znamená, že za indikatívnosť nejakej domény môžeme považovať aj to, že ju nejaké OS nenavštevujú.

Keďže pre daný príznak dostaneme hodnoty korelácie pre každú triedu, z praktických dôvodov je vhodné agregovať tieto hodnoty do jednej výslednej hodnoty, ktorú sme nazvali *korelačné skóre*. To je dané súčtom absolútnych hodnôt korelácie daného príznaku s jednotlivými triedami. Následne je už možné, podobne ako pri rozptyle, zvoliť nejakú prahovú hodnotu korelačného skóre, ktorú musí daný príznak dosiahnuť, aby bol naďalej používaný.

Korelácia medzi príznakmi

Posledným krokom v procese výberu príznakov je využitie korelácie medzi príznakmi, ktorej definícia je nasledovná.

Korelácia medzi príznakmi. *Nech X je matica obsahujúca vektory príznakov pre jednotlivé príklady, kde X[i,j] je hodnota j-teho príznaku pre i-ty príklad v dátach. Potom koreláciu medzi k-tym a l-tym príznakom definujeme ako:*

$$corr\_b\_f(k, l) = corr(X[:, k], X[:, l])$$

kde corr(.) označuje Pearsonovu metódu výpočtu korelácie.

Odstraňovanie jedného z dvojice vysoko korelovaných príznakov je v procese výberu príznakov známou praxou. V našom prípade však treba takisto upozorniť aj na súvislosť s automaticky generovaným slovníkom. Pri naučenom slovníku, ako nazývame náš automaticky generovaný slovník, totižto nie je braný do úvahy samotný názov domény, čo znamená, že sa v ňom môžu nachádzať dvojice veľmi podobných domén. Ako príklad môžeme uviesť dvojicu

domén *download.windowsupdate.com* a *au.download.windowsupdate.com*. V takomto prípade je korelácia medzi príznakmi samozrejme veľmi vysoká, čo môže mať na následnú klasifikáciu negatívny efekt. Z tohto dôvodu v našom prípade ďalej používame už iba jeden z takejto dvojice príznakov, pričom volíme príznak, ktorý má väčšie korelačné skóre (napočítané v predchádzajúcom kroku).

Celkovo tento krok vo výbere príznakov funguje tak, že je zvolená istá prahová hodnota pre koreláciu medzi príznakmi. V prípade, že pre danú dvojicu príznakov korelácia presahuje prahovú hodnotu, je táto dvojica vyhodnotená ako vysoko korelovaná. Z tejto dvojice je následne využívaný už iba príznak, ktorý dosahuje vyššie korelačné skóre.

2.4.5 Klasifikačná úloha

Po ukončení procesu výberu príznakov máme dáta vo formáte odpovedajúcom klasifikačnej úlohe. Rovnako ako v predchádzajúcich dvoch verziách algoritmu *IDB-OSId*, aj v 3. verzii záverečný krok spočíva v samotnej identifikácii OS, pričom táto úloha je formulovaná ako klasifikácia. Pre jej riešenie je opäť zvolený klasifikačný algoritmus k-NN.

Kapitola 3

Navrhované experimenty

Nasledujúca kapitola bude venovaná návrhu experimentov, ktorých cieľom je overiť kvalitu a zmysluplnosť našej metódy. V úvode kapitoly budú v krátkosti popísané techniky a nástroje potrebné pre realizáciu experimentálnej časti. Nasledovať bude samotný návrh experimentov, ktoré majú preskúmať teoretické aspekty práce z hľadiska aplikovateľnosti v praktických úlohách. Konkrétne budú navrhnuté 3 série experimentov, ktorých úlohou je overiť prínos práce z rôznych pohľadov.

3.1 Nástroje pre experimentálnu časť

3.1.1 Validačná schéma

Zásadnú úlohu pri logicky správnom využívaní princípov strojového učenia zohráva validačná schéma. V kontexte klasifikačnej úlohy je úlohou validačnej schémy zabezpečiť, aby testovanie kvality klasifikátora (evaluácia) neprebiehalo na rovnakých dátach ako proces učenia. Najjednoduchším spôsobom je rozdelenie dát na tréningovú a testovaciu časť. Tréningová časť, ktorá pozostáva z príkladov s výstupmi (anglicky samples with labels) je použitá na učenie sa klasifikačného algoritmu. Následne je úlohou klasifikačného algoritmu prideliť výstupy novým príkladom z testovacej časti dát.

Ďalšou bežne používanou možnosťou je k-behová validácia (anglicky *k-fold cross validation*). V tomto prípade sú dáta rozdelené na k častí, ktoré v angličtine nazývame *folds*. V každom behu je ako testovacia časť dát vybraná jedna z k častí, pričom ostatných $k-1$ častí tvorí trénin-

govú časť. Týmto spôsobom sa v každom behu použije pre testovanie iná časť, čím dostávame k behov.

V experimentálnej časti našej práce boli pre validáciu využívané práve tieto 2 popísané prístupy. Z povahy našej úlohy vznikajú na validačnú schému však aj dodatočné podmienky. Keďže chceme, aby naša metóda pre identifikáciu OS zariadenia fungovala v rôznych počítačových sieťach, je potrebné zabezpečiť, aby dáta z jednej siete neboli použité zároveň v tréningovej aj testovacej časti dát. To by totižto mohlo spôsobiť, že klasifikátor bude natrénovaný na jednu konkrétnu počítačovú sieť, avšak z pohľadu iných sietí nebudú dosiahnuté výsledky relevantné. Z týchto dôvodov sme v rámci našej práce využívali jeden z nasledujúcich 2 prístupov k validácii.

V prvom prípade boli jednotlivé počítačové siete rozdelené do dvoch skupín, pričom dáta z jednej skupiny sietí boli použité ako trénovacie dáta, zatiaľ čo dáta z druhej skupiny sietí boli využité pre testovanie. V druhom prípade bola použitá k-behová validácia, kde k počítačových sietí zodpovedalo k častiam dát. To znamená, že v rámci jedného behu validácie bola jedna počítačová sieť použitá pre testovanie, zatiaľ čo ostatné boli využité pre tréning.

3.1.2 Evaluácia

Za účelom evaluácie výsledkov klasifikácie boli v rámci práce využívané evaluačné metriky spoľahlivosť (precision) a úplnosť (recall). Keďže sme v našej úlohe pracovali na klasifikácii s viacerými možnými triedami/výstupmi, boli hodnoty spoľahlivosti a úplnosti napočítavané pre každú triedu. Pre danú triedu l sú spomínané evaluačné metriky definované nasledovne:

$$\textit{spoľahlivosť}(l) = \frac{TP(l)}{TP(l) + FP(l)},$$

$$\textit{úplnosť}(l) = \frac{TP(l)}{TP(l) + FN(l)},$$

kde:

TP(l) je počet príkladov, ktorým je správne pridelená trieda l,

FP(l) je počet príkladov, ktorým je nesprávne pridelená trieda l,

FN(l) je počet príkladov, ktoré majú triedu l, avšak pri klasifikácii iba bola nesprávne pridelená iná trieda.

3.2 Slovníky a datasety

Za účelom lepšej prehľadnosti v rámci experimentálnej časti práce si v tejto podkapitole zhrnieme informácie o používaných slovníkoch a datasetoch. Rovnako tak im pridelíme názvy, čo nám umožní jednoduchšiu formuláciu či už pri návrhu experimentov alebo aj pri prezentovaní výsledkov v nasledujúcej kapitole.

Slovníky

Celkovo boli v experimentálnej časti použité 3 slovníky. Prvým z nich je slovník, ktorý vznikol prevažne manuálnym vyhodnocovaním domén v rámci 1. verzie algoritmu *IDB-OSId*. Tento slovník pozostáva zo 122 domén a v ďalšom priebehu práce ho budeme nazývať *pôvodný slovník*. Druhým slovníkom je *naučený slovník*, ktorý bol vytvorený automatizovaným procesom popísaným v predchádzajúcej kapitole a ktorý je súčasťou 3. verzie navrhovaného algoritmu *IDB-OSId*. Tento slovník, ktorý je zložený z 1902 domén, reaguje na výzvy týkajúce sa automatizácie tvorby slovníka, čo sa javí ako zásadný krok v kontexte udržania jeho aktuálnosti. Posledným z trojice je slovník vychádzajúci z práce [1], ktorý obsahuje domény z databázy špecifických domén, ktorá bola vytvorená a používaná v rámci spomínanej práce. Tento slovník pozostávajúci z 98 domén budeme v ďalšom priebehu nazývať *rešeršný slovník*.

Datasety

V rámci experimentálnej časti práce boli použité 2 rôzne datasety, ktoré majú formu zodpovedajúcu klasifikačnej úlohe, ktorá vyjadruje úlohu identifikácie OS zariadenia tak, ako sme popisovali v predchádzajúcom priebehu práce.

Prvý dataset vznikol na základe pôvodného slovníka, čo znamená, že počet príznakov je 122. Presný popis, ako tento dataset vznikol je popísaný v predchádzajúcej kapitole v časti 2.2.2. Pre jeho tvorbu boli použité dáta z 10 rôznych počítačových sietí. Pri používaní tohto datasetu bola vždy použitá 10-behová validácia, kde jednotlivé počítačové siete zodpovedali jednotlivým častiam dát pri validácii. Tento dataset budeme v ďalšom priebehu práce nazývať *dataset 1*, prípadne *1. dataset*.

Druhý dataset je v porovnaní s datasetom 1 o niečo všeobecnejší, keďže je možné zvoliť, z ktorého z trojice slovníkov má vychádzať. Od tejto voľby samozrejme závisí počet príznakov. Tento krok je z hľadiska experimentálnej časti práce veľmi dôležitý, pretože umožňuje porovnanie kvality jednotlivých slovníkov. Dataset vznikol rovnakým spôsobom ako 1. dataset, teda

podľa popisu v časti 2.2.2. Jediným rozdielom je, že v druhom datasete je použitý aj príznak počtu aktívnych 5-minútoviek, pričom tento krok bol popísaný v časti 2.4.3. Pre tvorbu tohto datasetu boli použité dáta z 24 rôznych počítačových sietí. Čo sa týka validačnej schémy, pri využívaní druhého datasetu boli dáta pevne rozdelené na tréningovú a testovaciu časť. Tréningová časť pozostávala z dát zo 14 počítačových sietí, pričom dáta zo zvyšných 10 počítačových sietí boli použité pre testovaciu časť. V ďalšom priebehu práce budeme pre tento dataset využívať označenie *dataset 2*, prípade *2. dataset*.

Už v tomto momente je vhodné poznamenať, že jednotlivé datasety sa odlišujú v rôznych aspektoch. Pre ich tvorbu boli použité iné počítačové siete, čo sa prejavuje na rozložení OS zariadení a podobne. Z tohto dôvodu nie je relevantné porovnávať výsledky identifikácie OS dosiahnuté na jednom datasete s výsledkami dosiahnutými na druhom datasete.

3.3 Návrh experimentov

V nasledujúcej časti sa už budeme venovať konkrétnym návrhom experimentov. Zameriame sa pri tom na opis, ako experiment prebiehal a zároveň si priblížime jeho význam z hľadiska našej práce.

3.3.1 Základný experiment

Prvým zo série experimentov je *základný experiment*. Tento experiment je previazaný s 1. verziou navrhovaného algoritmu *IDB-OSId 1.0*, pričom jeho významom je stanoviť základné výsledky, ktoré naznačia ako navrhovaná metóda funguje, aké sú jej výhody, nevýhody či limitácie. Experiment prebiehal na datasete 1, ktorý je založený na pôvodnom slovníku, čo zodpovedá algoritmu vo verzii *IDB-OSId 1.0*. Pri klasifikácii, resp. identifikácii OS zariadení, uvažujeme 4 základné triedy - Windows, Apple, Linux a Android. Môžeme tak hovoriť o identifikácii na úrovni rodín OS. Pre riešenie klasifikačnej úlohy bol využitý algoritmus k-NN, pričom bolo využité jeho základné nastavenie vychádzajúce z implementácie v scikit-learn module [34]. Hodnoty parametra k sme pri testovaní brali z množiny $\{1, 3, 5, 7, 9, 11\}$. Ako už bolo spomenuté, pri využívaní datasetu 1 bola používaná 10-behová validácia.

3.3.2 Vplyv kontrastného učenia

Úlohou druhého experimentu je preskúmať zmysluplnosť a vplyv využitia kontrastného učenia pri vytváraní potenciálne kvalitnejších reprezentácií. Touto problematikou sme sa zaoberali v predchádzajúcej kapitole v rámci navrhovanej druhej verzie algoritmu, *IDB-OSId 2.0*. V tomto experimente sme rovnako ako pri základnom experimente vychádzali z datasetu 1, čo nám umožňuje vzájomne porovnať výsledky týchto dvoch experimentov a na základe tohto porovnania zhodnotiť vplyv kontrastného učenia.

Základné body postupu pri využití kontrastného učenia boli popísané už v časti 2.3.2, ale tento všeobecný postup je potrebné pre ujasnenie špecifikovať. Ako už bolo spomenuté v predchádzajúcej kapitole, učenie sa novej reprezentácie v tomto prípade zodpovedá trénovaniu neurónovej siete, ktorej architektúra je ilustrovaná na obrázku 2.2. Vstupná vrstva sa skladá zo 122 neurónov, čo odpovedá dimenzii datasetu 1. Nasledujú dve plne prepojené (dense) vrstvy s ReLU prenosovou funkciou (activation function), pričom obe sú zložené z 512 neurónov. Výstupná vrstva pozostáva z 256 neurónov.

Ako už bolo popísané v časti 2.3.1, pri tomto prístupe prebieha tréning neurónovej siete po tzv. šaržiach (v anglickom jazyku nazývaných *batches*), ktoré sú tvorené N pozitívnymi pármi. V našom prípade sme volili $N = 256$. Tréning spočíva v optimalizácii hodnoty stratovej funkcie **NT-Xent loss** (viď 2.3.1). Šarža bola vytvorená za pomoci generovania pozitívnych párov, pričom pozitívny pár je náhodne vybranou dvojicou zariadení s rovnakým OS (viď 2.3.2). Pri generovaní šarží bolo potrebné zvoliť, koľko pozitívnych párov bude prislúchať jednotlivým OS (triedam). V rámci tohto experimentu sme pozitívne páry pre vytvorenie šarže generovali v pomere 24 : 15 : 15 : 10 (Windows : Apple : Android : Linux). Počet šarží, ktorý bol použitý v rámci jednej epochy tréningu, je definovaný ako:

$$n\_batches = round(\frac{n\_train\_samples}{2N}),$$

kde $n\_train\_samples$ je počet príkladov v aktuálnej tréningovej časti dát a $round(.)$ označuje najbližšie celé číslo. Celkovo trénovanie pozostávalo zo 45 epoch.

Po ukončení tréningového procesu bola tréningová aj testovacia časť klasifikačných dát transformovaná pomocou natrénovanej neurónovej siete, čo znamenalo uplatnenie naučenej transformácie, ktorej výsledkom bola nová, potenciálne kvalitnejšia reprezentácia. Táto reprezentácia teda bola daná hodnotami z výstupnej vrstvy natrénovanej neurónovej siete. Reagujúc na závery prezentované v práci [29] sme testovali aj nastavenie, kedy naučená reprezentácia

bola daná hodnotami z predposlednej vrstvy natrénovanej neurónovej siete. Takýto krok sme avizovali už v predchádzajúcej kapitole.

Po aplikovaní naučenej transformácie sme už ku klasifikácii pristupovali analogicky ako v prípade základného experimentu. To znamená, že na klasifikáciu bol opäť využitý algoritmus k-NN, pričom v tomto experimente sme sa v prípade parametru k obmedzili na hodnoty z množiny $\{1, 5, 11\}$.

3.3.3 Porovnanie slovníkov

Záverečná séria experimentov bola zameraná primárne na porovnanie kvality slovníkov. Takýto experiment je veľmi zaujímavý a prínosný, pretože nám umožňuje priame porovnanie troch slovníkov - pôvodného slovníka vytvoreného v rámci algoritmu *IDB-OSId 1.0*, naučeného slovníka vychádzajúceho z algoritmu *IDB-OSId 3.0* a rešeršného slovníka pochádzajúceho z existujúceho výskumu. Týmto spôsobom je možné porovnať náš výskum s existujúcim výskumom a rovnako tak porovnať automaticky generovaný slovník s manuálne vytvoreným slovníkom. Zároveň je otestovaný aj vplyv procesu výberu príznakov, ktorý bol prezentovaný v algoritme *IDB-OSId 3.0*.

Pre tieto experimenty bol používaný dataset 2, ktorý bol pripravený v troch verziách zodpovedajúcich trom slovníkom. Ako už bolo spomenuté, v prípade 2. datasetu je rozdelenie na tréningovú a testovaciu časť pevne určené. Pre klasifikáciu bol rovnako ako v predchádzajúcich experimentoch využitý algoritmus k-NN, pričom hodnoty parametru k sú v tomto prípade brané z množiny $\{1, 5, 7, 11\}$.

V rámci tejto série prebehli najprv experimenty bez použitia procesu výberu príznakov. Následne boli vykonané takisto aj experimenty s použitím tohto procesu, čo nám umožnilo experimentálne overiť prínos daného kroku. Proces výberu príznakov, resp. predspracovania dát bol popísaný v časti 2.4.4. V krátkosti môžeme pripomenúť, že prvým krokom bola L-1 normalizácia vektorov príznakov a následne boli príznaky skúmané z hľadiska rozptylu, korelácie s výstupom a vzájomnej korelácie. Ako už bolo avizované, pri L-1 normalizácii boli vyskúšané dve nastavenia - teda buď bola použitá alebo nie. Pri výbere príznakov bolo potrebné zvoliť prahové hodnoty pre rozptyl, korelačné skóre a vzájomnú koreláciu. Prehľad vyskúšaných hodnôt parametrov je zobrazený v nasledujúcich bodoch:

- L-1 normalizácia: {*áno, nie*},
- prahové hodnoty pre rozptyl: $\{0.001, 0.0001, 0.000001\}$,

- prahové hodnoty pre korelačné skóre: $\{0, 0.001, 0.005, 0.05, 0.2\}$,
- prahové hodnoty pre vzájomnú koreláciu: $\{0.99, 0.9, 0.7\}$.

Kapitola 4

Výsledky a diskusia

V nasledujúcej kapitole sa budeme zaoberať dosiahnutými výsledkami v rámci navrhovaných experimentov, ktoré boli predstavené v predchádzajúcej kapitole. Postupne sa budeme venovať základnému experimentu, vplyvu kontrastného učenia a porovnaniu slovníkov, pričom budú prezentované a diskutované najdôležitejšie výsledky.

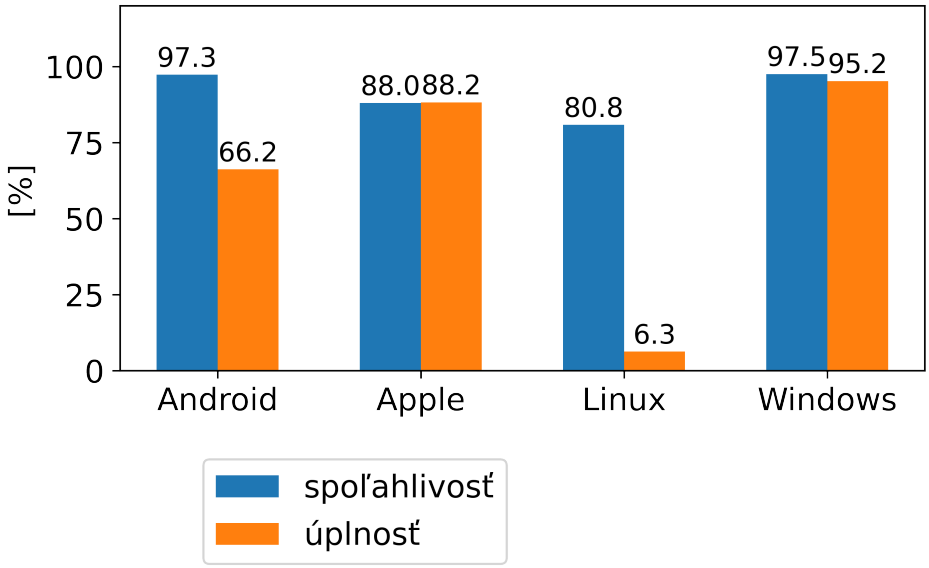
4.1 Základný experiment

Ako už bolo spomenuté v predchádzajúcej kapitole, v základnom experimente je otestovaná 1. verzia navrhovaného algoritmu, čiže *IDB-OSId 1.0*. Kvalita klasifikácie bola hodnotené na základe evaluačných metrík spoľahlivosť (precision) a úplnosť (recall).

Dosiahnuté výsledky

Dosiahnuté výsledky sú prezentované na grafe 4.1. Môžeme si všimnúť, že najlepšie výsledky boli zaznamenané pre triedu Windows, kde spoľahlivosť bola na úrovni 97.5% a úplnosť presiahla 95%. Pre triedu Apple zariadení boli výsledky takisto na veľmi dobrej úrovni, pretože spoľahlivosť aj úplnosť sa pohybovali na hranici 88%. Čo sa týka Android zariadení, v ich prípade môžeme vnímať značný rozdiel medzi spoľahlivosťou, ktorá prekračuje 97% a úplnosťou na úrovni iba 66%. To nám v praxi hovorí, že pokiaľ bolo zariadenie klasifikované ako Android, tak s vysokou spoľahlivosťou to aj Android zariadenie bolo. Na druhú stranu úplnosť naznačuje, že zo všetkých Android zariadení bolo ako Android identifikovaných iba približne 66%. Ešte markantnejší rozdiel medzi spoľahlivosťou a úplnosťou môžeme vidieť pre triedu

Linux, pre ktorú sme vo všeobecnosti mali najslabšie výsledky. Predovšetkým úplnosť dosahujúca iba 6.3% kontrastuje s ostatnými hodnotami evaluačných metrík.



Obr. 4.1: Graf ilustruje dosiahnuté výsledky klasifikácie v rámci základného experimentu.

Diskusia

Z celkového hľadiska môžeme veľmi pozitívne hodnotiť spoľahlivosť navrhovanej metódy, ktorá bola na dobrej úrovni naprieč všetkými triedami. Z pohľadu praktického použitia zohráva spoľahlivosť pri identifikácii OS zásadnú úlohu, pretože v momente, keď identifikujeme OS zariadenia danou metódou, spoľahlivosť vyjadruje, do akej miery môžeme tomuto výstupu dôverovať. Naopak, za nevýhodu či limitáciu testovaného algoritmu *IDB-OSId 1.0* môžeme považovať nízke hodnoty úplnosti pre triedy Android a Linux, pričom predovšetkým pre triedu Linux je táto limitácia veľmi výrazná. Interpretovať je ju možné tak, že daná metóda dokáže z celkového počtu Linux zariadení v dátach identifikovať iba veľmi malý podiel - 6.3%.

Rozdielnosť medzi výsledkami pre jednotlivé rodiny OS môže byť spôsobená viacerými faktormi. Prvým z potenciálnych faktorov je nevyváženosť dát (v tomto prípade nerovnomerné zastúpenie jednotlivých rodín OS), ktorá je pri klasifikačnej úlohe všeobecne známym problémom. Ďalším faktorom môže byť kvalita slovníka z pohľadu jednotlivých OS. V tomto smere

sa javí ako problematická trieda Linux, čo sa v praxi prejavilo tak, že väčšina vektorov príznakov zodpovedajúcich Linux zariadeniam bola nulová. To znamená, že tieto zariadenia počas daného časového intervalu nekomunikovali so žiadnou z domén v slovníku, čo je z pohľadu identifikácie mimoriadne problematické.

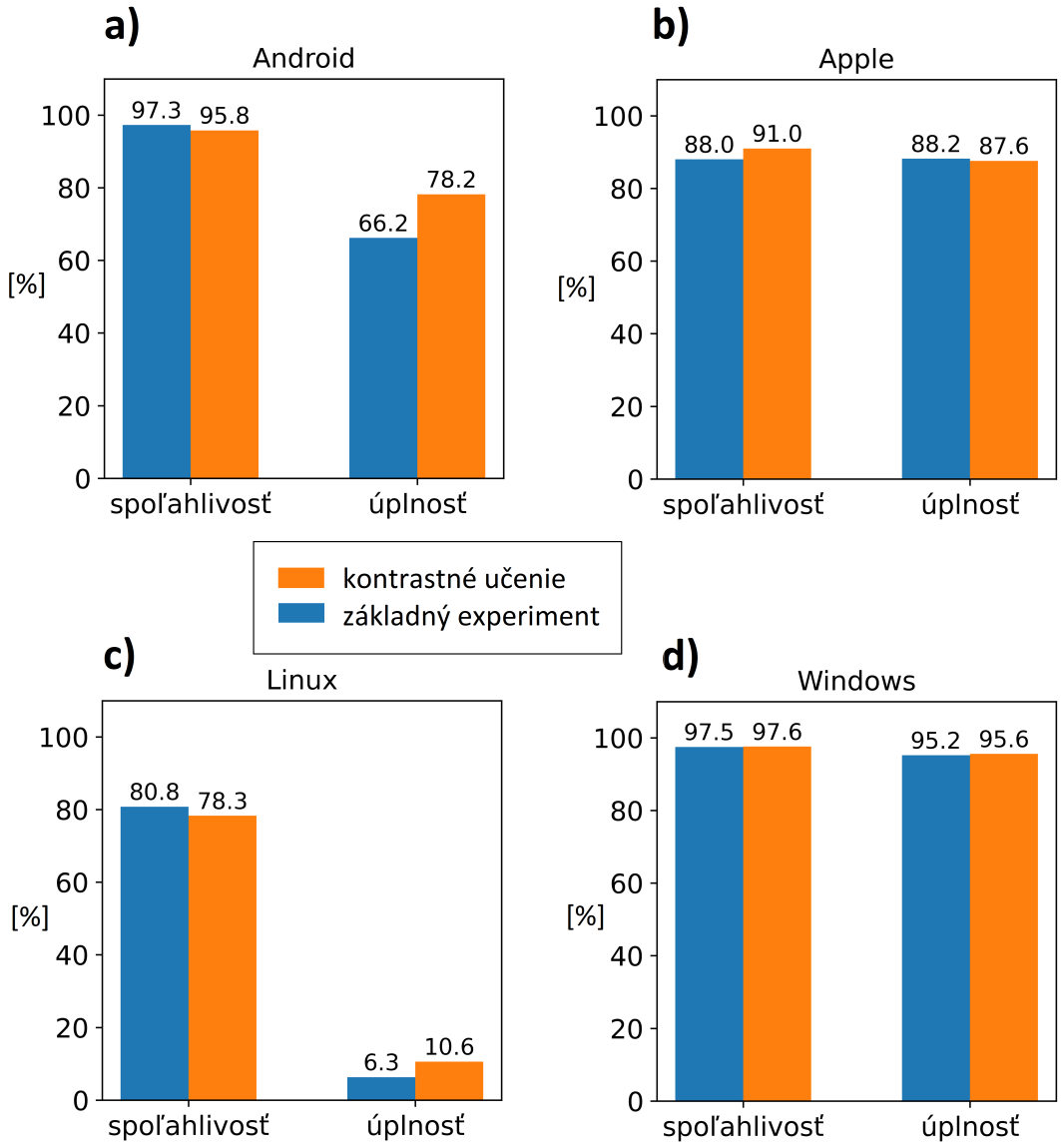
4.2 Vplyv kontrastného učenia

Hlavným cieľom 2. experimentu bolo preskúmať vplyv kontrastného učenia. Ako už bolo avizované v predchádzajúcej kapitole, v tomto experimente vychádzame z rovnakého datasetu ako v prípade základného experimentu. To nám umožňuje otestovať zmysluplnosť kontrastného učenia použitého vo forme, ktorá bola popísaná v 2. verzii navrhovaného algoritmu - *IDB-OSId 2.0*. Rovnako ako v základnom experimente, aj v tomto prípade sme pre vyhodnotenie klasifikácie využívali evaluačné metriky spoľahlivosť a úplnosť. Výsledky experimentu sú prezentované na grafoch 4.2, ktoré vyjadrujú porovnanie s výsledkami základného experimentu. Z praktických dôvodov sú výsledky pre jednotlivé triedy zobrazené v osobitných grafoch.

Dosiahnuté výsledky

Z pohľadu triedy Android môžeme na 4.2 v časti a) pozorovať výrazne zlepšenie v úplnosti - zo 66% na 78%. Takéto zlepšenie treba samozrejme hodnotiť pozitívne, predovšetkým v kontexte toho, že práve hodnoty úplnosti pre triedy Linux a Android sa v základnom experimente javili ako najproblematickejšie. Na druhú stranu je potrebné poznamenať, že v spoľahlivosti došlo ku poklesu o 1.5%. Napriek tomu spoľahlivosť dosahuje takmer 96%, čo je výborný výsledok. Čo sa týka triedy Apple zariadení (viď 4.2 graf b)), tak z hľadiska spoľahlivosti došlo ku zlepšeniu o 3%, čím sme sa dostali nad hranicu 90%. Z pohľadu úplnosti sme nezaznamenali výraznejšie zmeny.

Rovnako tak výraznejšie zmeny, či už z hľadiska spoľahlivosti alebo presnosti, neboli pozorované pre Windows zariadenia, o čom je možné presvedčiť sa na grafe d). O rozdielnej situácii môžeme hovoriť v prípade Linux zariadení, kde bolo zaznamenané výrazné zlepšenie v úplnosti. Z absolútneho hľadiska zmena predstavuje iba 4.3%, avšak relatívne ide o zmenu približne 68%. Podobne ako v prípade triedy Android, aj tu došlo ku miernemu poklesu spoľahlivosti z 80.8% na 78.3%.



Obr. 4.2: Grafy ilustrujú dosiahnuté výsledky klasifikácie v rámci experimentu, ktorý skúma vplyv kontrastného učenia.

Diskusia

Z celkového hľadiska môžeme využitie kontrastného učenia hodnotiť pozitívne, avšak je potrebné poznamenať, že zlepšenie oproti základnému experimentu bolo iba mierne. Najvýraz-

nejšie a zároveň pozitívne zmeny boli zaznamenané v rámci hodnôt úplnosti pre triedy Linux a Android. Takéto zlepšenie je zaujímavé predovšetkým z toho hľadiska, že práve hodnoty úplnosti pre triedy Linux a Android boli v rámci základného experimentu najväčšou limitáciou. Z tohto pohľadu môžeme zhodnotiť, že navrhovaná adaptácia myšlienok kontrastného učenia z oblasti počítačového videnia je zmysluplná. Inak povedané, učenie sa novej reprezentácie (využitím kontrastného učenia) má potenciál vylepšiť výsledky následnej klasifikácie. Je však potrebné zvážiť, či v kontexte zvýšenej výpočtovej a časovej náročnosti je potenciálne zlepšenie dostatočné.

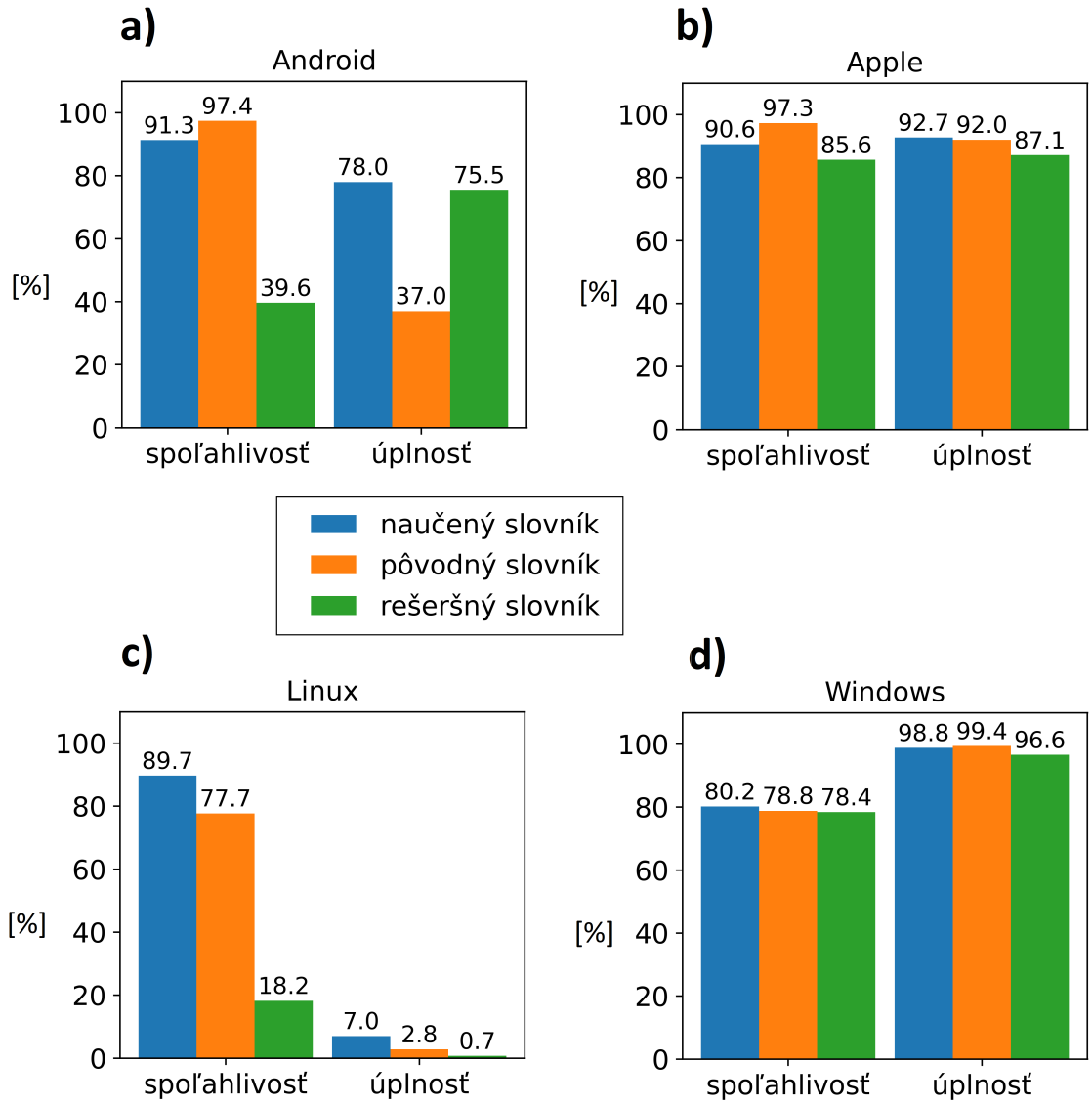
Z tohto pohľadu považujeme za vhodné zamerať sa na rôzne parametre, ktorých voľba je potrebná pre realizáciu daného prístupu. Medzi tieto parametre môžeme zaradiť veľkosť šarže, počet pozitívnych párov pre jednotlivé OS v rámci šarže, počet epoch tréningu, parameter teploty v stratovej funkcie, atď. Takisto zaujímavou možnosťou pre ďalší výskum je otestovanie iných architektúr pre neurónovú sieť z hľadiska počtu a typov vrstiev, prenosových funkcií a podobne. Z časových dôvodov bola v rámci tejto práce otestovaná iba úzka škála nastavení pre tieto parametre, čo považujeme za hlavnú limitáciu tejto verzie metódy. Na druhú stranu, táto limitácia vytvára priestor pre ďalší výskum, ktorý je potrebný pre odhalenie plného potenciálu navrhovanej adaptácie kontrastného učenia.

4.3 Porovnanie slovníkov a výber príznakov

Posledná séria experimentov je zameraná na niekoľko cieľov. Prvým z nich je porovnanie kvality trojice slovníkov - naučeného, pôvodného a rešeršného. Zmyslom porovnania naučeného a pôvodného slovníka je zistiť, či automatizácia procesu tvorby slovníka je zmysluplná. Porovnania s rešeršným slovníkom majú zas zaistiť zrovnanie s existujúcim výskumom. Druhá časť experimentov sa bude zaoberať vplyvom výberu príznakov, ktorý bol navrhnutý v rámci algoritmu *IDB-OSId 3.0*.

4.3.1 Porovnanie slovníkov

Experiment prebiehal na 2. datasete, ktorý existuje v 3 verziách zodpovedajúcich 3 slovníkom, pričom vždy obsahuje rovnakú sadu zariadení, čo zaručuje objektívne porovnanie. Výsledky dosiahnuté v rámci tohto experimentu sú zobrazené na grafoch 4.3. Za účelom prehľadnosti sú výsledky pre jednotlivé triedy rozdelené do samostatných grafov.



Obr. 4.3: Grafy ilustrujú porovnanie kvality jednotlivých slovníkov na základe výsledkov klasifikácie.

Dosiahnuté výsledky

Na grafe a) (viď 4.3) je možné vidieť výsledky pre triedu Android. Môžeme si všimnúť, že pre naučený slovník sme dostali veľmi kvalitné výsledky, kde spoľahlivosť presiahla 91%, zatiaľ čo úplnosť bola na úrovni 78%. V porovnaní s pôvodným slovníkom je síce spoľahlivosť

o 6% nižšia, avšak stále je nad hranicou 90%. Na druhú stranu, hodnota úplnosti je pri použití naučeného slovníka o vyše 40% lepšia ako v prípade pôvodného slovníka. Čo sa týka rešeršného slovníka, tam sú výsledky výrazne slabšie, predovšetkým z hľadiska spoľahlivosti, ktorá je iba na úrovni 40%.

Pre Apple zariadenia (viď 4.3 graf b)) sú dosiahnuté výsledky vo všeobecnosti výborné. Najlepšie výsledky môžeme vidieť pre pôvodný slovník, kde spoľahlivosť presiahla 97%, zatiaľ čo úplnosť je na úrovni 92%. Takisto kvalitné výsledky boli zaznamenané aj pre naučený slovník, pretože hodnoty oboch metrík boli nad hranicou 90%. O niečo slabšie, avšak stále kvalitné výsledky môžeme vidieť aj v prípade rešeršného slovníka.

Z hľadiska Linux zariadení je situácia výrazne odlišná. Najlepšie výsledky boli zaznamenané pre naučený slovník, pri ktorého použití spoľahlivosť atakovala hranicu 90%. Naproti tomu, v prípade pôvodného slovníka je spoľahlivosť o 12% nižšia. Čo sa týka úplnosti, tam sú dosiahnuté výsledky vo všeobecnosti slabé, s čím sa stretli už počas predchádzajúcich experimentov. Je však potrebné dodať, že v prípade naučeného slovníka je úplnosť v relatívnom porovnaní s ostatnými dvoma slovníkmi omnoho lepšia.

Na grafe d) (viď 4.3) sú zobrazené výsledky pre triedu Windows, kde nie sú výraznejšie rozdiely medzi jednotlivými slovníkmi. Pritom spoľahlivosť sa pohybuje na úrovni 80%, zatiaľ čo úplnosť vo všetkých prípadoch presahuje 96%.

Diskusia

Dosiahnuté výsledky z hľadiska nášho výskumu pôsobia veľmi pozitívne. Vo všeobecnosti najvyrovnanejšie a najkvalitnejšie výsledky boli totižto zaznamenané pri použití naučeného slovníka. To znamená, že automatizácia tvorby slovníka navrhnutá v záverečnej verzii našej metódy (*IDB-OSId 3.0*) funguje veľmi dobre. Takýto krok teda prináša do výskumu dve významné pozitíva. Za prvé, automatizácia tvorby slovníka je zásadným krokom smerom k tomu, aby bolo možné pravidelne slovník aktualizovať. A za druhé, ako ukazujú dosiahnuté výsledky, kvalita automaticky generovaného slovníka môže byť dokonca lepšia v porovnaní s manuálne vytvoreným slovníkom.

Rovnako pozitívne je možné hodnotiť porovnania s rešeršným slovníkom, ktorý vychádza z existujúceho výskumu. V prípade tohto slovníka boli dosiahnuté výsledky nevyrovnané a výrazne slabšie. Naopak, ako negatívny výsledok je opäť potrebné zmieniť hodnoty úplnosti v prípade Linux zariadení. Tá sa javí ako problematická aj pri využívaní nového naučeného slovníka a zostáva tak hlavnou limitáciou navrhovanej metódy.

4.3.2 Vplyv výberu príznakov

Ako už bolo avizované, v druhej časti záverečnej série experimentov bude diskutovaný vplyv výberu príznakov, ktorého použitie je popísané v časti 2.4.4. Najprv bude otestovaný na dátach, ktoré vychádzajú z naučeného slovníka, čím budú spoločne vyskúšané všetky modifikácie navrhované v 3. verzii našej metódy - *IDB-OSId 3.0*. Následne bude vplyv výberu príznakov overený aj s použitím ostatných dvoch slovníkov.

Naučený slovník

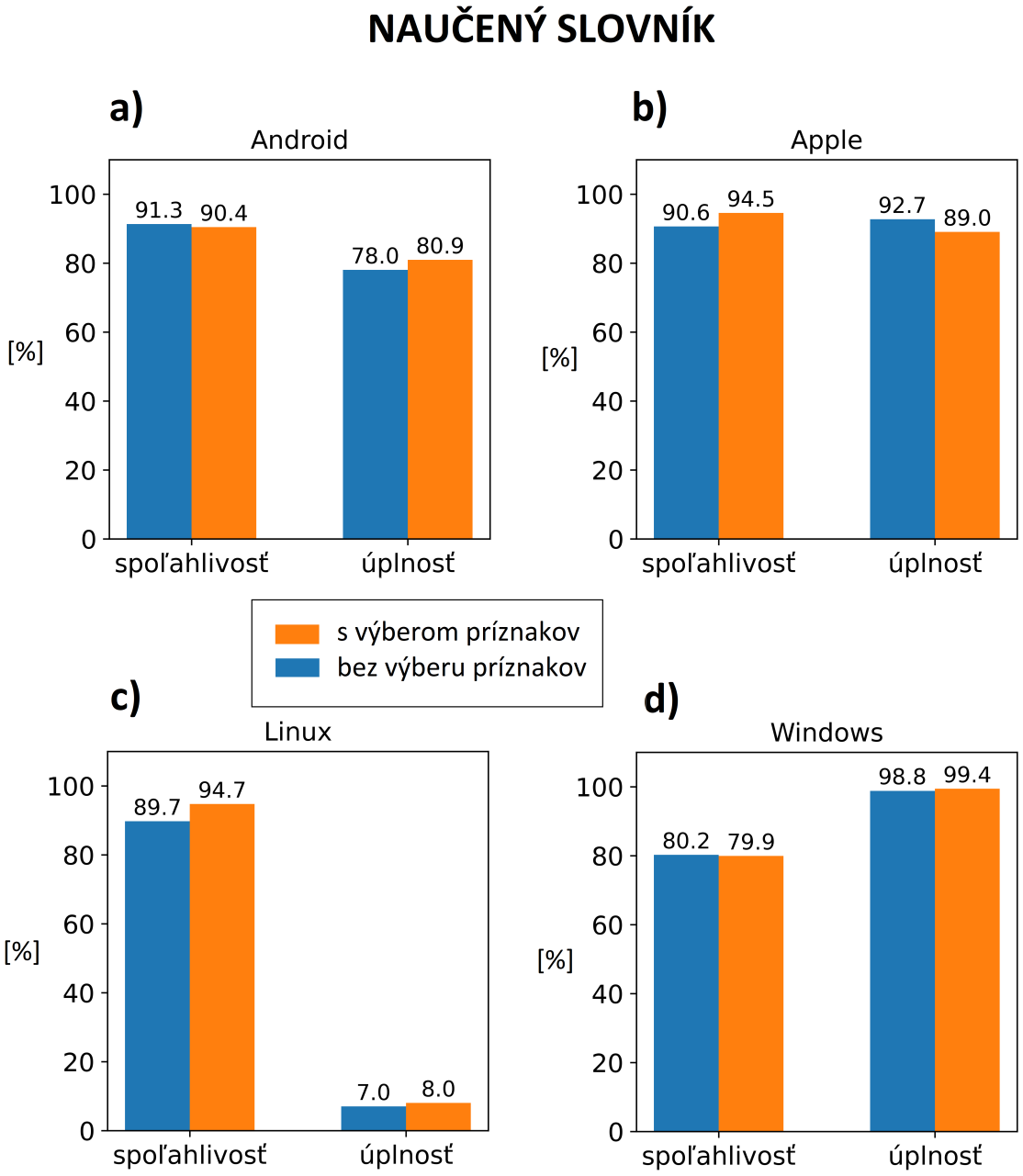
Podľa dosiahnutých výsledkov, ktoré sú prezentované na grafoch 4.4, sa dá vplyv výberu príznakov hodnotiť pozitívne. Zlepšenie je možné pozorovať predovšetkým na hodnotách spoľahlivosti pre triedy Apple a Linux, ktoré s použitím výberu príznakov atakujú hranicu 95%. Mierne zlepšenie je možné vidieť aj v úplnosti pre triedy Android a Linux. Naopak mierne zhoršenie pozorujeme v úplnosti pre triedu Apple. Z celkového hľadiska prevažujú pozitívne zmeny, čo potvrdzuje zmysluplnosť tohto kroku. Rovnako tak je potrebné pripomenúť, že výberom príznakov sa znižuje dimenzia vektorov príznakov, čo vo výsledku znižuje výpočtovú náročnosť klasifikačnej úlohy.

Pôvodný slovník

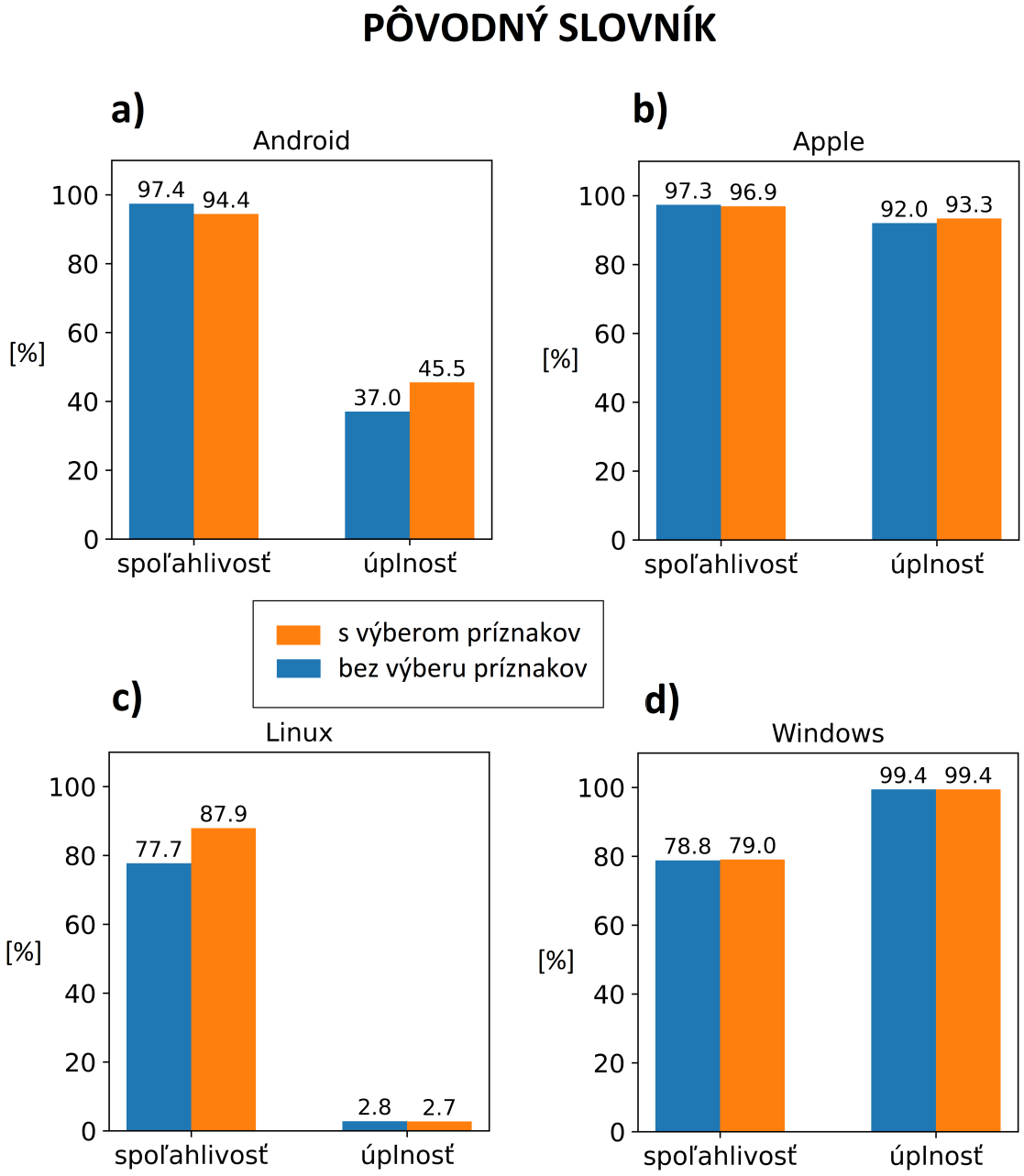
Výsledky, ktoré ilustrujú vplyv výberu príznakov pri použití pôvodného slovníku sú zobrazené na grafoch 4.5. Rovnako ako v prípade naučeného slovníka, aj tu môžeme pozorovať celkové mierne zlepšenie. Najvýraznejšie zlepšenia boli zaznamenané pre spoľahlivosť triedy Linux a úplnosť triedy Android. Naopak, k miernemu zhoršeniu došlo v spoľahlivosti pre triedu Android.

Rešeršný slovník

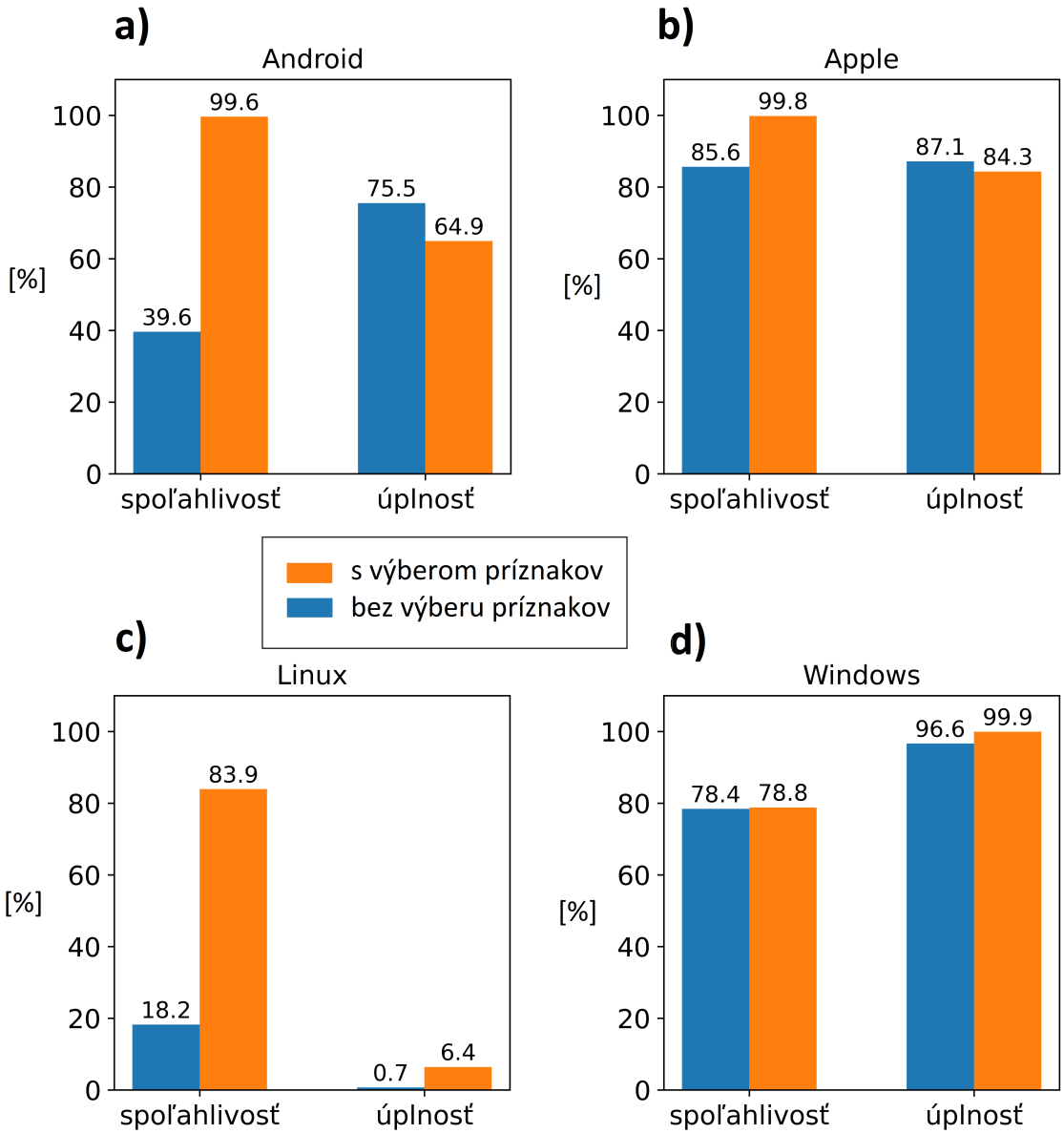
Najvýraznejší a zároveň veľmi pozitívny vplyv výberu príznakov bol zaznamenaný pri použití rešeršného slovníka, čo dokumentujú grafy zobrazené na 4.6. Môžeme si všimnúť, že rozdiely v hodnotách evaluačných metrík sú markantné. Predovšetkým hodnoty spoľahlivosti pre triedy Android a Apple stoja za povšimnutie, pretože prekračujú 99%. Zaujímavosťou je, že v tomto prípade bolo pri procese výberu príznakov zvolených iba 17 príznakov (z pôvodného počtu 99), na základe ktorých potom prebehla klasifikácia. Zlepšené výsledky pritom poukazujú na to, ako efektívny môže výber príznakov byť.



Obr. 4.4: Grafy ilustrujú vplyv využitia procesu výberu príznakov pri použití naučeného slovníka na základe výsledkov dosiahnutých v klasifikácii.



Obr. 4.5: Grafy ilustrujú vplyv využitia procesu výberu príznakov pri použití pôvodného slovníka na základe výsledkov dosiahnutých v klasifikácii.



Obr. 4.6: Grafy ilustrujú vplyv využitia procesu výberu príznakov pri použití rešeršného slovníka na základe výsledkov dosiahnutých v klasifikácii.

4.4 Záverečná diskusia

Z celkového hľadiska priniesli navrhnuté experimenty a dosiahnuté výsledky niekoľko zaujímavých pozorovaní, ktoré budú prediskutované v tejto záverečnej podkapitole. Základný experiment načrtol silné aj slabé stránky navrhovaného algoritmu. Za výhodu môžeme určite označiť pomerne vysokú spoľahlivosť metódy naprieč všetkými triedami, zatiaľ čo nevýhodou bola nižšia úplnosť, špeciálne pre triedu Linux. Hoci sa algoritmus v jednotlivých verziách úspešne vyvíjal, tieto charakteristiky zostali zachované. Za touto skutočnosťou identifikujeme dva hlavné dôvody. Prvým je nevyváženosť výskytu OS v počítačových sieťach, čo je prirodzený fenomén, ktorý identifikáciu OS sťažuje. Druhým potenciálnym dôvodom je kvalita slovníka indikatívnych domén. Indikatívne domény sa vo všeobecnosti vyhľadávali lepšie pre Windows a Apple zariadenia. V týchto prípadoch bola identifikácia kandidátov (na indikatívne domény) omnoho jednoduchšia než napríklad pre Linux zariadenia.

V druhom experimente sme sa zamerali na otestovanie adaptácie myšlienok kontrastného učenia na našu úlohu. Ako už bolo spomenuté, tento prístup naznačil istý potenciál, čo sa prejavilo na miernom zlepšení dosiahnutých výsledkov oproti základnému experimentu. Je však nutné poznamenať, že pre odhalenie plného potenciálu tohto prístupu by bolo potrebné rozsiahlejšie experimentálne preskúšanie, čo môže byť námetom pre ďalšiu výskumnú prácu.

Najzaujímavejšie pozorovania však vyplynuli z 3. série experimentov. Veľmi zaujímavé bolo porovnanie jednotlivých slovníkov na totožnej sade zariadení. Z tohto porovnania boli najlepšie výsledky zaznamenané pre naučený slovník. Tento slovník vznikol aplikovaním strojového učenia na úlohu výberu indikatívnych domén na základe zaznamenaných štatistík. Takáto automatizácia znamená výrazný pokrok oproti manuálnemu vyhľadávaniu indikatívnych domén, čo je jednou z hlavných limitácii v 1. verzii nášho algoritmu - *IDB-OSId 1.0*, ale zároveň napríklad aj v práci [1]. Automatické vyhľadávanie indikatívnych domén je zásadným krokom k tomu, aby bol slovník indikatívnych domén aktuálny a aby sa navrhovaná metóda dokázala adaptovať na zmeny v názvoch domén a podobne. Okrem toho sa ukázalo, že pre automaticky generovaný slovník sme dostali lepšie výsledky než pre slovníky, ktoré vznikli prevažne manuálnym vyhodnocovaním - pôvodný a rešeršný slovník. Z týchto dôvodov považujeme práve tento návrh automatizácie vyhľadávania indikatívnych domén za hlavný a výrazný prínos našej práce.

Pozitívne sa dá hodnotiť aj proces výberu príznakov, ktorý viedol k zlepšeniu výsledkov klasifikácie pri použití so všetkými troma slovníkmi, čo potvrdilo, že tento krok môže byť všeobecne prínosný. Obzvlášť zaujímavé výsledky boli dosiahnuté pri použití na dáta vytvorené na

základe rešeršného slovníka, kde použitie procesu výberu príznakov viedlo k signifikantnému vylepšeniu.

Na druhú stranu je potrebné spomenúť takisto hlavné nevýhody či limitácie nášho prístupu. Prvou z nich je nízka hodnota úplnosti pre triedu Linux naprieč všetkými experimentami. Túto nevýhodu sme si už bližšie rozobrali v úvode tejto podkapitoly. Druhou limitáciou navrhovanej metódy je všeobecnosť tried OS. V našom prípade rozlišujeme iba 4 základné triedy - Windows, Apple, Android a Linux. Námetom pre ďalší výskum môže preto byť práve väčšia špecifikácia tried, napríklad rozdelenie triedy Apple zariadení na Mac OS X a iOS zariadenia. Takisto zaujímavé by mohlo byť identifikovať najčastejšie Linux distribúcie. Práve týmto smerom sa môže uberať ďalší výskum, v ktorom by mohlo byť veľmi prínosné rozvinúť a zdokonaliť myšlienku automatického vyhľadávania indikatívnych domén.

Záver

Prvá fáza diplomovej práce spočívala v rešeršnej činnosti, ktorá bola zameraná na preskúmanie existujúcich metód pre identifikáciu OS zariadenia. Prehľad týchto metód je poskytnutý v kapitole 1, pričom bolo využité rozdelenie algoritmov do skupín na základe zdroja informácií, ktorý je používaný pre identifikáciu OS. Druhá časť kapitoly 1 bola zameraná na porovnanie metód. Špeciálna pozornosť bola venovaná preskúmaniu vplyvu šifrovania sieťovej komunikácie na jednotlivé algoritmy, čo následne umožnilo posúdenie ich perspektívnosti.

Kapitola 2 zachytáva druhú fázu našej práce, ktorá pozostávala z teoretického návrhu algoritmu pre identifikáciu OS. Navrhovaná metóda *IDB-OSId* je založená na perspektívnom princípe analyzovania komunikácie zariadenia s tzv. indikatívnymi doménami. Tento algoritmus je schopný pracovať aj s veľmi obmedzenými zdrojmi informácii (teda aj v prípade šifrovanej komunikácie), čo bolo jedným z hlavných cieľov práce. Algoritmus zároveň využíva techniky strojového učenia, keď formuluje identifikáciu OS ako klasifikačnú úlohu. Kapitola 2 takisto dokumentuje postupný vývoj metódy, ktorý je zachytený v jej troch verziách.

V rámci kapitoly 3 boli navrhnuté 3 série experimentov, ktorých cieľom bolo overiť teoretické aspekty práce na dátach z rôznych počítačových sietí. Výsledky týchto experimentov boli prezentované a diskutované v kapitole 4, pričom sme dospeli k nasledujúcim dôležitým pozorovaniam. Už základný experiment naznačil hlavné výhody a nevýhody navrhovaného algoritmu. Metóda sa vyznačovala pomerne vysokou úrovňou spoľahlivosti pre všetky klasifikované triedy, zatiaľ čo hlavnou nevýhodou bola nízka úplnosť pre triedu Linux.

Druhá séria experimentov sa zaoberala vplyvom kontrastného učenia, ktorého adaptácia na našu úlohu bola súčasťou teoretickej práce. Dosiahnuté výsledky naznačili možnosti využitia kontrastného učenia, avšak zároveň odhalili potrebu rozsiahlejšieho experimentálneho otestovania pre určenie plného potenciálu tohto kroku.

Najvýznamnejšie a súčasne veľmi pozitívne výsledky boli dosiahnuté v 3. sérii experimentov. Hlavným cieľom týchto experimentov bolo porovnanie slovníkov indikatívnych domén, pričom najlepšie výsledky boli zaznamenané pre naučený slovník. Je potrebné poznamenať, že

tento slovník vznikol na základe automatického procesu vyhľadávania indikatívnych domén, ktorý bol vytvorený v teoretickej časti našej práce. Automatické vyhľadávanie indikatívnych domén je pritom zásadným krokom v kontexte udržiavania aktuálnosti slovníka, čo sa javilo ako jedna z hlavných výziev v našom, ale aj súvisiacom výskume. Navyše pri využití tohto automaticky vytvoreného slovníka boli zaznamenané lepšie výsledky než pri použití manuálne vytvorených slovníkov či už v našej (pôvodný slovník) alebo súvisiacej práci (rešeršný slovník). Z týchto dôvodov považujeme návrh procesu automatickej tvorby slovníku za výrazný prínos našej práce.

Spomedzi pozitívnych výsledkov je takisto potrebné spomenúť proces výberu príznakov, ktorý bol v tejto konkrétnej podobe navrhnutý v rámci teoretickej časti práce. Jeho použitia sa osvedčilo pri použití s každým slovníkom, na čo poukázali výsledky 2. časti 3. série experimentov. Naopak, medzi limitáciami navrhovanej metódy je potrebné uviesť všeobecnosť tried, ktoré sú výstupom z identifikácie OS. Z pohľadu ďalšieho výskumu môže byť zaujímavé otestovať a prípadne adaptovať metódu práve na viac špecifické triedy, čo by mohlo byť značným prínosom. Budúci výskum by sa takisto mohol zaoberať zdokonalením automatického procesu vyhľadávania indikatívnych domén.

Zadanie práce bolo splnené, pričom je potrebné vyzdvihnúť automatizáciu procesu tvorby slovníka indikatívnych domén, ktorú je možné hodnotiť ako hlavný prínos práce v danej problematike. Okrem toho sú v práci poskytnuté námety pre ďalší výskum a potenciálne zlepšenie.

Literatúra

[1] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive os fingerprinting methods in the jungle of wireless networks," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2018.

[2] D. H. Hagos, M. Løland, A. Yazidi, Ø. Kure, and P. E. Engelstad, "Advanced passive operating system fingerprinting using machine learning and deep learning," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–11, IEEE, 2020.

[3] M. Laštovička, A. Dufka, and J. Komárková, "Machine learning fingerprinting methods in cyber security domain: Which one to use?," in *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*, pp. 542–547, 2018.

[4] A. Dufka, "Comparison of machine learning methods for operating system identification," Brno, 2018. Bachelor's Thesis. Faculty of Informatics Masaryk University. Vedoucí práce Martin Laštovička.

[5] A. Ornaghi, M. Valleri, E. Escobar, E. Milam, G. Costamagna, and A. Koeppe, "The ettercap project," [online]. 2015 [cit. 2021-3-26]. Dostupné z: https://ettercap.github.io/ettercap/index.html.

[6] M. Zalewski, "P0f v3," [online]. 2000 [cit. 2021-3-26]. Dostupné z: http://lcamtuf.coredump.cx/p0f3/.

[7] R. Spangler, "Analysis of remote active operating system fingerprinting tools," [online]. 2003 [cit. 2021-3-26]. Dostupné z: http://packetwatch.net/documents/papers/osdetection.pdf.

[8] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Proceedings of the First USENIX Workshop on Offensive Technologies*, WOOT '07, (USA), USENIX Association, 2007.

[9] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein, "Passive operating system identification from tcp/ip packet headers," in *Workshop on Data Mining for Computer Security*, vol. 40, Citeseer, 2003.

[10] R. Tyagi, T. Paul, B. Manoj, and B. Thanudas, "Packet inspection for unauthorized os detection in enterprises," *IEEE Security & Privacy*, vol. 13, no. 4, pp. 60–65, 2015.

[11] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, 2001.

[12] W. Loh, "Classification and regression trees," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[13] S. Marsland, *Machine learning: an algorithmic perspective*. 2nd ed. Boca Raton: CRC Press, c2009. Chapman & Hall/CRC machine learning & pattern recognition series. ISBN 978-1-4200-6718-7.

[14] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, [online]. 13(2), 415-425 [cit. 2021-3-30]. ISSN 10459227. Dostupné z: doi:10.1109/72.991427.

[15] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," [online]. 2008 [cit. 2021-3-30]. Dostupné z: https://www.hjp.at/doc/rfc/rfc5246.html.

[16] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport layer security (tls) extensions," [online]. 2003 [cit. 2021-3-30]. Dostupné z: https://www.hjp.at/doc/rfc/rfc4366.html.

[17] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passiveosfingerprint," [online]. 2018 [cit. 2021-1-25]. Dostupné z: https://github.com/CSIRT-MU/PassiveOSFingerprint.

[18] T. Matsunaka, A. Yamada, and A. Kubota, "Passive os fingerprinting by dns traffic analysis," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 243–250, IEEE, 2013.

[19] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," [online]. 2014 [cit. 2021-3-30]. Dostupné z: https://www.hjp.at/doc/rfc/rfc7231.html.

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Larry, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," [online]. 1999 [cit. 2021-3-30]. Dostupné z: https://www.hjp.at/doc/rfc/rfc2616.html.

[21] S. Lindsey, "A python implementation of the ua parser," [online]. 2009 [cit. 2021-4-5]. Dostupné z: https://github.com/ua-parser/uap-python.

[22] K. Nichols, S. Blake, F. Baker, and D. Black, "Rfc2474: Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers," [online]. 1998 [cit. 2021-3-30]. Dostupné z: https://dl.acm.org/doi/pdf/10.17487/RFC2474.

[23] R. Droms *et al.*, "Dynamic host configuration protocol," [online]. 1997 [cit. 2021-3-30]. Dostupné z: https://www.hjp.at/doc/rfc/rfc2131.html.

[24] E. Kollmann, "Chatter on the wire: A look at dhcp traffic," [online]. 2007 [cit. 2021-3-30]. Dostupné z: https://studylib.net/doc/8428933/chatter-on-the-wire–a-look-at-dhcp-traffic.

[25] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, [online]. 2009, 45(4), 427-437 [cit. 2021-5-1]. ISSN 03064573. Dostupné z: doi:10.1016/j.ipm.2009.03.002.

[26] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," [online]. 2015 [cit. 2021-3-30]. Dostupné z: https://www.hjp.at/doc/rfc/rfc7540.html.

[27] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, pp. 183–196, 2017.

[28] A. Kott, C. Wang, and R. F. Erbacher, *Cyber Defense and Situational Awareness*. [online]. Cham: Springer International Publishing, 2014 [cit. 2021-4-5]. Advances in Information Security. ISBN 978-3-319-11390-6. Dostupné z: doi:10.1007/978-3-319-11391-3.

[29] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 1597–1607, PMLR, 13–18 Jul 2020.

[30] A. Chaudhary, "The illustrated simclr framework," [online]. 2020 [cit. 2021-4-5]. Dostupné z: https://amitness.com/2020/03/illustrated-simclr.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[32] P. Sayak, "Simclr-in-tensorflow-2," [online]. 2020 [cit. 2021-4-6]. Dostupné z: https://github.com/sayakpaul/SimCLR-in-TensorFlow-2.

[33] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Machine learning proceedings 1992*, pp. 249–256, Elsevier, 1992.

[36] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[37] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.