FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Identifikace webového obsahu v šifrovaném provozu
Student:	Bc. Marek Mařík
Vedoucí:	Ing. Karel Klouda, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem práce je prozkoumat možnosti přibližné identifikace webových stránek na základě rozšířených síťových toků (IP flow) reprezentujících šifrovaný provoz. Součástí práce je vytvoření trénovacího a testovacího datasetu (např. pomocí nástroje Joy [1]) a rešerše příslušných výzkumných prací.

Zaměřte se na dva úkoly:
1) Pro zachycený HTTP/S provoz se pokuste najít příslušné mapovaní dvojic "IP flow" a URL, čímž vznikne trénovací a testovací dataset.
2) Na základě tohoto datasetu vytvořte model, který se pro daný záznam šifrovaného provozu pokusí uhodnout kompletní URL.
3) Navrhněte skript, který na základě pozorovaného šifrovaného provozu co nejpřesněji odhadne strukturu stránky (počet obrázků, javascriptové soubory, atp.) s použití například statistických metod nebo strojového učení.
4) Navržené řešení otestujte pomocí vytvořeného datasetu.

Seznam odborné literatury

[1] https://github.com/cisco/joy
[2] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. 2010. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP '10). IEEE Computer Society, USA, 191–206.
[3] Blake Anderson, Subharthi Paul, David A. McGrew: Deciphering malware's use of TLS (without decryption). J. Computer Virology and Hacking Techniques 14(3): 195-211 (2018)

Ing. Karel Klouda, Ph.D.
vedoucí katedry

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

V Praze dne 14. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Identifikace webového obsahu v šifrovaném provozu

Bc. Marek Mařík

Katedra aplikované matematiky
Vedoucí práce: Ing. Karel Klouda, Ph.D.

17. září 2020

Poděkování

Rád bych poděkoval panu Ing. Karlu Kloudovi, Ph.D. za vedení této práce, cenné rady a podněty. Dále bych rád poděkoval členům Laboratoře monitorování síťového provozu Ing. Tomáši Čejkovi, Ph.D a Ing. Karlu Hynkovi za rady a pomoc s tvorbou datových sad. V neposlední řadě bych rád poděkoval mé rodině a přítelkyni Barboře Jeriové za podporu a nekonečnou trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 17. září 2020 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2020 Marek Mařík. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Mařík, Marek. *Identifikace webového obsahu v šifrovaném provozu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá tím, zda je možné ze síťového provozu určit, jaké webové stránky byly uživatelem navštíveny i přes to, že komunikace probíhá šifrovaným způsobem. Dále pak tím, zda je možné alespoň přibližně určit obsah webové stránky z šifrovaného síťového provozu. To vše na základě charakteristik síťových toků, tedy aniž by byl provoz dešifrován.

V rámci této práce byl navrhnut a implementován generátor datových sad, který umožňuje vytvářet datové sady, které obsahují zachycené síťové toky pro návštěvy jednotlivých webových stránek. S pomocí tohoto generátoru byly vytvořeny dvě datové sady.

Byla navržena rozmanitá sada příznaků. Na základě vektorů příznaků byly provedeny experimenty s použitím různých modelů pro identifikaci webových stránek a odhad jejich obsahu. Dále byly vytvořeny modely, jejichž úkolem je detekce neznámých webových stránek.

Z provedených experimentů vyplývá, že na základě šifrovaného provozu lze poměrně přesně identifikovat webové stránky a dokonce i odhadnout některé atributy jejich obsahu.

Klíčová slova identifikace provozu, identifikace webových stránek, webový provoz, šifrovaný provoz, HTTPS, záznam šifrovaného provozu, klasifikace, regrese, strojové učení

Abstract

This master thesis deals with whether it is possible to determine from network traffic which websites were visited by the user despite the fact that the communication takes place in an encrypted way. Furthermore, whether it is possible to at least approximately determine the content of the web page from encrypted network traffic. All this based on the characteristics of network flows, i.e. without the traffic being decrypted.

As part of this work, a data set generator was designed and implemented, which allows to create data sets that contain captured network flows for visits to individual websites. Two datasets were created using this generator. A diverse set of features has been designed. Based on the features vectors, experiments were performed using multiple different models to identify websites and estimate their content. Furthermore, novelty detection models were created to detect unknown web pages.

Experiments show that based on encrypted traffic, websites can be relatively accurately identified and some attributes of their content can be estimated as well.

Keywords traffic identification, website identification, web traffic, encrypted traffic, HTTPS, encrypted traffic record, classification, regression, machine learning

Obsah

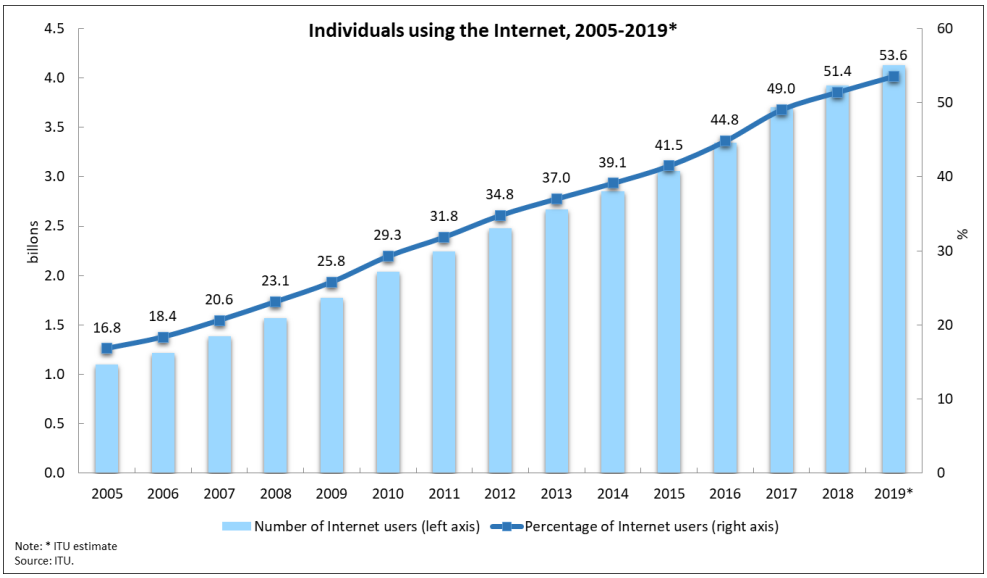
Seznam obrázků

Seznam tabulek

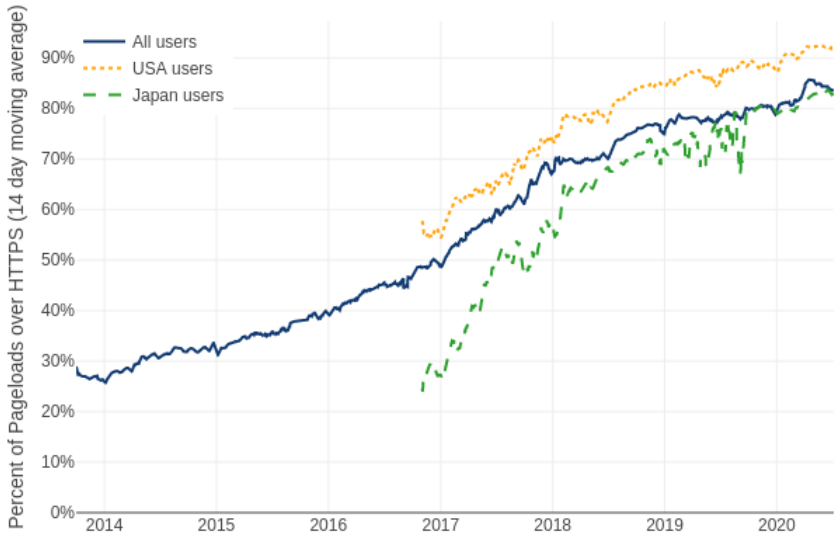
Úvod

Již od samotných počátků počítačové sítě Internet [1] v sedmdesátých letech minulého století se počet uživatelů prakticky neustále zvyšuje. Podle odhadů Mezinárodní telekomunikační unie [2] mělo na konci roku 2019 používat internet 53,6 % světové populace, tj. 4,1 miliardy lidí. S tím jak narůstá počet uživatelů internetu, narůstá i množství různých služeb poskytovaných přes internet.

Jedním z nejpoužívanějších protokolů pro komunikaci je protokol HTTP. „HTTP (Hypertext Transfer Protocol) je internetový protokol určený pro komunikaci s WWW servery.“ [3] Tento protokol používá architekturu ko-



Obrázek 0.1: Vývoj počtu uživatelů internetu dle Mezinárodní telekomunikační unie. Zdroj: [2].



Obrázek 0.2: Procentuální podíl webových stránek načítaných pomocí zabezpečeného protokolu HTTPS. Zdroj: [4].

munikace klient-server. Klientem je obvykle webový prohlížeč. Mezi klientem a serverem probíhá komunikace na základě požadavku klienta, klient odešle požadavek serveru a server mu odešle zpět odpověď. Tato komunikace, ale není zabezpečená. Protokol HTTP totiž neumožňuje šifrování ani zabezpečení integrity dat.

Z důvodu ochrany soukromí a zvýšení bezpečnosti se v dnešní době velmi často používá protokol HTTPS [3]. A jak lze vidět z grafu 0.2, tak se tento protokol používá pro načítání webových stránek čím dál častěji. Tento protokol je zabezpečenou variantou protokolu HTTP. Protokol HTTPS zajišťuje důvěrnost přenášených dat, integritu přenášených dat a autentizaci.

Motivace

Navzdory důležitým snahám a technickým prostředkům sloužícím k zachování soukromí uživatelů a ochraně důvěrnosti dat, je mnohdy kriticky důležitá schopnost odhadnout obsah komunikace pro zajištění bezpečnosti. Tato viditelnost do síťového provozu umožňuje detekovat různé bezpečnostní hrozby, jako je malware či podezřelé chování uživatele, které by mohlo ohrožovat buď technické prostředky a nebo i společnost.

Z toho důvodu si tato práce klade za cíl prozkoumat, do jaké míry je možné odhadnout přibližný obsah komunikace přenášené pomocí zabezpečených síťových protokolů. Pro jednoduchost se práce omezuje na modelový případ,

při kterém vyhodnocujeme, zda-li je možné s vysokou přesností identifikovat cílovou webovou stránku na známém serveru, na kterou přistupuje uživatel pomocí svého webového prohlížeče s využitím šifrované komunikace. Z pohledu bezpečnosti bychom si mohli představit, že bezpečnostní tým má přehled o nakažené nebo jinak nebezpečné cílové webové stránce a nad šifrovaným provozem se snaží rozhodnout, zda klient tuto konkrétní nebezpečnou adresu navštívil. Přirozeným předpokladem je, že šifrovací protokol této analýze brání a identifikaci obsahu znemožňuje.

Za předpokladu, že by například teroristická organizace hledala informace okolo výroby výbušnin, je velice důležité včasně tuto aktivitu odhalit. V případě našeho zjednodušeného prostředí experimentů a analýzy jsme se konkrétně omezili na celosvětově známou internetovou encyklopedii Wikipedia, která obsahuje velké množství různorodých stránek s bohatým obsahem. Některé stránky mohou obsahovat kontroverzní obsah a nebo mohou být předmětem cenzury v některých oblastech světa. Z našeho pohledu je proto zajímavé, jestli jsme pomocí strojového učení schopni identifikovat, na které stránky uživatel přistoupil. Konkrétní příklad potom může být detekce přístupu na stránky na serveru Wikipedia, které se týkají nukleárních zbraní, například `https://cs.wikipedia.org/wiki/Jaderná_zbraň`.

Cíl práce

Cílem této diplomové práce je pokusit se zjistit, zda je možné z šifrovaného provozu odhadnout aktivitu uživatele, který komunikuje po síti. Zjistit, zda je možné přibližně identifikovat webovou stránku na základě síťové komunikace, aniž by byla dešifrována. Pokusím se také zjistit, jestli lze odhadnout strukturu webové stránky na základě šifrovaného toku dat, jestli je například reálné odhadnout počet obrázků nebo javascriptových souborů na webové stránce. Vybrané metody implementuji a experimentálně ověřím jejich úspěšnost. Výsledky experimentů mohou vést ke zjištěním, která nám mohou dát tušit, zda je i komunikace zabezpečená pomocí současných technik, náchylná například k ohrožení soukromí uživatele.

Struktura práce

Tato práce se skládá ze tří hlavních kapitol. V kapitole 1 nejdříve stručně popíši základní myšlenky webové komunikace po počítačové síti, jak vlastně taková komunikace probíhá a vypadá. V následující podkapitole se zaměřím na identifikaci webového provozu. Poté představím několik prací, které souvisejí s touto diplomovou prací. Nakonec představím několik přístupů k řešení problémů, kterých se tato práce týká, jeden z konceptů zvolím a jeho výběr odůvodním.

Kapitola 2 se týká dat použitých v této práci. V první sekci popíši zdroj vstupních dat. V sekci druhé pak provedu analýzu návrhu generátoru datových sad. V následující sekci druhé kapitoly popíši návrh a implementaci programu pro získávání dat. V předposlední podkapitole popíši, jak jsme vytvářeli datové sady. A nakonec popíši data, která jsme pomocí implementovaného programu získali.

V kapitole 3 pak představím provedené experimenty. První podkapitola představuje hrubý návrh tvorby experimentů. V další části jsou zanalyzována vstupní data. Následující sekce se zabývají realizací samotných experimentů a v poslední sekci budou diskutovány výsledky experimentů.

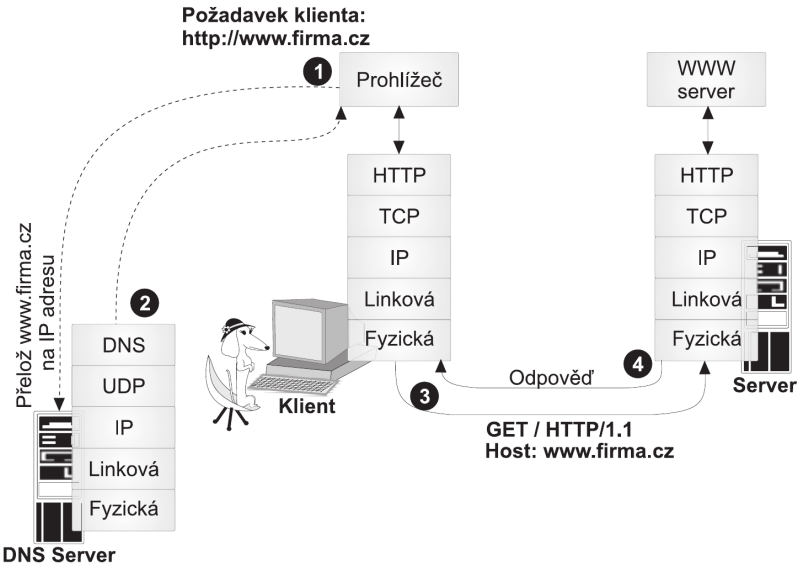
Kapitola 1

Analýza

1.1 Webový provoz

Webový obsah je popsán pomocí jazyka HTML nebo jeho variant. Součástí webu mohou být další soubory, jako například kaskádové styly (CSS), skripty nebo multimediální soubory. Přenos obsahu mezi počítači probíhá pomocí protokolu HTTP, případně HTTPS.

Protokol HTTP pracuje na aplikační vrstvě modelu ISO/OSI [5]. Jak již bylo v úvodu zmíněno, komunikace v protokolu HTTP je typu klient-server. Na obrázku 1.1 je ilustrována komunikace s přímým spojením mezi klientem a serverem pomocí protokolu HTTP. Komunikace začíná okamžikem, kdy uživatel zapíše do okna prohlížeče webovou adresu (URL) stránky, kterou chce



Obrázek 1.1: Ilustrace fungování protokolu HTTP. Zdroj: [3].

prohlížet. Webový prohlížeč ze zadané URL vybere jméno serveru, které pomocí služby DNS přeloží na IP adresu webového serveru, toto je v obrázku 1.1 naznačené jako kroky 1 a 2. Klient poté s IP adresou webového serveru naváže spojení pomocí protokolu TCP. Protokol TCP je jedním z protokolů transportní vrstvy modelu ISO/OSI. Následně prohlížeč pomocí vytvořeného TCP spojení odešle HTTP dotaz (krok 3 v obrázku 1.1). Webový server tento dotaz zpracuje a odešle v témže spojení odpověď (krok 4). Odpověď je poté zpracována a zobrazena prohlížečem.

U žádného z datových přenosů, které nastávají v krocích 1, 2, 3 a 4 předchozího diagramu, nejsou data po síti přenášena zabezpečeně. Takto přenášená data by mohla být útočníkem zneužita či napadena. Tento problém pro kroky 3 a 4 řeší protokol HTTPS. V případě kroků 1 a 2 je tento problém řešen pomocí protokolů DNS over HTTPS a DNS over TLS. Tato práce se zabývá komunikací v krocích 3 a 4, zabývá se tím, zda je možné alespoň přibližně odhadnout obsah komunikace i přes to, že je pro přenos dat v krocích 3 a 4 použit zabezpečený protokol HTTPS.

Stejně tak jako protokol HTTP, slouží protokol HTTPS především pro komunikaci webového prohlížeče s webovým serverem. V HTTPS je komunikační protokol šifrován pomocí kryptografického protokolu TLS nebo dříve pomocí SSL. Hlavním přínosem HTTPS oproti HTTP je ochrana soukromí, autentizace přístupového webu a zajištění integrity přenášených dat.

Protokol HTTP typicky komunikuje pomocí TCP na portu 80. Podobně jako HTTP komunikuje protokol HTTPS také téměř vždy pomocí TCP, ale na portu 443.

Více informací lze nalézt v knize [3], ze které jsem v této kapitole čerpal.

1.2 Identifikace webového provozu a odhad struktury obsahu webové stránky

Jedním z cílů této práce je přibližně identifikovat webovou stránku na základě síťové komunikace, aniž by byla tato komunikace dešifrována. Identifikace musí probíhat na základě nějaké předchozí znalosti. Musíme mít webový provoz s čím porovnat, abychom určili jaká webová stránka byla navštívena. Pro identifikaci webových stránek tedy bude potřeba vytvořit model, který bude naučen pomocí dvojic „webový provoz“ a „webová adresa“. Když budeme mít nový záznam síťového webového provozu a budeme chtít určit adresu webové stránky, jejímuž navštívení záznam síťového provozu odpovídá, použijeme model, který komunikaci „porovná“ s předem naučenými komunikacemi a vrátí webovou adresu, která této komunikaci odpovídá nejpravděpodobněji.

Jedná se tedy o klasifikační problém. Klasifikace je ve strojovém učení a statistice druh problému, kde je cílem na základě množiny trénovacích dat, zařadit nový vzorek do jedné nebo více kategorií. Přičemž, pro jednotlivé vzorky z trénovací množiny jsou známy kategorie do kterých náleží. V případě

této práce jsou třídami webové adresy. Lze předpokládat, že tříd bude velké množství. Počet vzorků na třídu bude nejspíše poměrně malý.

Druhý úkol, odhadnutí struktury webového obsahu, konkrétně odhad počtu obrázků či skriptů na webové stránce, je regresním problémem. Označení regresní analýza se používá pro statistické metody, které umožňují na základě známých veličin odhadovat hodnotu jiné veličiny. Model, který slouží pro řešení tohoto problému se někdy nazývá regresor. Odhadovaná proměnná se označuje jako cílová proměnná či vysvětlovaná proměnná. Známé veličiny jsou pak označovány jako vysvětlující proměnné. V případě této práce jsou vysvětlovanými proměnnými počet obrázků na webové stránce a počet skriptů na webové stránce.

Aby bylo možné webový provoz identifikovat, je potřeba ho nejdříve zachytit. Provoz zachycený pomocí nástrojů pro analýzu síťového provozu je obvykle ve formě jednotlivých paketů. To se ale pro tuto úlohu nehodí, protože data v paketech obsažená jsou zašifrovaná a samostatné pakety nepřinášejí takovou informaci, abychom na základě toho mohli klasifikovat webové stránky. Tato práce se snaží využít charakteristik toku paketů během komunikace mezi klientem a serverem, a na základě nich provoz klasifikovat.

Slovem **komunikace** bude v této práci označován veškerý přenos dat po síti během určité doby. Například jako „webovou komunikaci“ budu označovat veškerá webová data přenesená při provedení dotazu a načtení webové stránky. Někdy vynechám slovo „webovou“, protože tato práce je primárně o přenosu webového obsahu a tudíž je ve většině případů zřejmé, že se jedná o webovou komunikaci. Jako **tok** budu označovat to, co je v [6] označováno anglickým slovem „flow“. Flow je v textu [6] definováno jako „sada paketů se společnými charakteristikami. Společné charakteristiky paketů se nazývají flow key.“ Jako „flow key“ ve zmiňovaném textu používají zdrojovou a cílovou IP adresu, zdrojový a cílový port (pro provoz TCP a UDP) a číslo protokolu.

Identifikovat webový provoz a odhadnout strukturu webového obsahu, se pokusím na základě těchto toků.

1.3 Související práce

V této sekci jsou popsány některé práce související s touto diplomovou prací. Ne všechny se však nutně zabývají identifikací webového provozu.

V článku *Statistical Identification of Encrypted Web Browsing Traffic* [7] se autoři zaměřují na identifikaci šifrovaného webového HTTPS provozu. V práci identifikují webové stránky na základě četnosti a velikosti HTTP objektů. Autoři klasifikovali webové stránky, pomocí podobnostních vzdáleností mezi signaturami jednotlivých webových komunikací. Jako metriku použili Jaccardův koeficient. V rámci této práce autoři vytvořili datovou sadu, která obsahuje 100 000 webových stránek. Pouze část však byla považována za známé, ne-

boli „cílové stránky“, tedy stránky, které by měli být mezi ostatními stránkami identifikovány.

V textu je mimo jiné věnována pozornost tomu, jaké jsou teoretické možnosti identifikace webových stránek. A autoři došli k závěru, že typický vzorek síťového provozu, interpretovaný jako neuspořádaná množina HTTP objektů, který byl zapříčiněn načtením webové stránky, obsahuje 67 bitů informace, určitě více než je potřeba (teoreticky) k rozpoznání všech webových stránek dostupných na celém internetu.

Na datové sadě se sto tisíci webovými stránkami, z toho 2 191 cílovými stránkami, bylo dosaženo přesnosti identifikace přibližně 75 % a falešně pozitivní míry přibližně 1,5 %.

Text *Mobile Apps identification based on network flows* [8] se zabývá identifikací mobilních aplikací na základě šifrovaných síťových toků. Konkrétně šesticí různých často používaných aplikací jako je Facebook, Skype, Youtube a další. Text popisuje postup předzpracování síťových toků a získání sady příznaků. Příznaků je celkem 28, jsou to údaje o počtu přijatých paketů/bajtů, odeslaných paketů/bajtů, poměry mezi přijatými a odeslanými bajty/pakety a různé varianty výsledků statistických funkcí (průměr, medián, rozptyl) pro počet bajtů, počet paketů a časy mezi pakety. Pro klasifikaci byl použit klasifikátor Bagged Trees. Bylo dosaženo přesnosti klasifikace přibližně 93 % s nízkou hodnotou falešně pozitivní míry, méně než 0,5 %.

Práce *Deciphering Malware's use of TLS (without Decryption)* [9] je věnována identifikaci malwaru komunikujícího po síti šifrovaným způsobem pomocí TLS. Jsou zde použity tři skupiny příznaků. První skupina zahrnuje příznaky podobné těm v práci [8], tedy například počet příchozích bajtů, odchozích bajtů, příchozích paketů, odchozí paketů, zdrojové a cílové porty, a celkové trvání toku v sekundách. Druhou skupinou jsou hodnoty vyjadřující distribuci velikostí paketů. A poslední skupinou příznaků jsou prvky matice přechodů, která vychází z reprezentace velikostí paketů v komunikaci pomocí Markovova řetězce. Pro klasifikaci byl použit klasifikátor Logistic Regression.

Studie *Fingerprinting encrypted network traffic types using machine learning* [11] se zabývá identifikací typu provozu. Autoři práce vytvořili vlastní dataset, který obsahoval tyto druhy provozu: VoIP, Bittorrent, HTTP a YouTube stream. Autoři se pokusili tento provoz identifikovat v jak nešifrovaném, tak i v provozu šifrovaném pomocí techonlogie TOR. Klasifikace probíhala na základě manuálně definovaných příznaků. Použity byly tři různé klasifikátory: Naive Bayes, Logistic Regression a Random Forest. Všechny tři klasifikátory si vedly dobře, všechny dosáhly při klasifikaci šifrovaného provozu přesnosti kolem 80 %.

Publikace *Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow* [10] se zabývá narušením soukromí uživatelů webových aplikací, a to i přes použití zabezpečené komunikace pomocí protokolu HTTPS. Neposkytuje žádný konkrétní model, který by byl použit například pro identifikaci webových stránek, jako je cílem této diplomové práce. Ale přináší velmi

detailní analýzu narušení soukromí uživatele při používání několika různých webových aplikací. Zaměřuje se na webové aplikace, které udržují stav klienta v rámci aplikace. Využívá znalosti struktury webové aplikace a stavu uživatele k odhadu stavu nového, do kterého uživatel přešel pomocí nějaké akce, jen za pomocí velikostí paketů v komunikaci vyvolané akcí.

Práce *Traffic Analysis of the HTTP Protocol over TLS* [12], se podobně jako [10] zabývá ohrožením soukromí uživatelů webu. Analyzuje útoky, které využívají informace o délce dat přenesených protokolem HTTPS, s cílem propojit klienty s konkrétními prostředky, ke kterým na webu přistoupili. Ukazuje, kolik informací může útočník odvodit z webových dotazů, i když zná pouze délku komunikace. Poté je použit skrytý Markovův model, který analyzuje posloupnosti požadavků a vyhledává, k jakým zdrojům bylo přistoupeno nejpravděpodobněji. Použitá technika předpokládá znalost struktury webu. Autor v práci předpokládá, že uživatelé budou spíše následovat strukturu webu pomocí odkazů než ručně přistupovat k náhodným stránkám.

1.4 Koncept způsobu identifikace webových stránek a odhadu jejich obsahu

V této kapitole je diskutována volba konceptu způsobu identifikace webových stránek a odhadu jejich obsahu.

V případě identifikace webových stránek připadá v úvahu několik konceptů jejich klasifikace. Prvním z nich je použití nějakého druhu umělé neuronové sítě. Neuronové sítě zažívají v posledních letech boom. V některých případech vykazují velmi dobré výsledky, ale to neplatí vždy. Vždy je třeba zvážit konkretní situaci.

V případě této práce však neuronová síť příliš vhodná není, protože umělé neuronové sítě pro řešení klasifikačních problémů obyčejně mají tolik neuronů, kolik je tříd [14]. Ale v případě klasifikačního problému, který je řešen v této práci, jedna webová adresa odpovídá jedné kategorii a tudíž kategorií může být tříd velký počet. Identifikovaných webových stránek, tedy kategorií, tedy umělých neuronů ve výstupní vrstvě mohou být stovky, tisíce nebo i mnohem více. Tím pádem by výsledná neuronová síť byla velmi rozsáhlá. S tím souvisí i další problém, že neuronové sítě obecně potřebují pro naučení velké množství dat. Tím že, má tento klasifikační problém tolik tříd, je pravděpodobné, že bychom neměli dostatek trénovacích dat.

Druhým možným konceptem klasifikace by bylo, považovat toky za časové řady a pak například pomocí často používané kombinace algoritmů DTW a KNN [18] vzorky klasifikovat. DTW v tomto případě slouží jako metrika pro výpočet vzdáleností mezi časovými řadami, čehož využívá algoritmus KNN. Zde, ale nastává několik potíží. První problém je, že se komunikace skládají z více toků. Jak bychom tento problém vyřešili? Použili bychom jen nejdelší tok? Spojili bychom toky v jeden? Další problém je, že se algoritmy

pro porovnávání časových řad nepříliš dobře vyrovnávají s tím, když mají řady rozdílné délky. Řešit tento problém do jisté míry lze, jako například v práci [13], která obsahuje přehled metod pro klasifikaci časových řad s různými délkami. Řešení této komplikace, ale bývá vykoupeno vyšší výpočetní náročností či nižší přesností. Nevýhodou metod, které používají algoritmus KNN je také, nutnost uchovávat všechny trénovací vzorky, vysoká výpočetní náročnost samotné klasifikace, jelikož pro každý klasifikovaný vzorek je nutné vypočíst vzdálenost ke všem trénovacím vzorkům. Výpočetní náročnost KNN lze do určité míry redukovat některými pokročilejšími variantami, například využitím K–D stromu.

Dalším možným konceptem by byl přístup postavený na vektorech příznaků. Pro klasifikaci webové komunikace, by komunikace nejdříve byla pomocí nějaké transformace převedena na vektor příznaků. Takový vektor příznaků má vždy stejnou délku. Na takové vektory již lze použít některý z běžně používaných klasifikátorů. Za nevýhodu tohoto přístupu by mohlo být považováno to, že je třeba vhodně zvolit vektor manuálně definovaných příznaků.

Pro tuto práci jsem zvolil poslední zmíněný přístup, protože netrpí problémy jako první dva přístupy a protože také práce [8] a [9], zmíněné v předchozí podkapitole, tento přístup použily pro řešení obdobných problémů, jako řeší tato práce. Výsledky těchto prací byly dobré a to dává tušit, že by tento přístup mohl dobře fungovat.

V případě odhadu struktury webového obsahu použiji stejný princip jako v případě identifikace webových stránek, jen místo klasifikátoru použiji některý z nástrojů regresní analýzy.

Kapitola 2

Konstrukce datových sad

Pro implementaci, ladění, experimenty a vyhodnocení výkonnosti vytvořených modelů jsem potřeboval vhodnou datovou sadu. Bylo třeba, aby vstupní dataset obsahoval záznamy šifrovaného webového provozu a ke každému záznamu metadata. Pro splnění stanovených cílů bylo zapotřebí, aby jako metadata obsahoval alespoň kompletní URL, počet obrázků a počet javascriptových souborů na dané webové stránce.

2.1 Zdroj vstupních dat

Nepodařilo se mi nalézt žádný volně dostupný dataset, který by splňoval požadavky uvedené výše. Rozhodl jsem se nasbírat si data pro tuto práci sám. Jako zdroj vstupních dat jsem se rozhodl použít webovou online encyklopedii Wikipedia, protože je to rozsáhlá stránka s relativně pestrým obsahem.

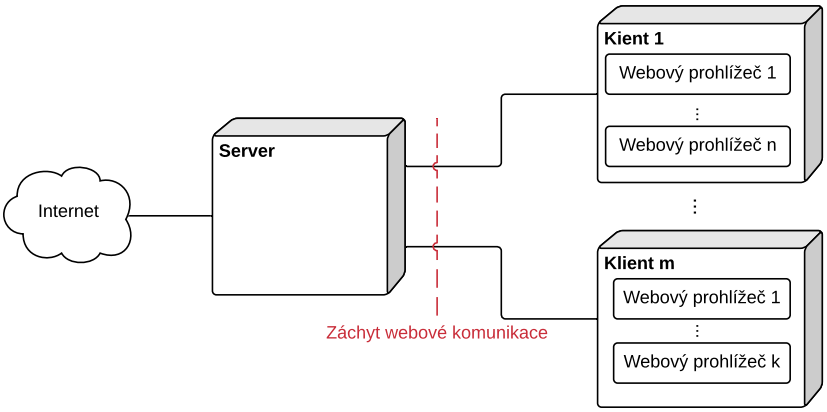
2.2 Generátor datových sad

Tato sekce se věnuje realizaci tvorby generátoru datových sad. V první podsekci popisuji jakým způsobem a proč jsem program navrhl. V druhé sekci se věnuji konkrétní implementaci.

2.2.1 Návrh

Základní myšlenka vytvořeného generátoru je taková, že generátor postupně navštěvuje webové stránky a během každé návštěvy zaznamenává, transformuje a ukládá webovou komunikaci. Ke každé zaznamenanému a uloženému toku dat ukládá metadata, jako je URL navštívené webové stránky, počet obrázků na stránce, počet javascriptových souborů stažených při načítání stránky a další.

Abych získal reálnou webovou komunikaci bylo nutné pro navštěvování webových stránek použít reálný webový prohlížeč. Požadavkem na vytvářený



Obrázek 2.1: Orientační návrh architektury generátoru datových sad.

generátor datových sad tedy bylo, aby pro navštěvování webových stránek byl používán reálný webový prohlížeč. Dalším požadavkem bylo, aby kvůli větší pestrosti výstupních dat, bylo možné použít více různých internetových prohlížečů.

Podobný požadavek platí i pro operační systém, na kterém by byl spuštěný onen prohlížeč či prohlížeče, tj. aby bylo možné data generovat pomocí různých operačních systémů.

Rozhodl jsem se pro architekturu typu klient–server, kdy by bylo možné provozovat vícero klientů a díky tomu více různý webových prohlížečů na vícero operačních systémech. Síťovou komunikaci by pak bylo možné zachytávat na síťovém rozhraní každého klienta, čímž by byl vyřešen problém s oddělením komunikací, na rozdíl od případu, kdy by byly všechny webové prohlížeče provozovány na jednom stroji. Návrh architektury lze vidět na diagramu 2.1.

Jak se dozvíte v následujícím odstavci, výstupem generátoru nebudou zaznamenané toky tak, jak data putují sítí, ale pouze posloupnosti příznaků, které popisují jednotlivé toky. Pokud bude zřejmé oč se jedná, tak takovou posloupnost příznaků budu pro jednoduchost také nazývat tokem.

Požadavkem na výstup generátoru bylo, aby záznam komunikace byl v takové formě, že by pro každý paket v toku obsahoval $n$tici příznaků, přičemž nás zajímají pouze pakety webové komunikace (protokol HTTPS komunikuje na portu 443). Pro další práci s vzniklou datovou sadou není třeba, aby byl uložen konkrétní obsah paketů, stačí příznaky. Vznikne tedy posloupnost (zachovává se pořadí paketů) $n$tic příznaků. Takovými příznaky by měly být položky jako směr toku paketu, velikost paketu a čas odeslání/přijetí paketu. Mimo této posloupnosti budou výstupem další příznaky, tyto příznaky se budou týkat celého toku, nikoli jednotlivých paketů.

```
java -jar selenium.jar -role hub -host 192.168.1.106
java -Dwebdriver.chrome.driver=./chromedriver -jar selenium.jar -role node
↪  -hub http://192.168.1.106:4444/grid/register/ -browser browserName=chrome
```

Zdrojový kód 2.1: Ukázka spuštění Selenium Grid s jedním *nodem*, který pracuje s jedním webovým prohlížečem Chrome.

Protože webová stránka může být načtena pomocí více TCP spojení najednou [21], výstupem každé zaznamenané komunikace by měl být jeden nebo více toků.

Generátor má dva případy užití. V prvním případě je vstupem soubor, který obsahuje seznam URL webových stránek, které mají být navštíveny. Generátor je poté postupně navštěvuje, zaznamenává, analyzuje a ukládá síťový provoz a metadata. Druhý možný způsob použití je, že se generátoru zadá iniciální množina URL a pravidlo, jaký tvar mají mít navštěvované URL. Generátor pak sám prochází web, na webových stránkách automaticky nachází odkazy, které poté postupně navštěvuje. Přitom webovou komunikaci zaznamenává, takto zachycenou a zpracovanou komunikaci pak společně s metadaty ukládá.

2.2.2 Implementace

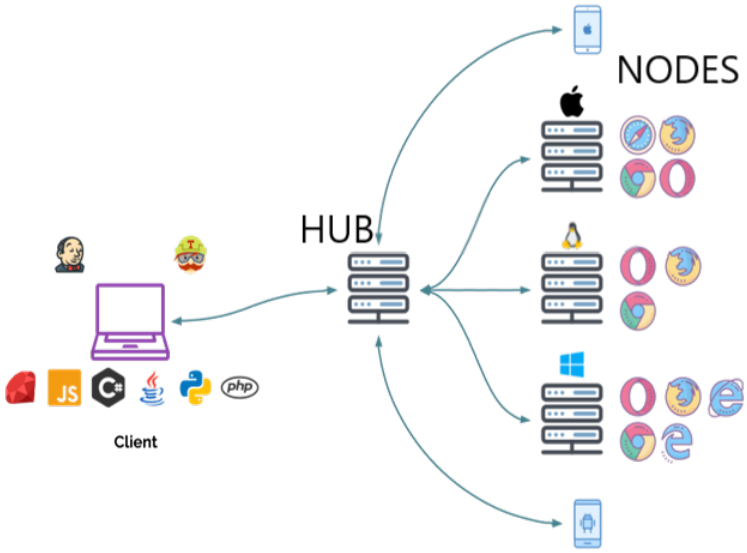
Pro sběr dat jsem vytvořil nástroj v programovacím jazyce Python. Tento nástroj postupně navštěvuje webové adresy a při každé návštěvě zaznamená a transformuje síťový provoz a společně s metadaty ho uloží.

2.2.2.1 Použité nástroje

Generátor je založen na nástroji Selenium[1]. Selenium je nástroj pro automatické testování webových aplikací, který umožňuje programově ovládat webový prohlížeč.

Generátor využívá takzvaný Selenium Grid, díky čemuž je možné, aby webový prohlížeč byl spuštěn na jiném než lokálním stroji. Selenium Grid se skládá z tzv. *hubu* a několika tzv. *nodů*. Jedná se o model master-slave, přičemž *hub* je v roli mastera, kontroluje a ovládá *nody*. Zda je Selenium spuštěno jako *hub* nebo *node*, se nastaví jednoduše pomocí přepínače `role`, ukázka je v kódu 2.1. Selenium Grid umožňuje paralelně provádět testy na více počítačích a centrálně spravovat různé verze prohlížečů a jejich konfigurace. Výhodou je, že na každém počítači může být jiný operační systém. Díky tomuto nástroji jsem mohl jednoduše splnit požadavky stanovené v kapitole 2.2.1. Ilustrace použití Selenium Grid, se čtyřmi *nody*, s různými operačními systémy, je na obrázku 2.2.

[1] `https://www.selenium.dev/`



Obrázek 2.2: Ilustrace použití Selenium Grid. Zdroj: [20].

V mém konkrétním případě předpokládám, jak již bylo navrženo v kapitole 2.2.1, že generátor datasetů a webové prohlížeče poběží na samostatných strojích. Jedna instance Selenium *node* umožňuje ovládat více prohlížečů, ale pak pokud je prováděna nějaká akce, například navštívení nějakého webu, tak všechny prohlížeče ovládané tímto *nodem* provádějí jednu a tu samou akci v jeden čas. To se ale v našem případě nehodí, protože by se nám kryly síťové přenosy dat. Proto budu mít vždy pouze jeden internetový prohlížeč na jeden *node* a spíše budu mít více *nodů*. Od této chvíle předpokládejme, že jeden *node* se rovná jednomu webovému prohlížeči.

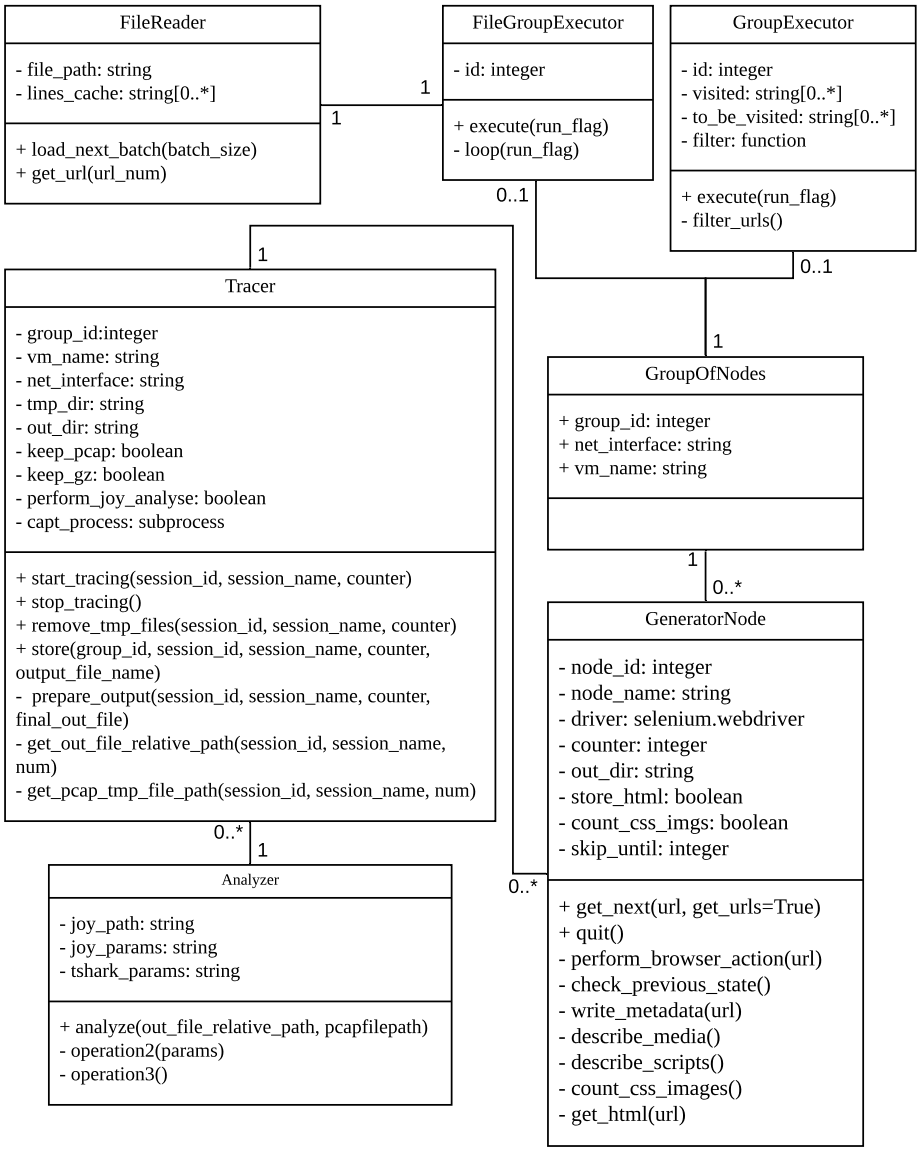
Program pro generování datasetů dále využívá nástroj TShark[2] pro zachycení komunikace jednotlivých *nodů*. TShark je analyzátor síťového provozu. Umožňuje zachytit data paketů ze živé síťové komunikace. Tento nástroj ukládá zachycenou komunikaci do souborů ve formátu PCAP [22].

Pro zpracování zachycené komunikace jsem použil program Cisco Joy[3]. Tento nástroj dovede zpracovat zaznamenané pakety, uložené v souboru ve formátu PCAP na jednotlivé toky paketů. Toky paketů, je pak možné uložit jako posloupnosti $n$tic příznaků, zmiňované v 2.2.1.

2.2.2.2 Objektový model

Tato sekce stručně popisuje z jakých tříd se generátor skládá a jaký je jejich význam. Vztahy mezi jednotlivými třídami jsou znázorněny na zjednodušeném

[2] `https://www.wireshark.org/docs/man-pages/tshark.html`
[3] `https://github.com/cisco/joy`



Obrázek 2.3: Diagram tříd programu pro generování datových sad.

diagramu tříd 2.3. Program se skládá z následujících entit.

Analyzer Tato třída slouží k analýze souborů PCAP pomocí nástroje Joy.

Tracer Objekty této třídy slouží ke sledování síťového provozu na síťovém rozhraní daném v parametru konstruktoru. Dvě stěžejní metody této třídy jsou metody `start_tracing` a `stop_tracing`. Metoda `start_tracing` slouží

k započetí zaznamenávání webového provozu. K ukončení záznamu provozu naopak slouží metoda `stop_tracing`. V době mezi voláním těchto metod je prováděn záznam pomocí nástroje TShark, který je spuštěn v samostatném procesu.

GeneratorNode Třída `GeneratorNode` reprezentuje instanci Selenia spuštěného jako *node*. Instance ovládá jeden konkrétní webový prohlížeč. Provádí webové dotazy a zároveň řídí objekt třídy Tracer tak, aby zachytil síťový provoz způsobený dotazem.

Třída `GeneratorNode` poskytuje také webové odkazy, které se vyskytují na navštívené webové stránce. Odkazy na stránce jsou nalezeny pomocí metody, kterou poskytuje Selenium. Metoda `find_elements_by_xpath` umožňuje vyhledávat elementy v HTML kódu webové stránky pomocí dotazovacího jazyka XPath. Pomocí této metody a dotazu `//a[@href]` jsou nalezeny všechny odkazy na webové stránce.

Dále tato třída poskytuje metody pro určení počtu obrázků na stránce a počtu javascriptových souborů. Určení počtu obrázků i skriptů na stránce probíhá tak, že za pomocí metody `find_elements_by_tag_name`, kterou nabízí Selenium, jsou nalezeny všechny tagy `img`, resp. `script` na stránce a ty jsou spočteny. Třída GeneratorNode umožňuje do počtu obrázků také započítávat obrázky použité v souborech kaskádových stylů CSS. Pro práci s kaskádovými styly Selenium bohužel neposkytuje funkce jako pro práci s HTML. Proto metoda pro získání počtu obrázků v CSS nejdříve v HTML kódu nalezne odkazy na CSS soubory, poté tyto soubory stáhne a v nich pomocí regulárního výrazu nalezne obrázky a ty jsou následně spočteny.

GroupOfNodes Účelem této třídy je sdružovat jednotlivé objekty třídy `GeneratorNode` do skupiny. Pro objekty v jedné skupině jsou prováděny akce sériově.

Toto je důležité, protože akce prováděné webovými prohlížeči na jednom stroji, respektive přes jedno síťové rozhraní, nesmí probíhat zároveň, jinak by zachycená komunikace nedávala smysl, protože by byly smíchány toky dat pro různé webové dotazy.

Je tedy potřeba, aby všechny objekty `GeneratorNode`, které ovládají webové prohlížeče na jednom stroji byly v jedné skupině. Všechny akce na tomto stroji jsou pak prováděny sériově.

Pokud je počítačů, na kterých jsou internetové prohlížeče ovládány tímto programem více, pak je také možné mít více skupin. Akce pak mohou být mezi skupinami prováděny paralelně a nezávisle.

GroupExecutor Tato třída slouží k procházení webového prostoru a provádění webových dotazů jednou skupinou *nodů*.

V konstruktoru objektu lze mimo jiné předat iniciální množinu webových adres a filtr adres. V metodě `execute` pak bude jedna URL z iniciální množiny navštívena všemi *nody* skupiny. Přitom bude, pro každý *node* skupiny, komunikace zaznamenána, transformována a uložena. Odkazy, které byly nalezeny na navštívené stránce a projdou filtrem, budou přidány do množiny. Poté je z množiny vybrána další URL a opět navštívena všemi *nody*. Tento proces se opakuje, dokud není z vnějšku zastaven. Je také zajištěno, že program při procházení webového prostoru neuvízne v cyklu, protože si pamatuje již navštívené webové adresy.

To znamená, že pokud například jako iniciální množinu předáme množinu o jedné URL a to `https://cs.wikipedia.org/` a jako filtr funkci, která ponechá pouze URL, které jsou ve tvaru `https://cs.wikipedia.org/wiki/*`, pak bude tato skupina objektů třídy `GeneratorNode` procházet českou variantu Wikipedie.

FileGroupExecutor Třída `FileGroupExecutor` slouží ke stejnému účelu jako `GroupExecutor`. Rozdíl je ale v tom, že místo toho, aby objekt této třídy automaticky procházel webový prostor, používá vstupní soubor se seznamem webových adres k navštívení.

Vstupní soubor je procházen postupně řádek po řádku a čtené adresy jsou navštěvovány skupinou *nodů*. Soubor je možné zpracovávat po dávkách a každou dávku lze $n$krát opakovat. Je také možné opakovat návštěvu každé URL $m$krát po sobě.

FileReader Toto je pomocná třída používaná ke čtení vstupního souboru se seznamem URL po dávkách. Je používaná objekty třídy `FileGroupExecutor`.

2.3 Tvorba datových sad

Implementoval jsem spustitelný skript, který využívá výše zmíněné třídy. Na jeho začátku jsou definovány skupiny uzlů a také samotné uzly. Dále je zde definována iniciální množina webových adres a funkce, která slouží jako filtr adres. Dále následuje funkce `main`, která pro každou definovanou skupinu vytvoří objekt třídy `GroupExecutor` nebo `FileGroupExecutor`, podle toho zda byla předána cesta k souboru se seznamem webových adres. Vytvořené *executory* jsou následně každý ve svém vlastním vlákně spuštěny a tím je zahájeno generování datové sady.

V mém případě jsem pro sběr dat použil Oracle VirtualBox[4], ale lze použít i jiný virtualizační nástroj nebo použít fyzické počítače. Na lokálním stroji jsem spustil Selenium hub. Ve VirtualBoxu pak několik virtuálních strojů, v každém takovém virtuálním stroji jsem pak spustil Selenium node. Každý

[4] `https://www.virtualbox.org/`

virtuální stroj může mít různý operační systém a mohou na něm být provozovány různé webové prohlížeče.

Virtuální stroje jsou provozovány v síťovém režimu Host-only [23]. Je-li použit Host-only síťový režim, vytvoří Oracle VirtualBox nové softwarové síťové rozhraní na hostiteli, které se poté zobrazí vedle stávajícího síťového rozhraní. Pro každý virtuální stroj je potřeba vytvořit nové virtuální síťové rozhraní. Generátor poté zachytává komunikaci na těchto virtuálních síťových rozhraních, takže na každém takovém rozhraní je zachycena pouze komunikace jednoho virtuálního stroje a nikoli například komunikace ostatních virtuálních strojů, nebo komunikace hostitele. Aby virtuální stroje měly přístup do internetu, je třeba na hostovském stroji nastavit směrování mezi virtuálním rozhraním a lokálním rozhraním, které do internetu přístup má. Pro usnadnění nastavení směrování v operačním systému Linux jsem připravil bashový skript.

Nastavení prostředí a použití mnou implementovaného generátoru datových sad je detailně popsané v souboru `readme`, který je součástí zdrojových kódů generátoru.

2.4 Získaná data

2.4.1 Vytvořené datové sady

Část dat jsem nasbíral sám. Největší část dat, ale byla nasbírána s pomocí členů Laboratoře monitorování síťového provozu[5], díky kterým bylo možné zachytit rozsáhlý dataset.

Datasety jsme generovali pomocí vstupního souboru se seznamem webových adres k navštívení. Použili jsme textový soubor, který obsahuje více než 2,3 milionu adres webových stránek Wikipedie.

Pro první dataset, který jsme nasbírali, jsme použili velikost dávky 500 adres, opakování každé dávky třikrát a opakované navštívení jedné webové stránky třikrát po sobě. Pro představu vizte obrázek 2.4, kde je naznačeno opakování navštívení webových adres ze vstupního souboru. První dataset vznikl provedením 32 913 načtení webových stránek, bylo navštíveno 915 unikátních url, celkem to dělá 5,2 GB dat. Data byla nasbírána pomocí dvou virtuálních strojů, na jednom s operačním systémem Windows a na druhém s operačním systémem Linux, konkrétně s distribucí Ubuntu. Na každém z nich s pomocí webových prohlížečů Mozilla Firefox a Google Chrome. Oba stroje pracovali na sobě nezávisle, paralelně, ale prohlížeče v rámci jednoho stroje navštěvovali webové stránky, sériově, tak aby nedošlo k smíchání komunikací na jednom síťovém rozhraní. Z popisu výše vyplývá, že většina URL (některé méněkrát, protože záleží na okamžiku ukončení běhu generátoru) je v datasetu 36krát, protože platí následující vztah

[5] `https://netmon.fit.cvut.cz/`

	Dávka 0
0	url 0
1	url 0
2	url 0
3	url 1
4	url 1
5	url 1
...	...
	Dávka 0
500	url 0
501	url 0
502	url 0
503	url 1
...	...
	Dávka 0
...	...
	Dávka 1
1500	url 500
1501	url 500
1502	url 500
1503	url 501
...	...
	Dávka 1
...	...

Obrázek 2.4: Tento obrázek slouží pro ilustraci opakování navštívení URL. Číslo v levém sloupci je pořadí návštěvy webové stránky. Druhý sloupec označuje číslo url ve vstupním souboru se seznamem webových adres.

$$U * D * M * B = 36,$$

kde

- U je počet opakování jedné URL v řadě: 3,
- D je počet opakování dávky: 3,
- M je počet strojů (skupin *nodů*): 2 a
- B je počet webových prohlížečů (*nodů*) na jednom stroji: 2.

```
fp = webdriver.FirefoxProfile()
fp.DEFAULT_PREFERENCES['frozen']["browser.cache.disk.enable"]
↪  = False
fp.DEFAULT_PREFERENCES['frozen']["browser.cache.memory.enable"]
↪  = False
fo = webdriver.FirefoxOptions()
fo.profile = fp
```

Zdrojový kód 2.2: Ukázka vypnutí cache internetového prohlížeče Mozilla Firefox.

Později jsme vytvořili i druhý dataset. Tento dataset jsme vytvářeli stejným způsobem, jako ten předchozí. Jediná změna oproti prvnímu datasetu je, že webové prohlížeče měly vypnutou cache při navštěvování stránek.

Cache prohlížeče Firefox se mi podařilo vypnout pomocí parametrů předaných webdriveru tak, jak je ukázáno v ukázce kódu 2.2.

V případě prohlížeče Google Chrome jsem se pokoušel vypnout použití cache podobným způsobem jako u Firefoxu. Přesto, že jsem vyzkoušel mnoho různých nastavení (ukázka některých je ve zdrojovém kódu 2.3, tato nastavení deaktivují cache jen částečně), nikdy se mi nepodařilo cache vypnout úplně. Ať už jsem použil jakákoli nastavení, v „Developer tools“ prohlížeče Chrome, v záložce „Network“ bylo stále vidět, že i přesto, že se některé druhy cache deaktivovat podařilo tak, že jiné druhy prohlížeč používá i nadále.

Pokoušel jsem se také používání cache potlačit pomocí doplňku doinstalovaného do prohlížeče, který před každým webovým dotazem cache vymaže. Doplněk, ale nebylo jednoduché skloubit s použitím Selenia. Doplněk se musel při každém spuštění doinstalovávat do jednotlivých prohlížečů a to dělalo problémy, a proto jsem toto řešení zavrhl.

Cache webového prohlížeče Google Chrome se mi podařilo kompletně vypnout pouze ručně a to přes „Developer tools“, záložku „Network“ a zde zaškrtnutím „disable cache“. Aby používání cache zůstalo vypnuté, je nutné nechat okno nástroje „Developer tools“ otevřené. Když jsme tedy sbírali druhý dataset, bylo nutné po nastartování prohlížeče Chrome na virtuálních strojích vypnout cache ručně. Je sice nepříjemné, že nelze cache vypnout programově, ale alespoň stačí tento ruční zásah provést jen při spuštění prohlížeče, který je automaticky nastartován při startu generátoru datových sad a použití cache pak již zůstane vypnuté po celou dobu běhu generátoru.

Druhý dataset vznikl provedením 31 090 načtení webových stránek a bylo stejně jako v případě prvního datasetu navštíveno 915 unikátních URL. Proč jsme vytvářeli tento dataset bude zdůvodněno v následující kapitole.

2.4.2 Podoba vygenerovaných dat

Výstupem generátoru jsou následující data.

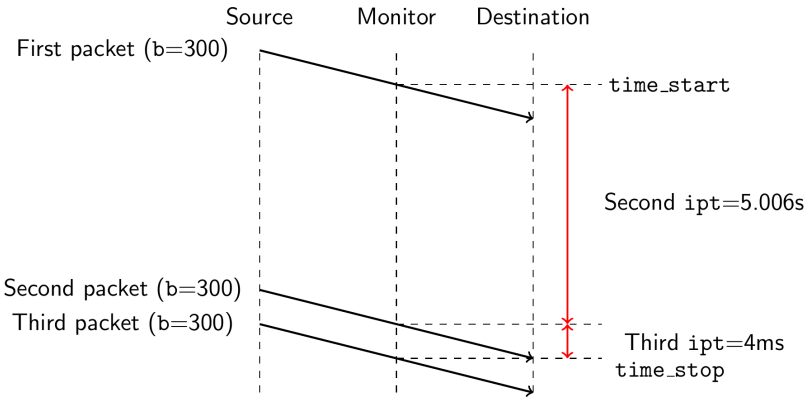
- Pro každý *node* je vygenerován soubor s metadaty. Soubor ve svém

```
cho = webdriver.ChromeOptions()
cho.add_argument("--disk-cache-dir=/dev/null")
cho.add_argument("--disk-cache-size=1")
cho.add_argument("--media-cache-dir=/dev/null")
cho.add_argument("--media-cache-size=1")
cho.add_argument("--disable-application-cache")
cho.add_argument("--aggressive-cache-discard")
cho.add_argument("--enable-aggressive-domstorage-flushing")
```

Zdrojový kód 2.3: Ukázka pokusu o vypnutí cache internetového prohlížeče Google Chrome. Pomocí tohoto se podařilo vypnout „media cache“, „prefetch cache“ a „disk cache“ nikoli však „memory cache“. Toto jsem vypozoroval, včetně názvů typů cache, z nástroje „Developer tools“, který je součástí prohlížeče Google Chrome.

```
{
  "sa": "0.0.0.0",                    // IP source address
  "da": "255.255.255.255",            // IP destination address
  "pr": 17,                           // IP protocol number (17 = UDP)
  "sp": 68,                           // UDP source port
  "dp": 67,                           // UDP destination port
  "bytes_out": 900,                   // bytes sent from sa to da
  "num_pkts_out": 3,                  // packets sent from sa to da
  "time_start": 1479227824.653818,    // start time in seconds since the epoch
  "time_end": 1479227829.665744,      // end time in seconds since the epoch
  "packets": [                        // array of packet information
    {
      "b": 300,                       // bytes in UDP Data field
      "dir": ">",                     // direction: sa -> da
      "ipt": 0                        // inter-packet time: 0 ms since time_start
    },
    {
      "b": 300,                       // bytes in UDP Data field
      "dir": ">",                     // direction: sa -> da
      "ipt": 5006                     // inter-packet time: 5006 ms since last packet
    },
    {
      "b": 300,                       // bytes in UDP Data field
      "dir": ">",                     // direction: sa -> da
      "ipt": 4                        // inter-packet time: 4 ms since last packet
    }
  ],
  "ip": {
    "out": {
      "ttl": 128,
      "id": [
        1,
        2,
        3
      ]
    }
  },
  "expire_type": "i"
}
```

Obrázek 2.5: Ukázka toku s popisem jednotlivých položek. Zdroj: [6]



Obrázek 2.6: Ilustrace toku paketů. Tento obrázek ilustruje tok popsaný v obrázku 2.5. Zdroj: [6]

názvu obsahuje ID skupiny *nodů* a také ID *nodu*. V souboru jsou na každém řádku metadata pro jeden webový dotaz. Ukázka souboru s metadaty je v příloze B.1.

- Ke každému řádku v souboru metadat patří příslušný soubor se záznamem komunikace ve formě toků příznaků neboli „flow“. Tento soubor obsahuje toky, které vznikly záznamem webové komunikace při návštěvě jedné určité URL. Toky v tomto souboru jsou výstupem nástroje Joy. Popis příznaků, které takový tok obsahuje, je na obrázku 2.5. Znázornění toho stejného toku je na obrázku 2.6. Každý záznam komunikace může obsahovat vícero toků. Ve výstupním souboru, je jeden tok na řádek a stejně tak, jako každý řádek v souboru metadat, je i zde každý řádek ve formátu JSON[6]. Ukázku výstupního souboru se záznamem komunikace můžete vidět v příloze B.2.

- Generátor umožňuje ukládat i HTML kód navštívených stránek. To se může hodit při další práci s vytvořeným datasetem.

- Program nabízí možnost uchovat i některé druhy dočasných souborů, které používá během tvorby datové sady. Jedním takovým druhem jsou soubory se zachycenými pakety nástrojem TShark ve formátu PCAP.

- Dalším druhem dočasných souborů, které lze uchovat jsou komprimované výstupní soubory z nástroje Joy.

[6] https://www.json.org

```
1  "packets":[{"b":110,"dir":">","ipt":0}, {"b":30,"dir":"<","ipt":18},
   ↪  {"b":117,"dir":"<","ipt":0}, {"b":292,"dir":">","ipt":42},
   ↪  {"b":113,"dir":">","ipt":0}, {"b":110,"dir":">","ipt":0},
   ↪  {"b":30,"dir":"<","ipt":17}, {"b":119,"dir":"<","ipt":0},
   ↪  {"b":30,"dir":"<","ipt":0}, {"b":119,"dir":"<","ipt":0},
   ↪  {"b":157,"dir":"<","ipt":17}]
2
3  [[110, -30, -117, 292, 113, 110, -30, -119, -30, -119, -157]]
```

Zdrojový kód 2.4: Ukázka transformace toku. První řádek obsahuje tok ze souboru, který byl vygenerován výše diskutovaným programem pro tvorbu datových sad. Třetí řádek obsahuje tok po provedené transformaci.

Které soubory má generátor uchovávat, kam má výstupní soubory zapisovat, parametry nástrojů TShark, Joy a další, lze nastavit v konfiguračním souboru `config.ini`, který je součástí generátoru datových sad.

2.4.3 Transformace vygenerovaných dat

Jelikož výstupní data z generátoru nejsou úplně vhodná pro přímé zpracování, vytvořil jsem v jazyku Python skript pro transformaci nasbíraných dat. Tento nástroj lze používat jak z jiného Python programu, tak z příkazové řádky. Umožňuje transformovat vstupní soubory se záznamy komunikací jedna ku jedné do výstupních souborů, ale i transformovat všechny do jednoho výstupního souboru.

Pro převod záznamů síťových toků je zde připravena abstraktní třída v souboru `converter.py`, díky tomu lze jednoduše změnit způsob transformace síťových toků. Pro transformaci toků jsem vytvořil jednoduchou implementaci této abstraktní třídy, která tok transformuje na pole celých čísel. Tato čísla vyjadřují počet bajtů, znaménko pak směr toku. Jak taková transformace toku vypadá, lze vidět v ukázce 2.4. Stejnou transformaci provede skript i pro mezipaketové časy, v ukázce toku na obrázku 2.5 to jsou položky s označením `ipt`.

Pokud se tedy použije varianta transformace záznamů komunikací do jednoho výstupního souboru, pak výstup vypadá následovně, vizte ukázku kódu C.1 v příloze C. Výstupní souboru je ve formátu CSV[7], který na každém řádku obsahuje data pro jednu komunikaci. Na pozicích v řádku, kde se sloupec netýká celé komunikace, ale týká se jednotlivých toků jsou hodnoty sdruženy v seznamu hodnot. Výstupní soubor na prvním řádku obsahuje hlavičku s názvy sloupců.

[7] `https://en.wikipedia.org/wiki/Comma-separated_values`

Kapitola 3

Identifikace webového provozu a odhad struktury webového obsahu

3.1 Návrh

Tato sekce popisuje hrubý návrh kroků potřebných ke splnění stanovených cílů.

V kapitole 1.4 bylo stanoveno, že identifikace webového provozu a odhad struktury obsahu webové stránky, bude prováděn pomocí příznakových vektorů. Prvním krokem následujícím po úvodní analýze a předzpracování dat bude tvorba nových příznaků. Tvorbě nových příznaků se věnuje sekce 3.3.2.

Poté rozdělím datovou sadu na webové stránky, ze kterých budu jednu část považovat za stránky klasifikátorem viděné a druhou část budu považovat za stránky klasifikátorem neviděné. To proto, abych mohl provádět experimenty s rozpoznáním známých a neznámých webových stránek a ověřit úspěšnost odhadu obsahu webových stránek, pro webové stránky pro model jak známé, tak i neznámé.

Viděnou část datasetu následně rozdělím na trénovací, validační a testovací podmnožinu. Abych mohl ladit model strojového učení a jeho hyperparametry.

Jelikož nelze dopředu určit, které příznaky jsou pro jednotlivé úlohy vhodné a které nikoliv, tak v dalším kroku bude proveden výběr vhodné podmnožiny příznaků. Díky tomuto kroku se sníží dimenzionalita dat.

Poté již bude následovat tvorba samotných modelů, jejich trénování a následné vyhodnocení jejich úspěšnosti. Hrubým návrhem tvorby modelů pro identifikaci webových stránek a odhad struktury webových stránek se budou zabývat následující dvě podkapitoly 3.1.1 a 3.1.2.

3.1.1 Identifikace webových stránek

Pro klasifikaci webových stránek zkusím použít více různých klasifikátorů, protože dopředu lze jen velmi těžko odhadnout, jaký bude fungovat dobře. Protože úloha identifikace webových stránek je úlohou klasifikace do vícero tříd a v tomto konkrétním případě dokonce do velmi mnoha tříd, rozhodl jsem se použít pouze takové klasifikátory, které ze své podstaty umožňují klasifikovat do více tříd [27] a nepoužít například binární klasifikátor postupem jeden versus zbytek [16]. Zvolil jsem následující klasifikátory:

- Random Forest,
- Extra Trees Ensemble,
- KNN,
- Multi-layer Perceptron,
- AdaBoost,
- Linear Discriminant Analysis a
- Logistic Regression.

Pokusil jsem se vybrat takové klasifikátory, aby jejich principy byly různé, aby byla experimentálně ověřena výkonnost pestrého výběru klasifikátorů a díky tomu bylo možné získat představu o vhodnosti použití různých druhů klasifikátorů pro úlohu identifikace webových stránek. Více informací o zvolených klasifikátorech lze nalézt v [27] a [17].

Jako metriku úspěšnosti klasifikátoru, použiji *accuracy*, jak při optimalizaci hyperparametrů, tak i při vyhodnocení úspěšnosti klasifikátoru. Tato metrika je vypočítána podle následujícího vzorce, kde y je skutečná hodnota, $\hat{y}$ je predikovaná hodnota a n_{samples} je počet všech vzorků:

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i). \tag{3.1}$$

Hodnota *accuracy* se pohybuje na intervalu $\langle 0, 1 \rangle$, pokud se předpovězené hodnoty pro celou testovací sadu rovnají skutečným hodnotám, pak je *accuracy* rovna jedné, v opačném případě kdy se žádná předpovězená hodnota nerovná skutečné hodnotě, tak je *accuracy* rovna nule.

3.1.2 Odhad struktury webového obsahu

Tak jako jsem v případě identifikace webových stránek navrhl použití více různých klasifikátorů, použiji v případě odhadu struktury webového obsahu více různých regresních metod. Zvolil jsem následující regresory:

- Linear Regression,
- Ridge Regression,
- SGD Regressor,
- ElasticNet Regressor,
- Random Forest Regressor,
- Extra Trees Regressor a
- AdaBoost Regressor.

Podobně jako v případě klasifikátorů, jsem se pokusil zvolit takové regresory, aby byla experimentálně ověřena výkonnost různých regresorů a díky tomu bylo možné získat představu o vhodnosti jejich použití pro úlohy odhadu struktury webových stránek, tj. odhadu počtu skriptů a odhadu počtu obrázků na webové stránce. Více informací o zvolených regresorech lze nalézt v [28].

Jako metriku regresního odhadu pro ladění hyperparametrů a výběr modelu použiji *Root mean square error* (RMSE). Tato metrika se vypočte pomocí následujícího výrazu

$$\texttt{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2}. \tag{3.2}$$

Díky tomu, že jsou chyby (rozdíly $y_i - \hat{y}_i$) umocněny na druhou ještě před tím než jsou zprůměrovány, tak RMSE dává velkým chybám velkou váhu. RMSE tedy zdůrazňuje velké chyby, čím je chyba větší, tím zdůrazněna více. Tato vlastnost může komplikovat interpretaci výsledků. Proto jsem se rozhodl pro vyhodnocení kvality regresních modelů použít druhou metriku.

Pro lepší interpretovatelnost výsledků tedy použiji také *Mean absolute error* (MAE). MAE na rozdíl od RMSE počítá průměrnou absolutní hodnotu chyby, tedy nejdříve je vypočtena absolutní hodnota rozdílu $y_i - \hat{y}_i$ a teprve poté je vypočten průměr, velkým i malým chybám je v tomto případě dávána stejná váha. Tato metrika je určena vzorcem

$$\texttt{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|. \tag{3.3}$$

MAE i RMSE vyjadřují průměrnou chybu predikce modelu v jednotkách sledované proměnné. Obě metriky se mohou pohybovat od 0 do ∞. Jsou to negativně orientované metriky, což znamená, že nižší hodnoty jsou lepší.

3.1.3 Zvolené nástroje

V této kapitole se zabývám nejdůležitejšími nástroji, které jsem zvolil pro zpracování datové sady, tvorbu modelů a provedení experimentů na vzniklých modelech.

Python Jako programovací jazyk jsem zvolil Python [24]. Pokud nebudeme brát v úvahu jazyky, jako je dotazovací jazyk SQL, pak jsou podle [15] dvěma nejpoužívanějšími jazyky v datové vědě Python a R, přičemž Python je v dnešní době ještě častěji používaný než R. Vybral jsem Python, protože s ním mám daleko větší zkušenosti než s R.

Jupyter notebook Jako prostředí jsem zvolil v dnešní době v oblasti strojového učení velmi oblíbené Jupyter notebooky [25].

Pandas Pro práci s daty jsem zvolil knihovnu Pandas [43]. Tato knihovna je dnes v podstatě standardem v oblasti zpracování a analýzy dat v jazyku Python.

Scikit-learn Scikit-learn [28] je opensource knihovna pro strojové učení v jazyce Python. Obsahuje nástroje pro předzpracování dat, redukci dimensionality, selekci modelu, shlukování, regresi, klasifikaci a další.

Matplotlib Pro vizualizaci dat jsem využil dvě knihovny. První z nich je knihovna Matplotlib [44].

Seaborn Druhou knihovnou pro vizualizaci dat je knihovna Seaborn [45]. Tato knihovna je založena na knihovně Matplotlib, zjednodušuje vytváření vizualizací, hodí se pro práci se statistickými grafy.

3.2 Analýza datové sady

Nejprve jsem začal zpracovávat první dataset zmiňovaný v podkapitole 2.4.1, tedy datovou sadu s povolenou cache webových prohlížečů. Tento dataset jsem nejdříve předzpracoval skriptem `converter.py` popisovaným v sekci 2.4.3 tak, abych dostal jeden CSV soubor. Tento soubor jsem jednoduše načetl pomocí knihovny Pandas do objektu `df` typu Pandas Dataframe. Dataframe je dvojrozměrná tabulková datová struktura s označenými osami (řádky a sloupce), která umožňuje snadno s daty manipulovat, dotazovat se na ně a provádět nad nimi všemožné operace. Dataframe `df` má na každém řádku jednu návštěvu webové stránky, sloupce pak udávají jednotlivé příznaky a metadata. Objekt `df` s načteným datasetem pak má konkrétně sloupce:

- `group_id` – metadata, tento sloupec udává číslo skupiny *nodů*,

- `session_id` – metadata, číslo *nodu*, pomocí nějž byla webová stránka navštívena,
- `os` – metadata, název operačního systému,
- `browser` – metadata, název webového prohlížeče, kterým byla stránka navštívena,
- `query_num` – metadata, číslo webového dotazu v rámci jednoho *nodu*,
- `url` – metadata, webová adresa navštívené stránky,
- `num_images` – metadata, počet obrázků na stránce,
- `num_scripts` – metadata, počet skriptů,
- `bytes_out` – počet odeslaných bajtů v každém toku (pole celých čísel o dél-ce počtu toků),
- `num_pkts_out` počet odeslaných paketů v každém toku (pole celých čísel o délce počtu toků),
- `bytes_in` – stejné jako `bytes_out`, ale v tomto případě obsahuje počet přijatých bajtů,
- `num_pkts_in` – stejné jako `num_pkts_out`, ale tento sloupec obsahuje počet přijatých paketů, nikoliv odeslaných,
- `duration` – pro každý tok v komunikaci udává celkovou dobu trvání toku v sekundách,
- `traffic` – tento sloupec obsahuje webovou komunikaci transformovanou pomocí skriptu v kapitole 2.4.3, obsahuje tedy seznam seznamů celých čísel, kde každý vnořený seznam představuje jeden tok a čísla v něm velikosti paketů v bajtech, s kladným či záporným znaménkem podle směru toku paketu,
- `inter_pkt_times` – obsah tohoto sloupce je analogický se sloupcem předchozím, jen místo velikostí paketů obsahuje mezipaketové časy. Mezipaketový čas, tak jak již bylo vysvětleno v obrázku 2.5, pro každý paket udává dobu v milisekundách od předchozího paketu v daném toku.

Ve zbytku této práce, budu někdy pro jednoduchost označovat velikost paketu s informací o směru toku paketu, udanou znaménkem a mezipaketový čas s informací o směru toku paketu, udanou znaménkem jako orientovanou velikost paketu, respektive orientovaný mezipaketový čas.

Hrubý přehled informací o načtené datové sadě je ve výpisu 3.1. V tomto výpisu jsem si všiml, že jsou zde tři různé názvy webového prohlížeče Mozilla Firefox. Po bližším zkoumaní jsem zjistil, že při generovaní dat na instanci

```
Number of records in the dataset: 32912
Number of unique urls in the dataset: 915
Operating systems used to cerate the dataset:['linux' 'windows']
Browsers used to cerate the dataset:['firefox' 'chrome' 'fire']
chrome     16467
firefox     8224
fire        8221
```

Zdrojový kód 3.1: Výpis základního přehledu informací o datové sadě. V tomto případě s nekonzistencí v identifikátorech webových prohlížečů.

```
Number of records in the dataset: 32912
Number of unique urls in the dataset: 915
Operating systems used to cerate the dataset:['linux' 'windows']
Web browsers used to cerate the dataset:['firefox' 'chrome']
chrome     16467
firefox    16445
```

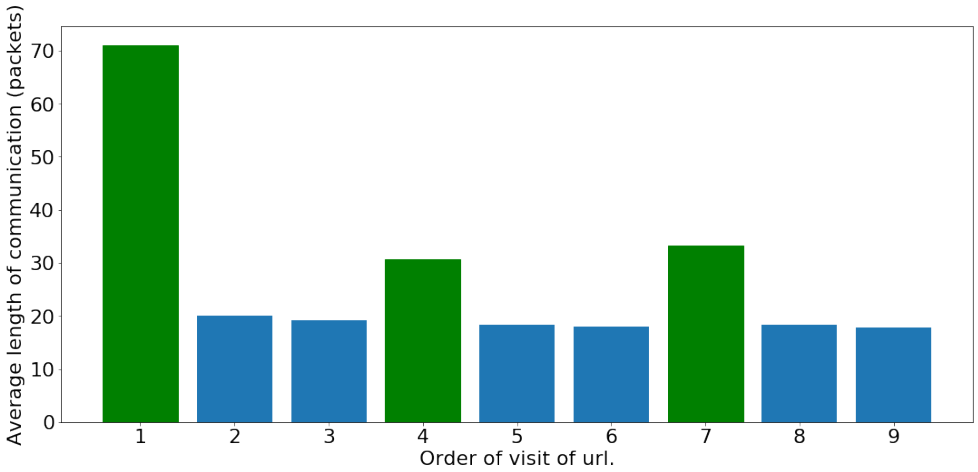
Zdrojový kód 3.2: Výpis základního přehledu informací o datové sadě.

s operačním systémem Windows, se ukládal jako název prohlížeče Firefox jen název „fire“. Chyba byla v definici *nodu*, kde jsem jako název zadal pouze „fire“. Nesprávný název webového prohlížeče jsem snadno v dataframu opravil. Stejný výpis po opravě vypadal tak, jak je vidět v kódu 3.2.

Již v této fázi jsem k `df` připojil několik nových sloupců, které ulehčily další práci s datovou sadou. Novými sloupci jsou:

- `num_flows`, který udává počet toků v komunikaci,
- `total_packets_1`, který udává celkový počet paketů v komunikaci, získaný z `traffic`,
- `total_packets_2` je celkový počet paketů v komunikaci, získaný jako součet polí `num_pkts_out` a `num_pkts_in`,
- `total_bytes_1`, který udává celkový počet bajtů v komunikaci, získaný z `traffic`,
- `total_bytes_2`, který udává celkový počet bajtů v komunikaci, získaný jako součet polí `bytes_out` a `bytes_in`,
- `order` vyjadřuje pořadí navštívení v rámci jedné webové adresy.

Jak je vidět z předchozího výčtu, pro počet bajtů a počet paketů přenesených v komunikaci jsem vytvořil dvě různé reprezentace. Součty se totiž pro každou variantu liší, nebyl jsem si jistý, která varianta je pro tvorbu modelů vhodnější. Ponechal jsem tedy obě varianty reprezentací a výběr nechal na později použité metodě výběru příznaků.



Obrázek 3.1: Demonstrace vlivu cache na délku komunikace udanou v počtu přenesených paketů.

Poté jsem dále zkoumal datovou sadu a provedl několik operací předzpracování dat. Jednou z otázek, kterou jsem si kladl bylo, jaký vliv má na zachycená data cache webových prohlížečů.

Zjistil jsem, že vliv cache na webovou komunikaci je značný. To lze vidět na grafu 3.1. Graf má na ose x pořadí navštívení webové stránky a na ose y je průměrná délka komunikace udaná v počtu přenesených paketů, vypočtená přes všechny webové adresy v datasetu. Z grafů lze vidět, že některá pořadí návštěvy stránky mají výrazně delší komunikaci než pořadí jiná. Toto chování téměř jistě způsobuje cache webového prohlížeče. Dále je vidět, že průměrně nejdelší komunikaci má první návštěva. To dává smysl, protože při první návštěvě webové stránky je cache prázdná či neobsahuje relevantní informace, a proto prohlížeč musí webovou stránku stáhnout kompletně. Také si lze všimnout, že každá třetí návštěva webové adresy, tj. sloupce zvýrazněné zelenou barvou, má vyšší průměrnou délku komunikace. To souvisí s tím, jak byly webové stránky navštěvovány. Jak již bylo řečeno, při sběru dat byly zvoleny parametry, velikost dávky 500, počet opakování url v dávce 3 a opakování dávky třikrát. Každá stránka tedy byla navštívena třikrát po sobě a poté znovu až při opakování dávky. Každý třetí sloupec tedy byl prvním navštívením webové stránky v řadě návštěv, které byly provedeny bezprostředně v řade za sebou. A jak je vidět, tak některé informace již v cache nebyly a proto musely být znovu staženy, ale také je vidět, že návštěvy 4 a 7 měli komunikace kratší než sloupec první, lze tedy usuzovat, ze určitá část dat oproti první návštěvě v cache zůstala.

Na obrázku 3.2 jsou grafy průběhů komunikací. Každý z grafů představuje jedno načtení stránky `https://en.wikipedia.org/wiki/American_Samoa`, na jedné instanci s operačním systémem Linux a webovým prohlížečem Google

```
Number of records in the dataset: 31090
Number of unique urls in the dataset: 915
Operating systems used to cerate the dataset:['linux' 'windows']
Web browsers used to cerate the dataset:['firefox' 'chrome']
chrome     15547
firefox    15543
```

Zdrojový kód 3.3: Výpis základního přehledu informací o datové sadě s deaktivovanou pamětí cache prohlížeče.

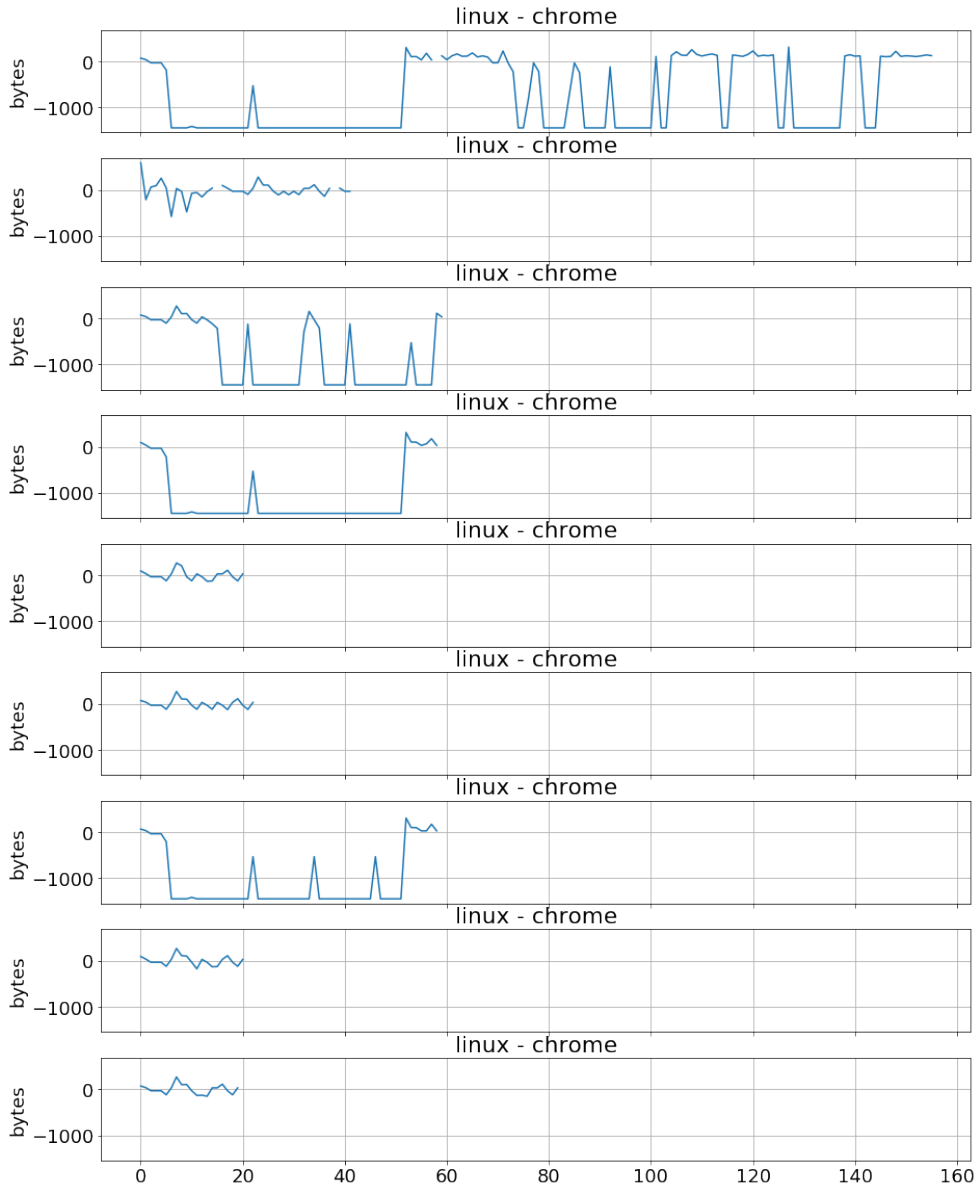
Chrome. Grafy jsou seřazeny od shora dolů podle pořadí návštěvy stránky. Každý z grafů má na ose y velikost paketu a na ose x samotné pakety, tyto diskrétní hodnoty jsou propojeny čarou, místo toho abych zobrazoval jen body, grafy jsou pak v tomto případě dle mého názoru lépe čitelné. Každý graf obsahuje jednu komunikaci, jelikož se jedna komunikace obvykle skládá z více toků, tak jsou pakety jednotlivých toků v grafu zřetězeny za sebou a odděleny prázdnou hodnotou (mezerou). Jak lze vidět, tak první komunikace je výrazně delší, než všechny ostatní. První komunikace obsahovala téměř sto paketů, kdežto většina ostatních méně než dvacet. Z grafů je velmi dobře vidět, jak velký vliv má cache webového prohlížeče na průběh načtení stránky. Průběh prvního načtení je opravdu velmi odlišný od načtení pozdějších. To značně komplikuje identifikaci webové stránky či odhad jejího obsahu.

Jelikož je vliv cache webových prohlížečů na přenos webového obsahu výrazný, rozhodl jsem se, omezit se na podproblém a to tím, že budu předpokládat vypnutou cache. Proto jsme vytvořili druhou datovou sadu, kterou jsem zmiňoval v kapitole 2.4.1.

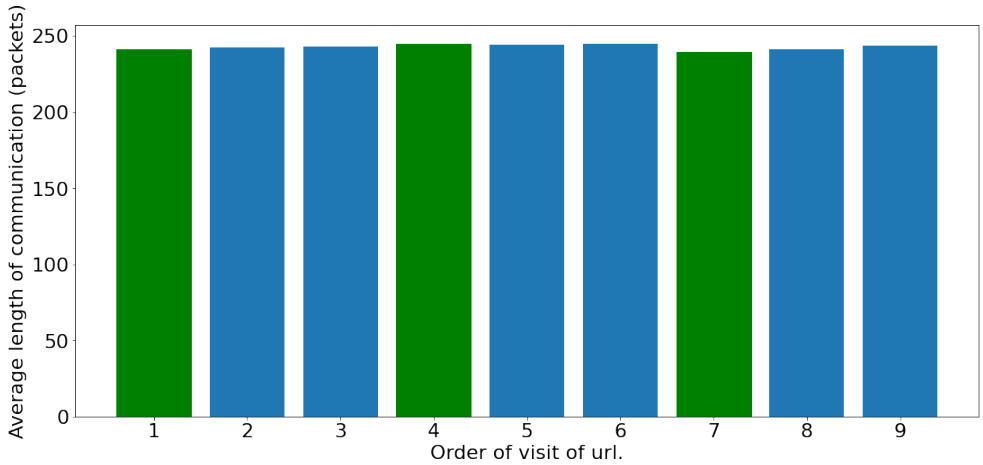
Následující text se již bude týkat pouze datasetu s vypnutou cache. Stejně tak jako v případě první datové sady, jsem dataset transformoval do jednoho CSV souboru, ten jsem načetl jako Pandas Dataframe. Také jsem opravil chybný identifikátor internetového prohlížeče Mozilla Firefox. A vytvořil jsem nové sloupce jako výše. Hrubý přehled informací o druhém datasetu je ve výpisu 3.3.

Graf 3.3 stejně jako graf 3.1 zobrazuje závislost průměrné délky komunikace na pořadí návštěvy URL. Jak lze vidět, v tomto případě jsou rozdíly mezi průměrnými délkami komunikací pro jednotlivá pořadí návštěv minimální. Trend popisovaný výše, kde měla každá třetí návštěva stránky výrazně delší průměrnou komunikaci, který byl způsobován vlivem cache, zde v tomto případě vůbec není.

Stejně tak jako v případě grafů na obrázku 3.2 zobrazují grafy na obrázku 3.4 načtení stránky `https://en.wikipedia.org/wiki/American_Samoa`, na jedné instanci s operačním systémem Linux a webovým prohlížečem Google Chrome, ovšem v tomto případě s deaktivovanou cache webového prohlížeče. V tomto případě lze vidět, že na rozdíl od předchozího případu, jsou si jednotlivé průběhy velmi podobné a průběhy další v pořadí se výrazně neliší od



Obrázek 3.2: Ukázka několika různých návštěv jedné webové stránky.



Obrázek 3.3: Průměrné délky komunikací pro jednotlivá pořadí návštěv s deaktivovaným použitím cache prohlížeče.

Počet skriptů na webové stránce	Počet výskytů
6	29167
4	1920
8	1
7	1

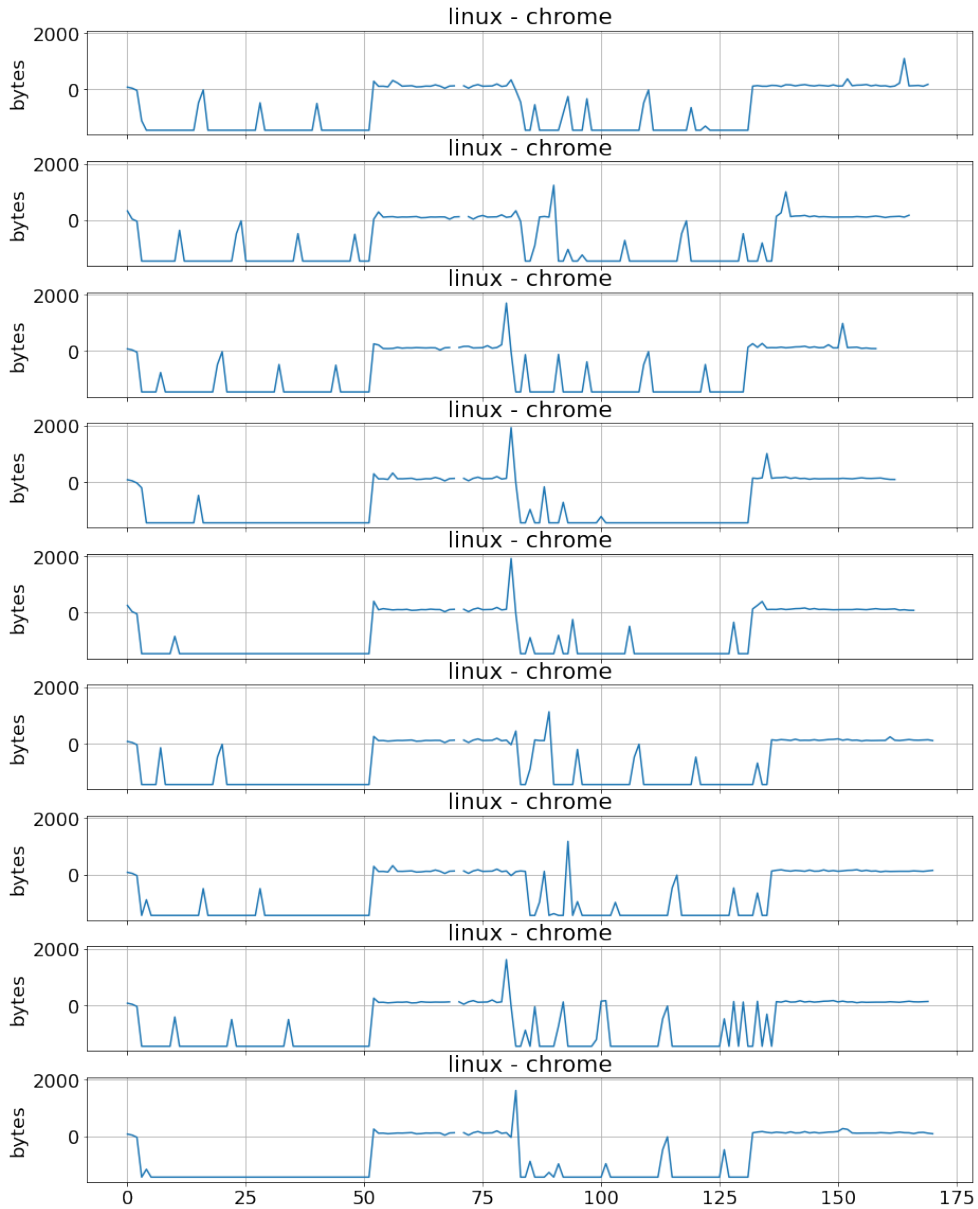
Tabulka 3.1: Distribuce počtu skriptů na stránce.

průběhu prvního. Lze tedy předpokládat že úlohy identifikace webové stránky a odhadu obsahu webové stránky budou na této datové sadě jednodušší.

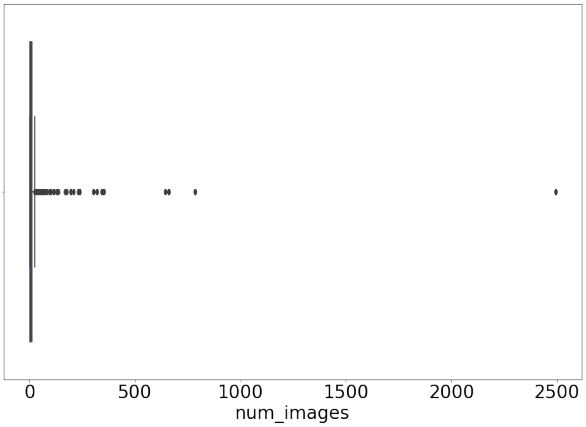
Dále jsem se zabýval tím, jaká je v datové sadě distribuce počtu obrázků a počtu skriptů na webové stránce. Zjistil jsem, že se v datové sadě vyskytují některé odlehlé hodnoty počtu obrázků na stránce, to lze vidět v grafu 3.5. Na webových stránkách v datové sadě je průměrně 14,89 obrázků a směrodatná odchylka počtu obrázků je 32,25, oboje při zanedbání odlehlých hodnot. Na grafu 3.6 je vidět distribuce počtů obrázků.

Počet výskytů počtů skriptů na webových stránkách je v tabulce 3.1. Jak lze vidět, tak na drtivé většině stránek je 6 skriptů. Protože je počet skriptů v této datové sadě téměř na všech webových stránkách stejný, rozhodl jsem se, zaměřit se spíše na odhad počtu obrázků na stránce. Větší smysl zabývat se odhadem počtu skriptů na stránce by mělo, pokud bych měl dataset, kde by byly zachycené komunikace pro více různých webů, v takovém případě by pravděpodobně byly počty skriptů mnohem rozmanitější.

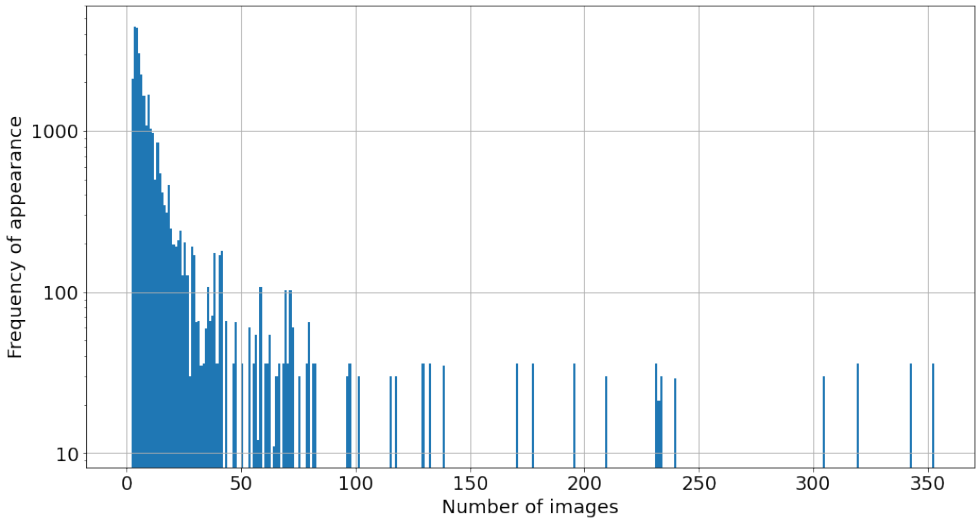
Pokusil jsem se také ověřit jaké je zastoupení jednotlivých URL v datové sadě. Vzhledem k povaze generování datové sady jsem předpokládal, že budou webové adresy v datasetu zastoupeny rovnoměrně, což se také potvrdilo.



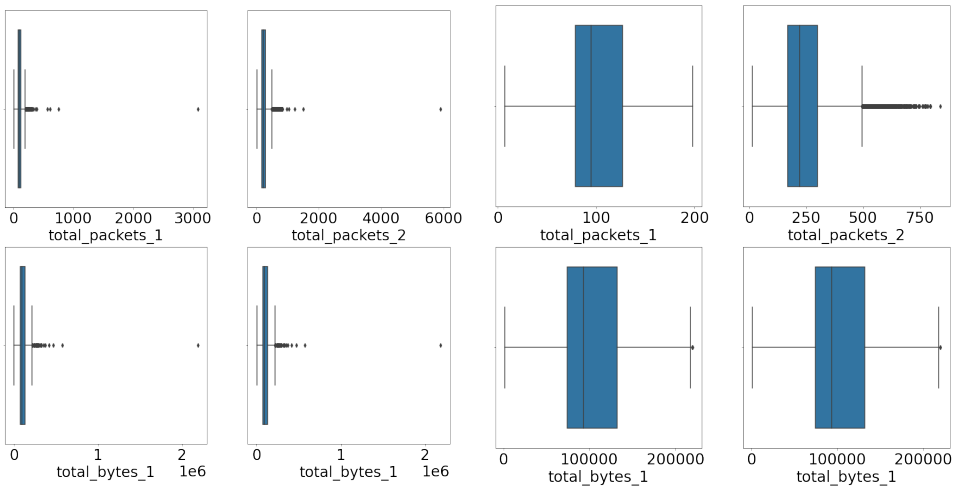
Obrázek 3.4: Ukázka několika různých návštěv jedné webové stránky s deaktivovanou cache webového prohlížeče.



Obrázek 3.5: Odlehlé hodnoty počtu obrázků na webových stránkách.



Obrázek 3.6: Distribuce počtu obrázků na stránce po odstranění odlehlých hodnot.



(a) Původní stav.

(b) Stav po odstranění odlehlých hodnot.

Obrázek 3.7: Odstranění odlehlých hodnot v rámci příznaků délky komunikací.

3.3 Realizace

3.3.1 Předzpracování dat

V rámci předzpracování dat jsem z jednotlivých komunikací odstranil prázdné a příliš krátké toky. Některé toky v datové sadě jsou totiž prázdné, to nemusí znamenat, že v daném toku nebyly přeneseny žádné pakety, nástroj Joy totiž do toku nezahrnuje TCP pakety s nulovou velikostí dat. Některé komunikace obsahovaly vedle „běžně“ dlouhých toků také velmi krátké toky. Dlouhé například jeden či dva pakety, tyto krátké toky se navíc často v komunikacích vyskytovaly nepravidelně, nejspíše náhodně. Tím je myšleno, že často pro různé návštěvy jedné webové stránky, obsahovalo jen několik málo komunikací tyto krátké toky. Na základě toho jsem usoudil, že tyto krátké toky, které mají jen jednotky paketů, mají minimální výpovědní hodnotu ve srovnaní s toky, které mají paketů desítky nebo více. Empiricky jsem zvolil odstranění toků kraších než pět paketů.

Následně jsem se pokusil datovou sadu očistit od řádků, které mají komunikaci úplně prázdnou. Ukázalo se, že takový řádek byl jen jeden.

Zjistil jsem, že v datové sadě existují nějaké odlehlé hodnoty délky komunikací. Vidět je to na grafu 3.7a na více různých příznacích. Grafy 3.7b pak zobrazují rozložení délek komunikací po odstranění odlehlých hodnot.

3.3.2 Tvorba příznaků

Pro další práci jsem vytvořil řadu nových příznaků. Vytvořil jsem dvě pomyslné kategorie příznaků, jednoduché příznaky a složitější příznaky.

Předtím, než jsem začal příznaky generovat, tak jsem ještě vytvořil sloupce `traffic_burst` a `inter_pkt_times_burst`. Jednotlivé řádky v nově vytvořeném sloupci `traffic_burst` obsahují pole polí celých čísel, které vzniklo tak, že jsem v tocích sdružil skupiny paketů, které jdou bezprostředně za sebou a mají stejný směr toku. Jednu takovou sdruženou část toku nazývám v této práci *burstem*. Tento *burst* má stejný význam jako to, co v práci [11] také nazývají *burst*. Při implementaci funkce, které tento sloupec vytvoří jsem vycházel ze zmíněné práce [11]. Sloupec `inter_pkt_times_burst` je obdobou sloupce `traffic_burst`, jen nevychází ze sloupce `traffic`, ale z `inter_pkt_times`, neboli nesdružuje velkosti souvislých posloupností paketů v jednom směru, ale mezipaketové časy souvislých posloupností paketů v jednom směru.

3.3.2.1 Jednoduché příznaky

Všechny jednoduché příznaky, jsou vytvořeny tak, že je z jednoho sloupce v `df` pomocí nějaké operace vytvořen jeden nový sloupec. Vytvořené příznaky jsou inspirovány pracemi [8], [9] a [11]. Popis jednotlivých příznaků je v následujícím výčtu.

- `total_pkts_sent` – Celkový počet odeslaných paketů.
- `total_pkts_rec` – Celkový počet přijatých paketů.
- `total_bytes_sent` – Celkový počet odeslaných bajtů.
- `total_bytes_rec` – Celkový počet přijatých bajtů.
- `total_duration` – Suma dob trvání všech toků.
- `pkts_in_out_ratio` – Poměr mezi počtem přijatých a odeslaných paketů.
- `bytes_in_out_ratio` – Poměr mezi počtem přijatých a odeslaných bajtů.
- `max_inter_packet_time` – Maximální mezipaketový čas pro odeslané pakety.
- `max_inter_packet_time_rec` – Maximální mezipaketový čas pro přijaté pakety.
- `min_inter_packet_time` – Minimální mezipaketový čas pro odeslané pakety.
- `min_inter_packet_time_rec` – Minimální mezipaketový čas pro přijaté pakety.
- `std_inter_packet_time` – Směrodatná odchylka mezipaketových časů pro odeslané pakety.

- `std_inter_packet_time_rec` – Směrodatná odchylka mezipaketových časů pro přijaté pakety.
- `mean_inter_packet_time` – Průměrný mezipaketový čas pro odeslané pakety.
- `mean_inter_packet_time_rec` – Průměrný mezipaketový čas pro přijaté pakety.
- `mean_packet_size` – Průměrná velikost paketu pro odeslané pakety.
- `mean_packet_rev_size` – Průměrná velikost paketu pro přijaté pakety.
- `std_packet_size` – Směrodatná odchylka velikostí paketů pro odeslané pakety.
- `std_packet_rev_size` – Směrodatná odchylka velikostí paketů pro přijaté pakety.
- `max_flow_len` – Maximální délka toku.
- `mean_flow_len` – Průměrná délka toku.
- `max_flow_duration` – Maximální doba trvání toku.
- `mean_flow_duration` – Průměrná doba trvání toku.
- `median_flow_duration` – Medián doby trvání toku.
- `min_burst_size` – Minimální velikost odchozího *burstu*.
- `min_burst_size_rec` – Minimální velikost příchozího *burstu*.
- `max_burst_size` – Maximální velikost odchozího *burstu*.
- `max_burst_size_rec` – Maximální velikost příchozího *burstu*.
- `mean_burst_size` – Průměrná velikost odchozího *burstu*.
- `mean_burst_size_rec` – Průměrná velikost příchozího *burstu*.
- `std_burst_size` – Směrodatná odchylka velikostí odchozích *burstů*.
- `std_burst_size_rec` – Směrodatná odchylka velikostí příchozích *burstů*.
- `max_traffic_burst_len` – Délka nejdelšího toku *burstů*.
- `total_burst_sent` – Počet odchozích *burstů*.
- `total_burst_rec` – Počet příchozích *burstů*.
- `bursts_in_out_ratio` – Poměr příchozích a odchozích *burstů*.
- `num_of_direction_switches` – Počet změn směru toku paketů.

3.3.2.2 Složitější příznaky

Další vytvořené příznaky, mají společné to, že jsou vytvořené pomocí nějaké funkce, která jako vstup bere jeden sloupec, ale vytvoří více nových příznaků, neboli sloupců. Tyto příznaky jsem vytvořil na základě článku [9].

Implementoval jsem třídu `Binner`, která slouží pro přiřazování čísel do intervalů. Konstruktor třídy přijímá parametry: typ tvorby intervalů, vstupní seznam čísel, počet intervalů a volitelně i minimální a maximální hodnotu. Na základě vstupního seznamu čísel jsou vypočítány a uloženy intervaly. Počet intervalů je dán vstupním parametrem. Třída umožňuje intervaly vytvářet dvojím způsobem.

První způsob je, že mají všechny intervaly stejnou délku. Tento způsob je použit pokud je konstruktoru jako typ tvorby intervalu předán řetězec „equal_width“. Délka intervalu je vypočtena jako $\lceil \frac{max\_value - min\_value}{num\_of\_bins} \rceil$. Pokud nejsou zadány volitelné parametry konstruktoru, minimální a maximální hodnota, pak jsou nalezeny ve vstupním seznamu.

Druhý způsob je takový, že vstupní seznam je rozdělen na intervaly tak, aby do každého intervalu padlo přibližně stejné množství prvků. Na rozdíl od prvního případu, ale vypočítané intervaly nebudou mít stejné délky. Tento způsob je použit, pokud je v konstruktoru předána hodnota „equal_height“.

Poté co je objekt konstruktorem zkonstruován, je možné používat metodu `get_bin`, která pro parametrem zadané vstupní číslo vrací číslo intervalu, do kterého vstupní číslo spadá.

V konstruktoru je, jak již bylo řečeno, možné nastavit maximální a minimální hodnotu. Pokud je tak učiněno, pak jsou hodnoty větší než maximum a hodnoty menší než minimum při tvorbě intervalů ignorovány. To znamená, že později při použití metody `get_bin`, jsou hodnoty větší než bylo zvolené maximum a hodnoty menší než bylo zvolené minimum, zařazovány do nejvyššího, respektive nejnižšího intervalu.

Objekt třídy `Binner` využívá třída `HistConverter`. Objekt této třídy ze vstupního seznamu čísel vytvoří histogram na základě intervalů objektu třídy `Binner`. Neboli, vrací seznam, který má délku rovnu počtu intervalů. Pro každé číslo ze vstupního seznamu je inkrementována hodnota v seznamu výstupním, na pozici, která odpovídá číslu intervalu, do kterého hodnota patří. `HistConverter` také umožňuje zpracování seznamu vnořených seznamů hodnot. A to dvěma způsoby. Jedna varianta je taková, že je histogram vytvořen ze všech seznamů, to znamená, že všechny seznamy v seznamu jsou započítány do výsledného histogramu, nebo na toto lze také nahlížet jako na součet jednotlivých histogramů pro jednotlivé vnořené seznamy. Druhá varianta je ta, že `HistConverter` vybere pouze nejdelší podseznam a z něho histogram vytvoří.

Objekt třídy `HistConverter` jsem využil pro tvorbu příznaků ze sloupců `traffic`, `inter_pkt_times`, `traffic_burst` a `inter_pkt_times_burst`. Pro každý řádek v datové sadě `HistConverter` vezme obsah daného sloupce a vytvoří na základě něj histogram, ten je pak vrácen ve formě seznamu četností

hodnot, každá položka tohoto seznamu je použita jako nově vzniklý příznak. Z jednoho sloupce je tedy vytvořeno tolik příznaků, jako byl počet intervalů použitého objektu třídy `Binner`.

Druhou implementovanou třídou určenou pro tvorbu složitějších příznaků, která využívá `Binner` je třída `MarkovConverter`. Tato třída poskytuje několik metod, z nichž nejdůležitější je metoda `get_flat_markov_chain`.

Metoda `get_flat_markov_chain` této třídy bere jako vstupní parametr pole číselných hodnot. Toto pole je následně pomocí objektu třídy `Binner` nahrazeno indexy intervalů, do kterých jednotlivé hodnoty spadají. Poté je vytvořena matice $\mathbf{A}_{n,n}$, kde n je počet intervalů. Prvky této matice jsou rovny 0. Následně je postupně procházeno pole indexů intervalů. Pro každé dva sousední prvky v poli, tj. prvky na pozicích i a $i+1$, jejichž hodnoty jsou x a y, je inkrementován prvek $a_{x,y}$ matice $\mathbf{A}$. Po zpracování celého pole je matice $\mathbf{A}$ „normalizována“, protože musí splňovat podmínku, že součty prvků v řádcích se rovnají jedné, tj. $\forall s = 0, \ldots n-1 : \sum_{j=0}^{n-1} a_{s,j} = 1$, přičemž jednotlivé prvky matice musí být v intervalu $(0, 1)$. Tato matice je nakonec zploštěna, tj. je převedena na vektor hodnot, tak jako bychom prvky matice četli zleva doprava, ze shora dolů a postupně je zapisovali do jednoho řádku. Výsledný vektor je použit jako vektor příznaků.

Tento přístup, který jsem implementoval na základě [9], je založen na Markovských řetězcích s diskrétním časem [26]. Zmiňovaná matice $\mathbf{A}$ je maticí pravděpodobností přechodu. Kde hodnota $a_{x,y}$, má význam pravděpodobnosti přechodu mezi „stavem“ x a „stavem“ y. Konkrétně tedy například v případě toku to znamená, že hodnota prvku $a_{x,y}$ matice $\mathbf{A}$ je pravděpodobností přechodu mezi paketem o velikosti spadající do intervalu číslo x a paketem o velikosti která spadá do intervalu číslo y.

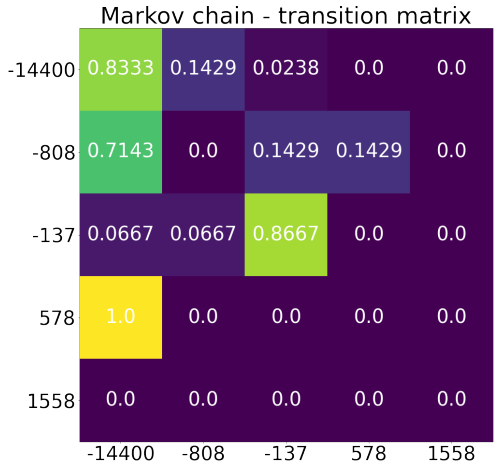
Na obrázku 3.8 je příklad matice pravděpodobností přechodu pro jeden tok, která vznikla na základě komunikace a objektu `Binner` s pěti intervaly typu „equal_height“.

Další užitečnou metodou, kterou třída `MarkovConverter` nabízí je metoda `get_markov_features`. Tato metoda umožňuje, zpracovat seznam vnořených seznamů. Podobně jako v případě `HistConverter` umožňuje zpracovat vnořené seznamy dvěma způsoby, tedy použitím pouze nejdelšího podseznamu, nebo součtem pro všechny podseznamy.

3.3.2.3 Generování datových sad s novými příznaky

Pro účely experimentů jsem vygeneroval několik datových sad s různými množinami příznaků.

- `df_simple` – Tento dataset obsahuje pouze jednoduché příznaky, zmiňované v sekci 3.3.2.1.
- `df_complex10` – Tato datová sada obsahuje pouze markovské příznaky vygenerované nad sloupcem `traffic`, s maticí přechodů o délce strany



Obrázek 3.8: Ukázka matice pravděpodobností přechodu vzniklé z velikostí paketů v toku.

deset. Použity byly intervaly typu „equal_height“ a byla nastavena maximální a minimální hodnota 1500 a -1500.

- `df_complex5a` – Obsahuje pouze markovské příznaky vygenerované nad sloupcem `traffic`. Použito bylo pět intervalů typu „equal_height“ a byla nastavena maximální a minimální hodnota 1500 a -1500. Matice přechodů tedy má délku strany 5.

- `df_complex5b` – Je stejný jako `df_complex5a` až na to, že při tvorbě intervalů nebyla nastavena maximální a minimální hodnota.

- `df_hist` – Tento dataset používá jako příznaky histogramy nad sloupci `traffic`, `traffic_burst`, `inter_pkt_times` a `inter_pkt_times_burst`. Každý histogram používá pět intervalů typu „equal_height“.

- `df_complex` – Tato datová sada obsahuje markovské příznaky i histogramy nad sloupci `traffic`, `traffic_burst`, `inter_pkt_times` a také `inter_pkt_times_burst`.

- `df_complex_longest` – Tato datová sada byla vytvořena stejným způsobem jako datová sada `df_complex`, s tím rozdílem, že na rozdíl od všech ostatních datových sad byly pro tvorbu markovských a na histogramech založených příznaků z každé komunikace použit pouze nejdelší tok, kdežto u ostatních datových sad byly využity všechny toky v komunikaci.

- `df_all` – Tento dataset je spojením datasetů `df_simple` a `df_complex`.

3.3.3 Rozpoznání webové stránky

V této sekci popíši postup rozpoznávání webové stránky na základě vektoru příznaků, nezávisle na zvolené verzi datasetu, ale ukázky výstupů demonstruji na datové sadě `df_simple`.

Z datasetu jsem nejdříve odstranil sloupce, které k řešení této úlohy nepotřebuji. Odstranil jsem sloupce metadat a sloupce které byly využity pro generování příznaků. Například jsem odebral sloupec `traffic`, ze kterého bylo vygenerováno mnoho příznaků, ale sám obsahuje vnořené seznamy, což se pro klasifikaci nehodí, protože klasifikátory s takovými daty obyčejně pracovat neumí. Tím jsem získal datovou sadu, které má ve všech sloupcích pouze numerické hodnoty.

Poté jsem zkontroloval, jestli se v datech nevyskytují hodnoty jako `NaN` (Not a Number) nebo `Inf` (Infinity). V datech jsem ovšem žádné takové hodnoty neměl, jen během ladění se mi pár takových hodnot vyskytlo, ale byly jich jen jednotky a tak jsem řádky s nimi odstranil.

Jak již bylo řečeno, datová sada s vypnutou cache obsahuje 915 unikátních webových adres. Data jsem náhodně rozdělil na dvě části podle adres tak, že první část má 215 unikátních webových adres a druhá 700 unikátních adres. Žádná URL není v obou částech zároveň. První část datové sady představuje webové stránky, které klasifikátorem nebyly viděny a naopak druhá část představuje webové stránky, které viděny byly. Viděný dataset jsem rozdělil dále na trénovací, validační a testovací podmnožiny. Rozdělení jsem provedl takovým způsobem, aby v trénovací množině byl od každé třídy alespoň jeden vzorek, toto lze dosáhnout pomocí parametru `stratify` funkce `train_test_split`, která je součástí knihovny Scikit-learn. Parametr `stratify` určuje, podle kterého sloupce nebo sloupců je rozdělení provedeno, v tomto případě to byl sloupec `url`. Výsledkem jsou následující podmnožiny,

- trénovací podmnožina, která obsahuje 16 590 komunikací,
- validační podmnožina, která obsahuje 4977 komunikací,
- testovací podmnožina, která obsahuje 2134 komunikací.

Definoval jsem slovník s klasifikátory a slovník s hyperparametry pro každý klasifikátor. Tak, abych mohl jednoduše ladit hyperparametry jednotlivých klasifikátorů, vyzkoušet úspěšnost jednotlivých klasifikárotů a na základě toho vybrat nejlepší klasifikátor s co nejlepšími hyperparametry. Použité klasifikátory a jejich hyperparametry jsou v příloze D.

Po zadefinování klasifikátorů a hyperparametrů jsem přistoupil k výběru příznaků. Výběr vhodných příznaků je v případě datasetů, které mám k dispozici důležitý, protože v případě ručně vytvořených příznaků lze předpokládat, že některé příznaky budou vzájemně silně korelované. Silně korelované příznaky mít v modelu obecně nechceme, protože nové silně korelované příznaky, nepřinášejí novou informaci a nezlepšují model. Jedna z vlastností dobrého

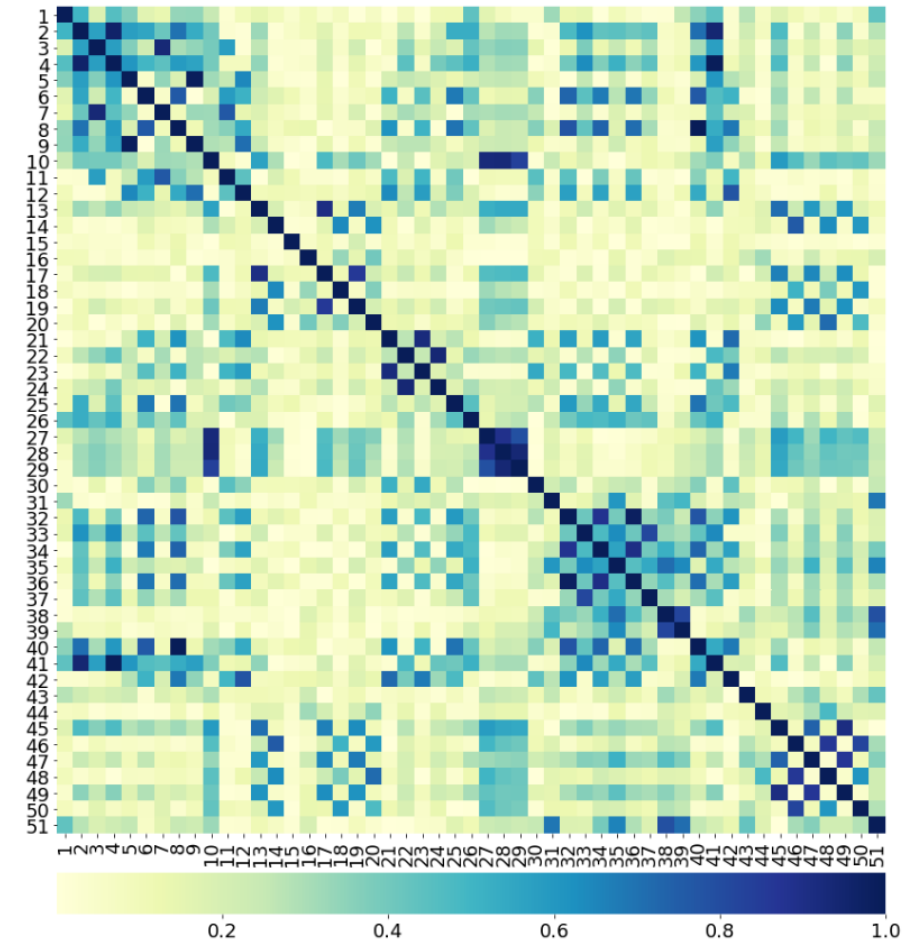
modelu je, že by měl být jednoduchý. Další věc je, že v případě datasetů s příznaky, které jsou výstupem `HistConverter` a `MarkovConverter` dostáváme poměrně hodně příznaků, například v případě markovských pravděpodobností přechodů při rozdělení vstupu do deseti intervalů dostaneme 100 příznaků a o těchto příznacích dopředu nevíme, které budou vhodné a které ne.

Nejdříve jsem přistoupil k odstraňovaní příznaků, které jsou vysoce korelované. K tomuto účelu jsem vytvořil funkci `get_correlated_features`.

Tato funkce nejdříve pomocí funkce `corr` integrované v Pandas vypočte pro vstupní dataset korelační matici, neboli pro každý pár příznaků vypočte vzájemnou korelaci pomocí zvolené metody. Funkce `corr` jako metody pro výpočet korelace nabízí Pearsonův korelační koeficient [29] a Spearmanův korelační koeficient [30]. V této funkci jsem zvolil použití Spearmanova koeficientu. Pearsonův koeficient je vhodný pro lineární závislost mezi veličinami, předpokládá normální rozdělení veličin a je náchylný k odlehlým hodnotám, já nic z toho pro vstupní příznaky zaručit nemohu. Naopak Spearmanův koeficient je robustní vůči odlehlým hodnotám a odchylkám od normality. Spearmanův koeficient popisuje, jak dobře vztah veličin odpovídá monotónní funkci, vztah mezi veličinami tedy ani nemusí být lineární. Hodnoty Spearmanova koeficientu se pohybují od -1 do 1. Hodnot kolem nuly nabývá v případě, že mezi veličinami není žádný vztah, naopak hodnota 1 nebo -1 znamená, že jedna veličina je monotónní funkcí veličiny druhé. V tomto odstavci jsem také čerpal z knihy [19].

Funkce `get_correlated_features` poté korelační matici vykreslí, ukázka pro datovou sadu `df_simple` je na obrázku 3.9. A následně pro každou dvojici příznaků, jejichž absolutní hodnota korelačního koeficientu je vyšší než práh, který je parametrem funkce, náhodně vybere jeden z příznaků a ten z datasetu odstraní. Během procházení dvojic příznaků si však funkce zapamatovává dosud odstraněné příznaky a bere na ně ohled při odstraňování následujících dvojicí, tedy pokud jeden (nebo oba) příznak z odstraňované dvojice již odstraněn byl, je tato dvojice přeskočena a funkce pokračuje odstraňováním dvojice následující. V kódu 3.4 je ukázka výpisu funkce `get_correlated_features` při běhu, stejně jako v předchozí zmíněné ukázce korelační matice, na datasetu `df_simple` s prahem 0,9.

Pro další snížení dimenzionality jsem použil nástroj *Recursive feature elimination with cross-validation* (RFECV) [31] knihovny Scikit-learn. Tento nástroj pomocí externího modelu (klasifikátoru, regresoru, ...) postupně eliminuje příznaky z datové sady. Model je vždy na iniciální datové sadě natrénován, na základě modelu je určena důležitost jednotlivých příznaků a následně je nejméně důležitý příznak z datové sady odebrán. Tato procedura je rekurzivně opakována na čím dál menších a menších datových sadách dokud není dosaženo požadovaného počtu příznaků. Pro dosažení lepších výsledků využívá metoda RFECV křížovou validaci. Při použití tohoto nástroje jsem vycházel z článků [33] a [32]. Použití RFECV probíhá následujícím způsobem.



1 - num_flows
2 - total_packets_1
3 - total_packets_2
4 - total_bytes_1
5 - total_bytes_2
6 - total_pkts_sent
7 - total_pkts_rec
8 - total_bytes_sent
9 - total_bytes_rec
10 - total_duration
11 - pkts_in_out_ratio
12 - bytes_in_out_ratio
13 - max_inter_packet_time
14 - max_inter_packet_time_rec
15 - min_inter_packet_time
16 - min_inter_packet_time_rec
17 - std_inter_packet_time
18 - std_inter_packet_time_rec
19 - mean_inter_packet_time
20 - mean_inter_packet_time_rec
21 - mean_packet_size
22 - mean_packet_rev_size
23 - std_packet_size
24 - std_packet_rev_size
25 - max_flow_len
26 - mean_flow_len
27 - max_flow_duration
28 - mean_flow_duration
29 - median_flow_duration
30 - min_burst_size
31 - min_burst_size_rec
32 - max_burst_size
33 - max_burst_size_rec
34 - mean_burst_size
35 - mean_burst_size_rec
36 - std_burst_size
37 - std_burst_size_rec
38 - max_traffic_burst_len
39 - mean_traffic_burst_len
40 - total_burst_sent
41 - total_burst_rec
42 - bursts_in_out_ratio
43 - min_ipt_burst_size
44 - min_ipt_burst_size_rec
45 - max_ipt_burst_size
46 - max_ipt_burst_size_rec
47 - mean_ipt_burst_size
48 - mean_ipt_burst_size_rec
49 - std_ipt_burst_size
50 - std_ipt_burst_size_rec
51 - num_of_direction_swtches

Obrázek 3.9: Ukázka korelační matice pro datovou sadu `df_simple`.

```
To be dropped: total_bytes_1 or total_packets_1: 0.958
total_bytes_1 - added to list of features to be dropped
To be dropped: total_pkts_rec or total_packets_2: 0.936
total_packets_2 - added to list of features to be dropped
To be dropped: total_bytes_rec or total_bytes_2: 0.991
total_bytes_2 - added to list of features to be dropped
To be dropped: std_inter_packet_time or max_inter_packet_time:
0.911
max_inter_packet_time - added to list of features to be dropped
To be dropped: std_packet_size or mean_packet_size: 0.924
std_packet_size - added to list of features to be dropped
To be dropped: std_packet_rev_size or mean_packet_rev_size: 0.94
std_packet_rev_size - added to list of features to be dropped
To be dropped: max_flow_duration or total_duration: 0.925
max_flow_duration - added to list of features to be dropped
To be dropped: mean_flow_duration or total_duration: 0.932
total_duration - added to list of features to be dropped
To be dropped: median_flow_duration or mean_flow_duration: 0.937
mean_flow_duration - added to list of features to be dropped
To be dropped: std_burst_size or max_burst_size: 0.981
std_burst_size - added to list of features to be dropped
To be dropped: std_burst_size or mean_burst_size: 0.907
std_burst_size - already dropped
To be dropped: total_burst_sent or total_bytes_sent: 0.99
total_burst_sent - added to list of features to be dropped
To be dropped: total_burst_rec or total_packets_1: 0.938
total_burst_rec - added to list of features to be dropped
To be dropped: total_burst_rec or total_bytes_1: 0.991
total_bytes_1 - already dropped
To be dropped: std_ipt_burst_size or max_ipt_burst_size: 0.907
std_ipt_burst_size - added to list of features to be dropped
```

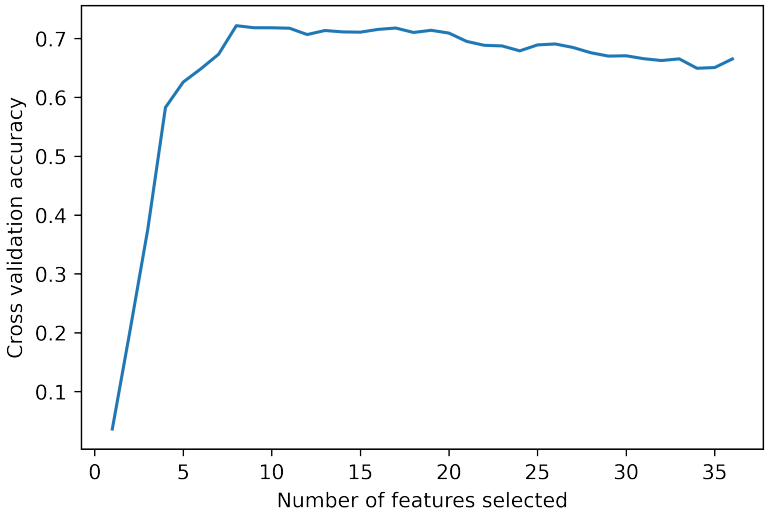
Zdrojový kód 3.4: Ukázka výpisu funkce `get_correlated_features` pro datovou sadu `df_simple` a práh 0,9.

```python
def tune_classifier(scaler, classifier, param_grid, scoring, cv, n_jobs,
↪  X, y, SCV=GridSearchCV):
    # Define steps in pipeline
    steps = [("scaler", scaler), ("classifier", classifier)]

    # Initialize Pipeline objectPipeli
    pipeline = Pipeline(steps = steps)

    # Initialize hyperparameters search object (GridSearchCV,
    ↪  RandomizedSearchCV)
    scv = SCV(pipeline, param_grid, cv=cv,  n_jobs=n_jobs,
    ↪  scoring=scoring, verbose=1)

    scv.fit(X, np.ravel(y))

    # Get best parameters and score
    best_params = scv.best_params_
    best_score = scv.best_score_

    # Update classifier parameters
    tuned_params = {item[12:]: best_params[item] for item in best_params}
    classifier.set_params(**tuned_params)

    print(f'Train - best params: {best_params}')
    print(f'Train - best score: {best_score}')
    print(f'Params: {tuned_params}')

    return classifier, scv
```

Zdrojový kód 3.5: Ukázka funkce `tune_classifier`.

Jako první je třeba zvolit a natrénovat (na trénovací datové sadě) vhodný klasifikátor. Zvolený klasifikátor musí poskytovat informaci o důležitosti jednotlivých příznaků buď prostřednictvím atributu `coef_`, nebo prostřednictvím atributu `feature_importances_`. Pro toto jsem zvolil Random Forest klasifikátor, jenž jsem natrénoval pomocí implementované funkce `tune_classifier` 3.5. Funkce `tune_classifier` umožňuje nalezení optimálních hyperparametrů daného klasifikátoru, pomocí některého optimalizátoru hyperparametrů, tj. například pomocí *GridSearchCV* [34] nebo *RandomizedSearchCV* [35], jenž jsou součástí knihovny Scikit-learn. Jaký optimalizátor hyperparametrů je ve funkci `tune_classifier` použit je dáno parametrem této funkce. Klasifikátor použitý pro výběr příznaků nástrojem RFECV jsem, stejně jako klasifikátor později použitý pro samotnou klasifikaci, používal v pipeline [37] z knihovny Scikit-learn. Pipeline slouží k postupnému aplikování seznamu transformací a konečného modelu(klasifikátoru, regresoru, ...). Jako transformaci jsem totiž použil *StandardScaler* [38], který příznak standardizuje tak, že má výsledný příznak nulovou střední hodnotu a jednotkový rozptyl. To je provedeno odečtením výběrového průměru od původního příznaku a vydělením směrodatnou odchylkou, tj. $x' = \frac{x-\overline{x}}{\sigma}$.

Následně je pomocí zmíněného klasifikátoru a nástroje RFECV přibližně určen optimální počet příznaků. Křivka, která je výstupem ladění počtu příznaků pomocí RFECV pro datovou sadu `df_simple` je na obrázku 3.10. Na

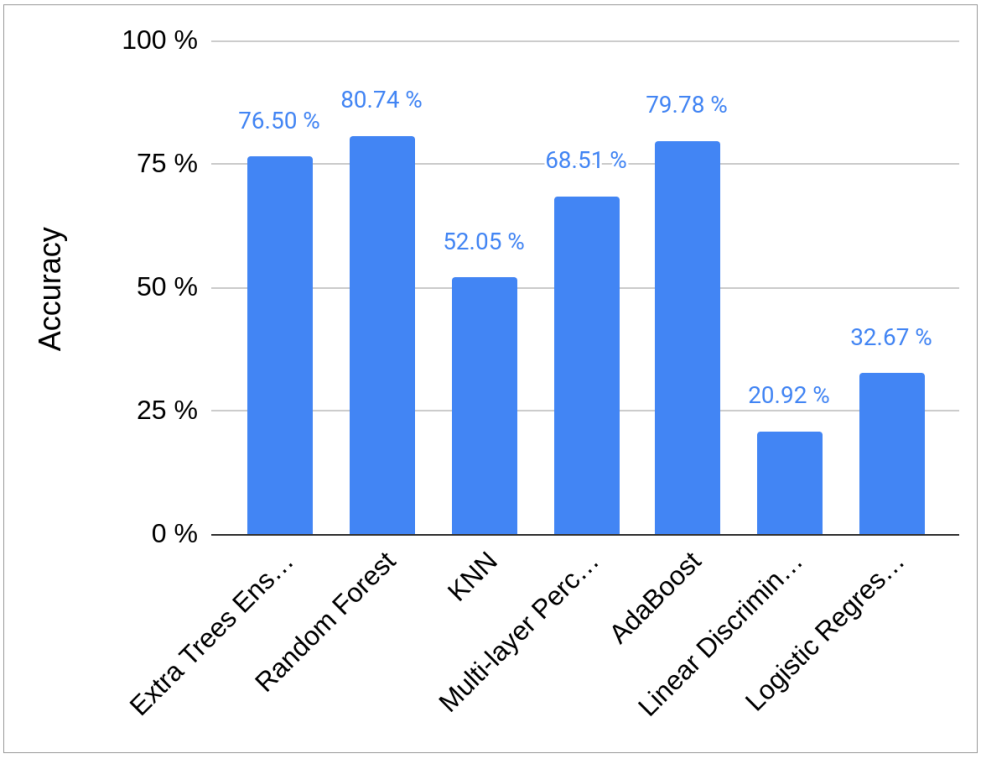


Obrázek 3.10: Křivka ladění počtu příznaků pro datovou sadu `df_simple`.

základě tohoto výstupu jsem pak zvolil, že má být použito 15 příznaků.

A nakonec jsem znovu pomocí RFECV, postupnou rekurzivní eliminací příznaků, získal 15 nejvhodnějších příznaků. V případě datové sady `df_simple` to byly příznaky:

- `total_packets_1`,
- `total_pkts_rec`,
- `total_bytes_sent`,
- `total_bytes_rec`,
- `bytes_in_out_ratio`,
- `mean_packet_size`,
- `max_flow_len`,
- `mean_flow_len`,
- `min_burst_size`,
- `min_burst_size_rec`,
- `max_burst_size_rec`,
- `mean_burst_size_rec`,
- `std_burst_size_rec`,



Obrázek 3.11: *Accuracy* klasifikátorů (identifikace webových stránek) na testovací podmnožině datové sady `df_simple`.

- `bursts_in_out_ratio` a
- `max_ipt_burst_size`.

Na počátku dataset `df_simple` obsahoval 51 příznaků, počet příznaků byl technikami popsanými výše postupně redukován na 15.

Nad datovou sadou, která obsahuje již jen vybrané příznaky, jsem pak spustil mnou implementovanou funkci `evaluate_classifiers`. Tato funkce nejdříve nalezne nejvhodnější hyperparametrey a natrénuje klasifikátor pomocí již zmiňované funkce 3.5. Poté je vyhodnocena kvalita klasifikátoru. A nakonec jsou do slovníku, kde je klíčem identifikátor daného klasifikátoru, uloženy pro daný klasifikátor zvolené hyperparametry, dosažená *accuracy* a výstup funkce `classification_report` [36] jenž je součástí knihovny Scikit-learn. Toto je provedeno pro každý definovaný klasifikátor a jeho hyperparametry, použité klasifikátory a jejich hyperprametry jsou v příloze D. Funkce `evaluate_classifiers` nakonec vrátí zmíněny slovník, pro všechny použité klasifikátory.

Graf na obrázku 3.11 zobrazuje dosažené výsledky (*accuracy*) na datové sadě `df_simple` pro testovací množinu.

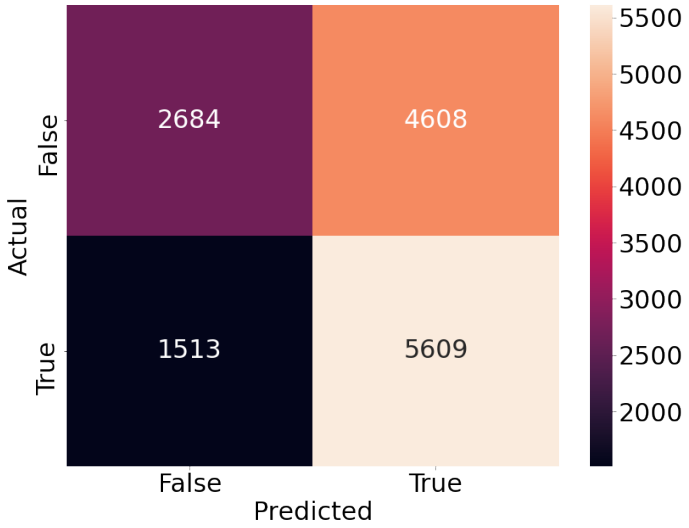
3.3.4 Detekce nových webových stránek

V této sekci se zabývám tím, zda je možné o novém síťovém provozu prohlásit, jestli webová stránka jejíž načtení provoz vyvolalo byla v trénovací množině nebo ne. Následující algoritmy jsem se pokusil natrénovat pomocí trénovací podmnožiny tzv. viděného datasetu. Následně jsem spojil testovací část viděného datasetu a neviděnou datovou sadu. Vzorky této spojené datové sady se v tomto případě nesnažím zařadit do jedné z mnoha kategorií, jako v předchozím případě, ale pouze do dvou, a to do kategorií „známá webová stránka“ a „neznámá webová stránka“. V testovací datové sadě bylo 7292 vzorků, které viděny nebyly (neznámé stránky) a 7122 vzorků, které viděny byly (známé stránky). Pro jednoduchost jsem použil pouze sadu příznaků vybranou v předchozí sekci. Metody zmíněné níže se pokoušejí odpovědět na otázku, jestli konkrétní komunikace byla vyvolána načtením známé (viděné) webové stránky.

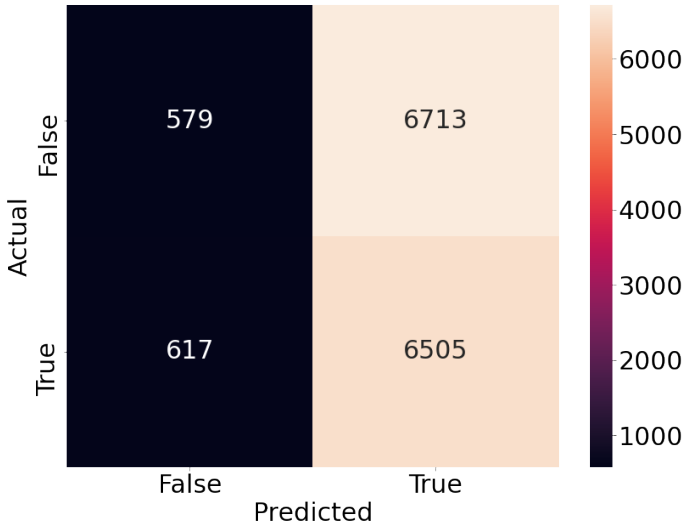
Nejprve jsem se pokusil použít metodu LocalOutlierFactor [40]. Tato metoda slouží pro detekci anomálií, ale například v článku [40], je využívána pro tzv. „Novelty detection“, neboli rozpoznání toho, že se jedná vzorek z nové/neznámé třídy. Tento algoritmus pracuje na podobném principu jako KNN, funguje zhruba následovně. Vzdálenost nejbližších sousedů prvku se používá jako jakási hustota. Lokální hustota prvku, u kterého chceme určit, zda se jedná prvek neznámé třídy, je porovnávána lokálními hustotami ostatních prvků. Lze říci, že pokud je hustota prvku podstatně nižší než hustoty ostatních prvků, pak se jedná o prvek z třídy, která dosud nebyla viděna. Použitím této metody na datasetu `df_simple` jsem dosáhl výsledků, které jsou ilustrovány v „confusion matrix“ na obrázku 3.12. Těchto výsledků bylo dosaženo po experimentálním ladění hyperparametrů této metody a jejich nastavení na hodnoty `n_neighbors=2`, `novelty=True` a `contamination=0.2`.

Druhým algoritmem, který jsem vyzkoušel, je IsolationForest [42]. Tento algoritmus je nejčastěji používán pro detekci anomálií v datech, ale je možné ho použít i pro detekci vzorků z nových tříd. S tímto algoritmem jsem dosáhl výsledků naznačených v obrázku 3.13. Tento model silně preferuje výstup, že se jedná o známou stránku a to i přesto, že obě třídy jsou v testovacích datech vyvážené a to, že jsem ladil různé varianty kombinací hyperparametrů tohoto modelu.

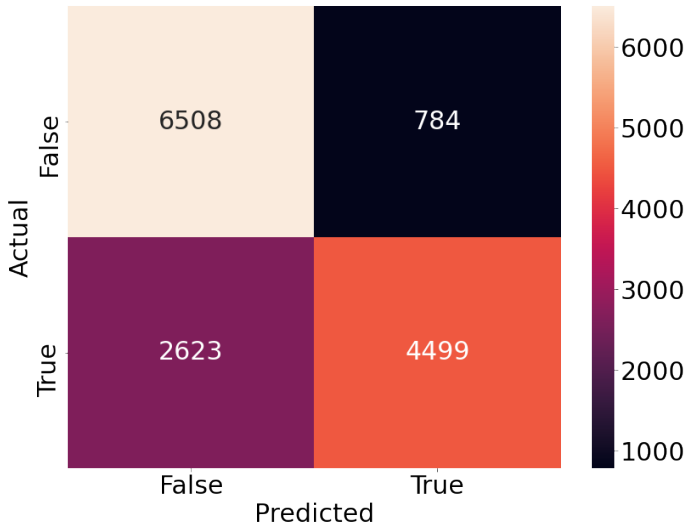
Jako třetí jsem se pokusil použít klasifikátor již naučený v předchozí části. Ze všech klasifikátorů použitých pro rozpoznávání webových stránek jsem vybral klasifikátor s nejlepší dosaženou *accuracy*, který zároveň poskytuje informaci o pravděpodobnosti příslušnosti k jednotlivým kategoriím prostřednictvím metody `predict_proba`. Rozhodl jsem se pro klasifikátor RandomForest, tento klasifikátor metodu `predict_proba` poskytuje. Toho jsem se rozhodl využít ve spojení s jednoduchou prahovací funkcí. A to tak, že při klasifikaci datasetu, který obsahuje jak viděné, tak neviděné webové stránky, je pro každý vzorek nejprve zavolána metoda `predict_proba`, ta vrátí pravděpodobnosti



Obrázek 3.12: Výsledky predikce zda se jedná o známou webovou stránku, pomocí metody LocalOutlierFactor.



Obrázek 3.13: Výsledky predikce zda se jedná o známou webovou stránku, pomocí metody IsolationForest.



Obrázek 3.14: Výsledky predikce zda se jedná o známou webovou stránku, pomocí klasifikátoru RandomForest a jednoduché prahovací funkce s prahem nastaveným na hodnotu 0,3.

Model	*Accuracy*
LocalOutlierFactor	0,575
IsolationForest	0,491
RandomForest + práh	0,763

Tabulka 3.2: Distribuce počtu skriptů na stránce.

příslušnosti ke každé třídě (URL). Prahovací funkce pak, na základě zadaného prahu, rozhodne, pokud jsou všechny pravděpodobnosti příslušnosti nižší než práh, pak je rozhodnuto, že se jedná o neznámou webovou stránku, pokud je tomu naopak, pak je rozhodnuto, že se jedná o známou webovou stránku. Na obrázku 3.14 je „confusion matrix“ pro datovou sadu `df_simple` s prahem 0,3. Jak lze vidět, tak tato metoda dosáhla poměrně dobrého výsledku a to i přes to, že jsem pro rozhodnutí o tom, zda se jedná o viděnou či neviděnou stránku použil jen velmi jednoduchou prahovací funkci. Jsem si jistý, že je možné vymyslet lepší metodu jak na základě výstupu metody `predict_proba` rozhodnout zda se jedná o viděnou či neviděnou webovou stránku.

Porovnání dosažené *accuracy* pro všechny tři použité metody je v tabulce 3.2. Jak lze vidět, tak první dvě metody mají o poznání horší *accuracy* než použití již naučeného klasifikátoru ve spojení s jednoduchou prahovací funkcí. Jak jsem již zmínil poslední metodu by bylo možné dále zlepšit použitím sofistikovanější rozhodovací funkce, než jen jednoduchého prahu.

Na tento experiment jsem navázal tím, že jsem se pokusil spojit detekci neznámé webové stránky a identifikaci webových stránek do jednoho experi-

mentu. Pokusil jsem se určit, jaká je *accuracy* klasifikace webových stránek na datové sadě vzniklé spojením testovací datové sady viděných webových stránek a datové sady neviděných webových stránek, přičemž počet kategorií, do kterých bylo klasifikováno byl roven počtu známých webových stránek plus jedna kategorie, která představuje neznámé webové stránky. Klasifikace probíhala tak, že jsem pro každý testovací vzorek nejprve použil klasifikátor společně s prahovací funkcí a tím určil zda se jedná o viděnou webovou stránku, pokud klasifikátor společně s prahovací funkcí určil, že se jedná o stránku, která viděna nebyla, byl vzorek přiřazen do kategorie reprezentující neznámé webové stránky, pokud určil opak, byla vzorku přiřazena kategorie, kterou určil klasifikátor. Na datové sadě `df_simple` s již zmiňovaným RandomForest klasifikátorem a prahem nastaveným na 0,3 dosáhl *accuracy* 75,7 %.

3.3.5 Odhad počtu obrázků na webové stránce

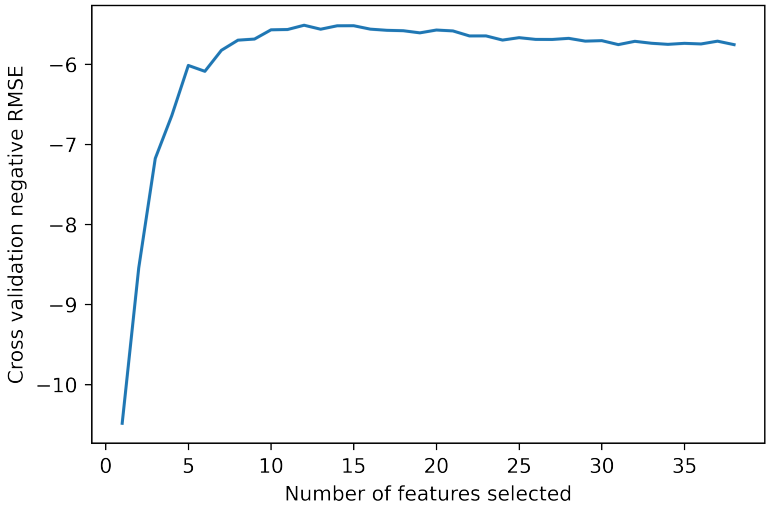
Pro odhad počtu obrázků na webové stránce jsem postupoval podobně jako v případě identifikace webových stránek. Stejně tak jako v předchozí sekci popíši postup odhadu počtu obrázků na webové stránce, nezávisle na zvolené verzi datasetu, ale ukázky výstupů demonstruji na datové sadě `df_simple`. Datovou sadu jsem prozkoumal, připravil a předzpracoval analogicky jako v případě identifikace.

Stejně jako jsem v předchozí sekci definoval slovník s klasifikátory a slovník s hyperparametry pro každý klasifikátor, tak jsem definoval i slovník s regresory a slovník s hyperparametry pro každý regresor. Tak, abych mohl jednoduše ladit hyperparametry jednotlivých regresorů, vyzkoušet úspěšnost jednotlivých regresorů a na základě toho vybrat nejlepší regresor s co nejlepšími hyperparametry. Použité regresory a jejich hyperparametry jsou v příloze E.

Při rozdělování této datové sady na podmnožiny jsem na rozdíl od předchozí podkapitoly, nepoužil parametr `stratify` funkce `train_test_split`, protože v tomto případě se snažíme odhadovat numerickou diskrétní hodnotu, a tak by použití tohoto parametru při rozdělování datové sady nedávalo smysl.

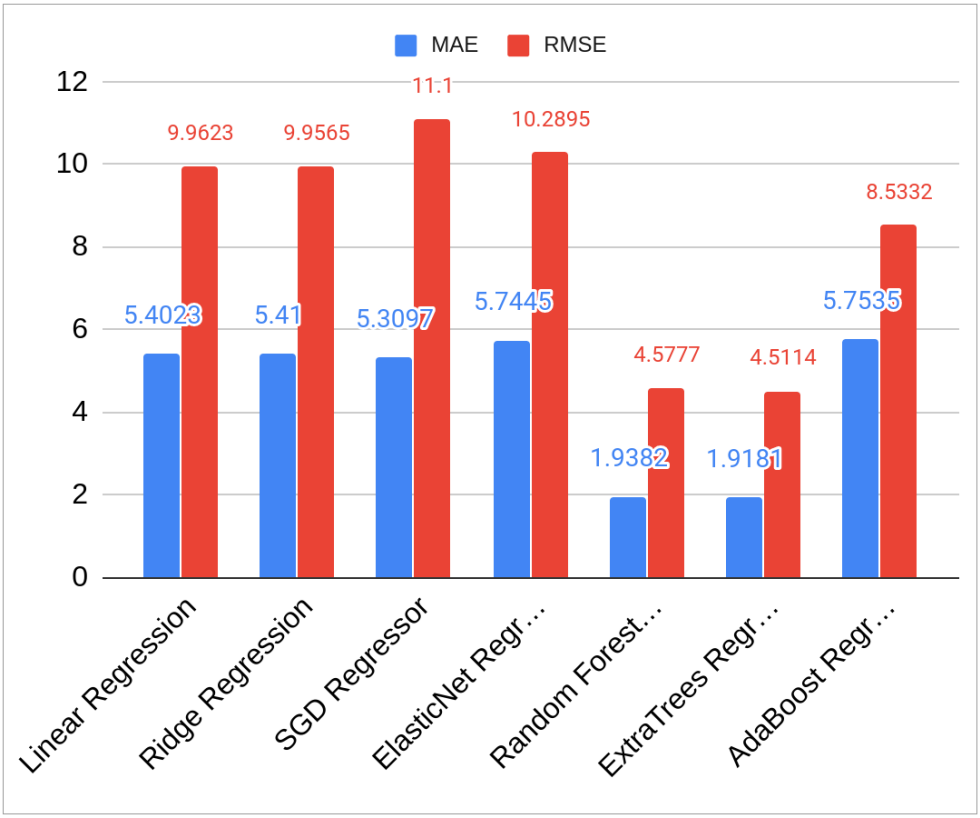
Poté, co jsem datovou sadu prozkoumal a předpřipravil, provedl jsem selekci příznaků stejným způsobem jako v předchozím případě. Stejně jako v předchozím případě jsem nejdříve provedl odstranění příznaků na základě vzájemné korelace. Pro datovou sadu `df_simple` jsem zvolil práh pro funkci `get_correlated_features` 0,9. Poté jsem tak jako u identifikace webových stránek použil metodu RFECV s tím rozdílem, že jsem na místo Random Forest klasifikátoru použil Random Forest regresor. Jako první jsem se pokusil určit vhodný počet příznaků. Graf, který zobrazuje výstup ladění počtu příznaku pomocí RFECV je na obrázku 3.15. Na základě tohoto výstupu jsem zvolil použití patnácti příznaků. Pomocí postupné rekurzivní eliminace příznaků nástrojem RFECV jsem získal 15 nejvhodnějších příznaků:

- `total_packets_2`,



Obrázek 3.15: Křivka ladění počtu příznaků pro datovou sadu `df_simple` pro úlohu odhadu počtu obrázků na webové stránce.

- `total_bytes_2`,
- `total_pkts_sent`,
- `total_bytes_sent`,
- `bytes_in_out_ratio`,
- `mean_packet_rev_size`,
- `std_packet_size`,
- `max_flow_len`,
- `mean_flow_len`,
- `min_burst_size`,
- `min_burst_size_rec`,
- `max_burst_size`,
- `max_burst_size_rec`,
- `bursts_in_out_ratio` a
- `std_ipt_burst_size`.



Obrázek 3.16: RMSE a MAE regresních modelů pro odhad počtu obrázků na webové stránce. Výsledky byly dosaženy na testovací podmnožině datové sady `df_simple`.

Nad vzniklou datovou sadou s patnácti příznaky jsem pak spustil funkci `evaluate_regressors`, která je obdobou, v kapitole 3.3.3 zmiňované, funkce `evaluate_classifiers`. Tato funkce stejně jako `evaluate_classifiers` pro každý model najde nejvhodnější hyperparametry, tentokrát ovšem pro regresní modely, následně je natrénuje, vyhodnotí jejich kvalitu a vrátí slovník, který pro každý model obsahuje zvolené parametry, MAE, RMSE a další. Regresní modely a parametry použité pro ladění hyperparametrů jsou v příloze E. Analogicky jako v případě identifikace webových stránek jsem v případě odhadu počtu obrázků na webové stránce použil pro normalizaci příznaků *StandardScaler*. Dosažené výsledky pro viděnou testovací podmnožinou datové sady `df_simple` jsou pro různé regresní modely ukázány na obrázku 3.16.

Dále jsem se pokusil ověřit, jaká je úspěšnost regresního modelu pro odhad počtu obrázků pouze pro neznámé webové stránky z datové sady `df_simple`. Použil jsem tedy již natrénované modely na trénovací podmnožině `df_simple` a vyhodnotil jejich úspěšnost na neviděné podmnožině datasetu `df_simple`. Pro nejlepší regresní model, tj. regresor Extra Trees, jsem na těchto datech

Datová sada	# příznaků	klasifikátor	test	komb.
`df_simple`	15	Random Forest	0,8074	0,7570
`df_complex10`	20	AdaBoost	0,6591	0,6514
`df_complex5a`	15	Extra Trees E.	0,5657	0,5906
`df_complex5b`	15	Extra Trees E.	0,4626	0,4484
`df_hist`	15	Random Forest	0,5001	0,5952
`df_complex`	20	Random Forest	0,5531	0,5797
`df_complex_longest`	20	AdaBoost	0,4762	0,49151
`df_all`	20	Random Forest	0,8673	0,7896

Tabulka 3.3: *Accuracy* identifikace webových stránek pro různé datové sady.

dosáhl hodnot RMSE 5,836 a MAE 3,056.

3.4 Výsledky a zhodnocení experimentů

Tato podkapitola se věnuje dosaženým výsledkům v různých experimentech, pro různé datové sady a různé použité modely.

Tabulka 3.3 ukazuje dosaženou *accuracy* identifikace webových stránek pro různé datové sady. *Accuracy* v tabulce je vždy pro nejlepší model, ze všech použitých modelů pro konkrétní datovou sadu. Modely byly vytvořeny stejným způsobem jako model popsaný v sekci 3.3.3, na základě hodnot uvedených v příloze D. Čtvrtý sloupec této tabulky představuje *accuracy* dosaženou na testovací podmnožině dané datové sady. Poslední sloupec pak obsahuje dosaženou *accuracy* pro kombinaci detekce neznámých webových stránek a klasifikaci těch o kterých je rozhodnuto, že jsou známé, pro toto byla použita kombinace nejlepšího klasifikátoru (pro daný dataset) a prahovací funkce. Počet použitých příznaků je ve sloupci druhém, tento počet byl určen pro každou datovou sadu na základě výstupu metody RFECV. Ve sloupci třetím je uveden název klasifikátoru, který dosáhl nejlepší *accuracy* pro danou datovou sadu a jehož výsledky jsou uvedeny v posledních dvou sloupcích.

Jak je vidět v tabulce 3.3, identifikace webových stránek pomocí datové sady `df_simple` dává poměrně dobré výsledky. Datové sady se složitějšími příznaky též nedosahují špatných výsledků, když si uvědomíme, do kolika tříd je klasifikováno, ale nedosahují tak dobrých výsledků, jako jednoduché příznaky v datové sadě `df_simple`. Z datasetů `df_complex10`, `df_complex5a` a `df_complex5b` si nejlépe vedl dataset `df_complex10`, lze tedy usuzovat, že dělení do pěti intervalů při tvorbě markovských příznaků je příliš hrubé, protože `df_complex10`, který používá dělení do deseti intervalů, dosáhl lepší *accuracy*, než ostatní dva datasety. Pro datovou sadu `df_complex5a` bylo dosaženo lepších výsledků než pro `df_complex5b`. Rozdíl mezi těmito sadami je v tom, že při tvorbě sady `df_complex5a` byla nastavena maximální a minimální hodnota pro tvorbu intervalů, to znamená, že velké a malé hodnoty

přesahující nastavenou maximální, respektive minimální, hodnotu spadaly do prvního, resp. posledního, intervalu a díky tomu, hodnoty s menší absolutní hodnotu, byly rozděleny do intervalů jemněji, což jak se zdá, výsledkům klasifikace prospívá. Překvapivě dobře dopadla datová sada `df_simple`, která obsahuje pouze příznaky vzniklé na základě histogramů. Jak lze vidět z porovnaní výsledku datových sad `df_simple` a `df_all`, kde `df_all` je sjednocením příznaků datové sady `df_simple` a `df_complex`, přidané složitější příznaky, sice přinášejí zlepšení (pro některé klasifikátory), ale zlepšení oproti `df_simple` není příliš výrazné. Celkově se nejlepší *accuracy* podařilo dosáhnout s datovou sadou `df_all` a klasifikátorem Random Forest. V tabulce 3.4 je však vidět, že ne všechny klasifikátory dosáhly nejlepších výsledků na datové sadě `df_all`. S datasetem `df_all` bylo také dosaženo nejlepších výsledků detekce neznámých webových stránek, konkrétně bylo dosaženo *accuracy* 0,8021, při kombinaci detekce neznámých stránek a klasifikace webových stránek pak bylo, také na datasetu `df_all`, dosaženo *accuracy* 0,7896. V případě datové sady `df_all`, kde jsem použil 20 příznaku, byly vybrány příznaky:

- `total_pkts_rec`,
- `total_bytes_sent`,
- `total_bytes_rec`,
- `bytes_in_out_ratio`,
- `max_flow_len`,
- `mean_flow_len`,
- `min_burst_size`,
- `min_burst_size_rec`,
- `max_burst_size_rec`,
- `std_burst_size_rec`,
- `bursts_in_out_ratio`,
- `traffic_0`,
- `traffic_10`,
- `traffic_12`,
- `traffic_burst_21`,
- `traffic_burst_22`,
- `traffic_burst_23`,
- `t_hist_0`,
- `t_hist_2`,
- `burst_hist_2`.

Jak lze z výčtu výše vidět, přibližně polovina příznaků byla vybrána ze skupiny jednoduchých příznaků a přibližně polovina ze skupiny složitějších příznaků. Také si můžeme všimnout, že žádný z příznaků není založen na mezipaketových časech, zdá se tedy, že mezipaketové časy mají pro identifikaci webových stránek malý význam.

V tabulce 3.4 jsou ukázány výsledky dosažené na vybraných datových sadách jednotlivými klasifikátory. *Accuracy* zaznamenaná v této tabulce byla dosažena na testovací podmnožině dané datové sady. Počet použitých příznaků pro konkrétní datovou sadu je shodný s počtem uvedeným v tabulce 3.3. Z tabulek 3.3 a 3.4 je zřejmé, že nejlepších výsledků dosahují klasifikátory Ran-

Klasifikátor	df_simple	df_complex	df_all
Extra Trees Ensemble	0,7651	0,4836	0,8145
Random Forest	0,8074	0,5531	0,8673
KNN	0,5205	0,3118	0,4923
Multi-layer Perceptron	0,6851	0,4367	0,5661
AdaBoost	0,7978	0,5451	0,8349
Logistic Regression	0,3267	0,3785	0,2842
Lin. Discriminant Analysis	0,2092	0,3262	0,2672

Tabulka 3.4: Přehled nejlepší dosažené *accuracy* různých klasifikátorů pro vybrané datové sady.

dom Forest, Extra Trees Ensemble a AdaBoost, jejichž pořadí se pro různé datové sady liší, ale jejich dosažená *accuracy* se vzájemně většinou liší jen málo a zároveň většinou výrazně překonávají klasifikátory ostatní.

Co se týče datasetů `df_complex10`, `df_complex5a` a `df_complex5b`, tak v případě odhadu počtu obrázků na webové stránce je situace poněkud opačná oproti identifikaci webových stránek, v případě odhadu počtu obrázků totiž na datové sadě `df_complex5a` nebylo dosaženo lepšího výsledku než na datasetu `df_complex5b`. Také `df_complex10` nepřekonala výsledky `df_complex5a` a `df_complex5b`, kromě případu, kdy bylo na testovací podmnožině datové sady `df_complex10` dosaženo mírně lepšího výsledku, než na testovací podmnožině `df_complex5a`.

Tabulka 3.5 obsahuje výsledky dosažené při odhadu počtu obrázků na webové stránce pro různé datové sady. Výsledky jsou uvedeny jak jako RMSE, tak jako MAE. Odhad počtu obrázků byl pro každou datovou sadu proveden na dvou podmnožinách, na testovací podmnožině a na podmnožině, která obsahuje pouze webové stránky, které nebyly modely viděny během jejich trénování. Pro každou datovou sadu je v této tabulce uveden výsledek dosažený nejlepším modelem pro konkrétní dataset. Jak lze vidět, tak nejlepší výsledky jsou opět dosaženy pro datovou sadu `df_all` a opět je vidět, že dobrý výsledek dosažený na datové sadě `df_simple` je rozšířením o složitější příznaky v datové sadě `df_all` zlepšen nepříliš významně. Pro téměř všechny datové sady byl nejlepší výsledek dosažen pomocí regresního modelu Extra Trees, velmi dobrých výsledků dosahoval také Random Forest regresor.

Úspěšnost odhadu počtu obrázků na webové stránce pomocí různých regresních modelů pro vybrané datové sady je v tabulce 3.6. Tato tabulka zachycuje úspěšnost odhadu počtu obrázků na testovací podmnožině vybraných datových sad, jako RMSE a MAE.

	testovací podmnožina		neviděná podmnožina	
Datová sada	RMSE	MAE	RMSE	MAE
`df_simple`	4,5114	1,9181	5,8363	3,0562
`df_complex10`	8,2424	4,1011	10,7991	5,5746
`df_complex5a`	8,4545	4,3418	9,7986	4,9055
`df_complex5b`	6,4007	3,0525	7,3958	3,3823
`df_hist`	5,8327	2,8079	6,3591	3,4752
`df_complex`	5,5314	2,4913	6,2254	3,1410
`df_complex_longest`	5,5518	2,5531	6,9663	3,6315
`df_all`	4,2761	1,9068	5,5445	2,9146

Tabulka 3.5: Úspěšnost odhadu počtu obrázků na webové stránce pro různé datové sady.

	`df_simple`		`df_complex`		`df_all`	
Regresor	RMSE	MAE	RMSE	MAE	RMSE	MAE
Linear Reg.	9,962	5,402	8,097	4,654	7,970	4,484
Ridge Reg.	9,956	5,410	8,097	4,650	7,969	4,481
SGD Reg.	11,100	5,309	8,113	4,732	8,011	4,606
ElasticNet Reg.	10,289	5,744	8,194	4,773	8,121	4,681
Random Forest Reg.	4,577	1,938	5,667	2,514	4,505	1,930
Extra Trees Reg.	4,511	1,918	5,531	2,491	4,276	1,907
AdaBoost Reg.	8,533	5,753	8,192	4,676	4,351	1,927

Tabulka 3.6: Úspěšnost odhadu počtu obrázků na webové stránce pomocí různých regresních modelů pro vybrané datové sady.

Závěr

Cílem této práce bylo pokusit se zjistit, zda je možné z šifrovaného síťového provozu identifikovat webovou stránku a zde je možné šifrovaného provozu odhadnout strukturu webové stránky.

Po provedení analýzy v první části práce, jsem v druhé části navrhl a implementoval generátor datových sad. Pomocí tohoto generátoru jsem vytvořil datové sady, jejichž zpracování jsem se věnoval ve třetí části. Byly vytvořeny dvě datové sady, každá obsahující více než třicet tisíc komunikací. Tento generátor lze samozřejmě použít pro vytvoření dalších datových sad. Díky univerzálnosti tohoto generátoru lze generovat rozmanité datové sady.

Ve třetí kapitole jsem pak datové sady zpracoval a vytvořil modely strojového učení pro splnění stanovených cílů.

Ověřil jsem, že z šifrovaného provozu lze přibližně identifikovat webovou stránku. Podařilo se mi dosáhnout více než osmdesáti procentní *accuracy* identifikace webové stránky na datové sadě s více než devíti sty unikátním webovými stránkami. Přičemž všechny stránky byly na jednom a tom samém webu, což je vlastně složitější úloha, než kdyby se jednalo o webové stránky v rámci různých webů. Zjistil jsem, že je vytvořený model schopný s přibližně osmdesáti procentní *accuracy* rozhodnout, zda byla šifrovaná síťová komunikace vyvolána načtením webové stránky, kterou model zná či nezná.

Dále jsem zjistil, že jsem z šifrované komunikace s pomocí vytvořeného regresního modelu schopen odhadnout počet obrázků na webové stránce s chybou (MAE) přibližně 1,5 obrázku. Pro webové stránky, které nikdy nebyly modelem viděny (tj. stránky, které vůbec nebyly v trénovací množině) se mi podařilo dosáhnout chyby (MAE) méně než 3 obrázky.

Z těchto zjištění vyplývá, že i přes to, že je komunikace šifrována, lze poměrně dobře z komunikace určit, jaká webová stránka byla navštívena. A také, že je možné, překvapivě přesně určit některé atributy obsahu webových stránek, jako například počet obrázků na webové stránce.

Tato práce nabízí další možnosti rozvoje. Všechny použité metody by jistě bylo možné dále rozvíjet a zlepšovat. Bylo by vhodné dále prozkoumat, jaké

jsou možnosti identifikace webových stránek a odhadu jejich obsahu v případě, kdy je cache webových prohlížečů aktivní. Také by bylo možné, díky implementovanému generátoru dat, rozšířit stávající datové sady či vytvořit další datové sady, například datovou sadu, která by byla vytvořena na základě webových stránek více různých webů. Další možností by bylo, pokusit se zaměřit na jiný druh šifrované síťové komunikace, než je webová komunikace, kterou se zabývala tato práce.

Literatura

[1] DENNIS, Michael Aaron a Robert KAHN. Internet. *Encyclopædia Britannica* [online]. Chicago: Encyclopædia Britannica, 2020 [cit. 2020-07-08]. Dostupné z: `https://www.britannica.com/technology/Internet`

[2] Statistics. *ITU: Committed to connecting the world* [online]. Geneva: International Telecommunication Union, 2019 [cit. 2020-07-08]. Dostupné z: `https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx`

[3] KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5., aktualiz. vyd. Brno: Computer Press, 2008. ISBN 978-80-251-2236-5.

[4] Let's Encrypt Stats: Percentage of Web Pages Loaded by Firefox Using HTTPS. *Let's Encrypt - Free SSL/TLS Certificates* [online]. San Francisco: Internet Security Research Group, 2020 [cit. 2020-07-12]. Dostupné z: `https://letsencrypt.org/stats/`

[5] OSI model. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-07-28]. Dostupné z: `https://en.wikipedia.org/wiki/OSI_model`

[6] *Joy: A package for capturing and analyzing network data features* [online]. 2018. San Jose: Cisco Systems, 2018 [cit. 2020-07-29]. Dostupné z: `https://github.com/cisco/joy/blob/master/doc/using-joy-05.pdf`

[7] QIXIANG, Sun, Yi-min WANG, W. RUSSELL, V.N. PADMANABHAN a Lili QIU. Statistical identification of encrypted Web browsing traffic. *Proceedings 2002 IEEE Symposium on Security and Privacy* [online]. IEEE Comput. Soc, 2002, , 19-30 [cit. 2020-07-29]. DOI: 10.1109/SECPRI.2002.1004359. ISBN 0-7695-1543-6. Dostupné z: `http://ieeexplore.ieee.org/document/1004359/`

[8] AJAEIYA, Georgi, Imad H. ELHAJJ, Ali CHEHAB, Ayman KAYSSI a Marc KNEPPERS. Mobile Apps identification based on network flows. *Knowledge and Information Systems* [online]. 2018, **55**(3), 771-796 [cit. 2020-07-29]. DOI: 10.1007/s10115-017-1111-8. ISSN 0219-1377. Dostupné z: http://link.springer.com/10.1007/s10115-017-1111-8

[9] ANDERSON, Blake, Subharthi PAUL a David MCGREW. Deciphering malware-s use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques* [online]. 2018, **14**(3), 195-211 [cit. 2020-07-14]. DOI: 10.1007/s11416-017-0306-6. ISSN 2263-8733. Dostupné z: http://link.springer.com/10.1007/s11416-017-0306-6

[10] CHEN, Shuo, Rui WANG, XiaoFeng WANG a Kehuan ZHANG. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. *2010 IEEE Symposium on Security and Privacy* [online]. IEEE, 2010, 2010, , 191-206 [cit. 2020-07-29]. DOI: 10.1109/SP.2010.20. ISBN 978-1-4244-6894-2. Dostupné z: http://ieeexplore.ieee.org/document/5504714/

[11] LEROUX, Sam, Steven BOHEZ, Pieter-Jan MAENHAUT, Nathan MEHEUS, Pieter SIMOENS a Bart DHOEDT. Fingerprinting encrypted network traffic types using machine learning. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium* [online]. IEEE, 2018, 2018, , 1-5 [cit. 2020-07-29]. DOI: 10.1109/NOMS.2018.8406218. ISBN 978-1-5386-3416-5. Dostupné z: https://ieeexplore.ieee.org/document/8406218/

[12] DANEZIS, George. *Traffic Analysis of the HTTP Protocol over TLS* [online]. Cambridge: University of Cambridge Computer Laboratory [cit. 2020-07-29]. Dostupné z: https://www.semanticscholar.org/paper/Traffic-Analysis-of-the-HTTP-Protocol-over-TLS-Danezis/9d759184cdc524624fe551b9fc15de9a4cd199fa

[13] TAN, Chang Wei, François PETITJEAN, Eamonn KEOGH a Geoffrey WEBB. *Time series classification for varying length series* [online]. 2019, **2019** [cit. 2020-07-29]. Dostupné z: https://arxiv.org/abs/1910.04341

[14] NIELSEN, Michael. Using neural nets to recognize handwritten digits. *Neural Networks and Deep Learning* [online]. Determination Press, 2015 [cit. 2020-07-29]. Dostupné z: http://neuralnetworksanddeeplearning.com/chap1.html

[15] HAYES, Bob. Programming Languages Most Used and Recommended by Data Scientists. *Business Over Broadway* [online]. Seattle: B.O.B., 2019 [cit. 2020-07-29]. Dostupné z: https://businessoverbroadway.com/2019/01/13/programming-languages-most-used-and-recommended-by-data-scientists/

[16] BAND, Amey. Multi-class Classification - One-vs-All & One-vs-One. *Towards Data Science* [online]. 2020 [cit. 2020-07-30]. Dostupné z: https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b

[17] GEURTS, Pierre, Damien ERNST a Louis WEHENKEL. Extremely randomized trees. *Machine Learning* [online]. 2006, **63**(1), 3-42 [cit. 2020-07-30]. DOI: 10.1007/s10994-006-6226-1. ISSN 0885-6125. Dostupné z: http://link.springer.com/10.1007/s10994-006-6226-1

[18] TOLPYGO, Alexander. Time-Series Analysis: Wearable Devices using DTW and kNN. *SFL Scientific* [online]. Boston: SFL Scientific, LLC., 2017 [cit. 2020-07-29]. Dostupné z: https://sflscientific.com/data-science-blog/2016/6/4/time-series-analysis-fitbit-using-dtw-and-knn

[19] ANDĚL, Jiří. *Statistické metody*. Vyd. 3. Praha: Matfyzpress, 2003. ISBN 80-867-3208-8.

[20] Components. *The Selenium Browser Automation Project* [online]. Software Freedom Conservancy, 2020 [cit. 2020-07-14]. Dostupné z: https://www.selenium.dev/documentation/en/grid/grid_3/components_of_a_grid/

[21] HTTP: The Definitive Guide: Parallel Connections. *O'Reilly Media - Technology and Business Training* [online]. Sebastopol: O'Reilly Media, 2020 [cit. 2020-07-29]. Dostupné z: https://www.oreilly.com/library/view/http-the-definitive/1565925092/ch04s04.html

[22] Libpcap File Format. *The Wireshark Wiki* [online]. 2015 [cit. 2020-07-29]. Dostupné z: https://wiki.wireshark.org/Development/LibpcapFileFormat

[23] Virtual Networking: Host-Only Networking. *Oracle VM VirtualBox* [online]. Redwood Shores: Oracle Corporation, 2020 [cit. 2020-07-29]. Dostupné z: https://www.virtualbox.org/manual/ch06.html

[24] *Python* [online]. Beaverton: Python Software Foundation, c1990-2014 [cit. 2020-07-29]. Dostupné z: https://www.python.org/

[25] Jupyter. *Project Jupyter* [online]. Project Jupyter, c2020 [cit. 2020-07-29]. Dostupné z: https://jupyter.org/

[26] Chapter 8: Markov Chains. *Stochastic Processes* [online]. Auckland: University of Auckland, 2014, s. 149-173 [cit. 2020-07-30]. Dostupné z: https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf

[27] Multiclass and multilabel algorithms. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/multiclass.html

[28] API Reference. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-09-17]. Dostupné z: https://scikit-learn.org/stable/modules/classes.html

[29] Pearsonův korelační koeficient. *Matematická biologie učebnice* [online]. Brno: Masarykova univerzita [cit. 2020-07-30]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=aplikovana-analyza-klinickych-a-biologickych-dat--analyza-a-management-dat-pro-zdravotnicke-obory--zaklady-korelacni-analyzy--pearsonuv-korelacni-koeficient

[30] Spearmanův korelační koeficient. *Matematická biologie učebnice* [online]. Brno: Masarykova univerzita [cit. 2020-07-30]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=aplikovana-analyza-klinickych-a-biologickych-dat--analyza-a-management-dat-pro-zdravotnicke-obory--zaklady-korelacni-analyzy--spearmanuv-korelacni-koeficient

[31] Rfecv. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html

[32] Recursive feature elimination with cross-validation. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html

[33] CEBALLOS, Frank. Model Design and Selection with Scikit-learn. *Towards Data Science* [online]. 2019 [cit. 2020-07-30]. Dostupné z: https://towardsdatascience.com/model-design-and-selection-with-scikit-learn-18a29041d02a

[34] GridSearchCV. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[35] RandomizedSearchCV. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[36] ClassificationReport. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[37] Pipeline. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[38] StandardScaler. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[39] VarianceThreshold. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

[40] LocalOutlierFactor. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

[41] Novelty detection with Local Outlier Factor. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_novelty_detection.html

[42] IsolationForest. *Scikit-learn* [online]. scikit-learn developers, c2007-2019 [cit. 2020-07-30]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[43] *Pandas* [online]. The pandas development team [cit. 2020-09-17]. Dostupné z: https://pandas.pydata.org/

[44] *Matplotlib* [online]. The Matplotlib development team, c202-2020 [cit. 2020-09-17]. Dostupné z: https://matplotlib.org/

[45] *Seaborn* [online]. Michael Waskom, c2012-2020 [cit. 2020-09-17]. Dostupné z: https://seaborn.pydata.org/

Příloha **A**

Seznam použitých zkratek

CSS Cascading Style Sheets

CSV Comma-separated values

DNS Domain Name System

DTW Dynamic time warping

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IP Internet Protocol

ISO International Organization for Standardization

JSON JavaScript Object Notation

KNN K Nearest Neighbors

MAE Mean Absolute Error

OSI Open Systems Interconnection

PCAP Packet Capture

RFECV Recursive Feature Elimination with Cross-Validation

RMSE Root Mean Square Error

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

TOR The Onion Router

UDP User Datagram Protocol

URL Uniform Resource Locator

VoIP Voice over Internet Protocol

WWW World Wide Web

Příloha B

Ukázka dat vygenerovaných generátorem datových sad

B.1 Soubor metadat

```
1 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!Action_pact!", "query_num": 0, "media":
  ↪ {"num_images": 3}, "scripts": {"num_sripts": "6"}, "session_id": 0}
2 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!Action_pact!", "query_num": 1, "media":
  ↪ {"num_images": 3}, "scripts": {"num_sripts": "6"}, "session_id": 0}
3 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!Action_pact!", "query_num": 2, "media":
  ↪ {"num_images": 3}, "scripts": {"num_sripts": "6"}, "session_id": 0}
4 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!Arriba!", "query_num": 3, "media":
  ↪ {"num_images": 3}, "scripts": {"num_sripts": "6"}, "session_id": 0}
5 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!Arriba!", "query_num": 4, "media":
  ↪ {"num_images": 3}, "scripts": {"num_sripts": "6"}, "session_id": 0}
6 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!Arriba!", "query_num": 5, "media":
  ↪ {"num_images": 3}, "scripts": {"num_sripts": "6"}, "session_id": 0}
7 {"session_name": "linux_firefox", "url":
  ↪ "https://en.wikipedia.org/wiki/!BANG!", "query_num": 6, "media":
  ↪ {"num_images": 2}, "scripts": {"num_sripts": "6"}, "session_id": 0}
```

Zdrojový kód B.1: Ukázka souboru `META_0_0_linux_firefox`. Jsou zde vidět metadata uložená pro prvních sedm provedených dotazů.

B.2 Soubor zachycené a transformované komunikace

```
1 {"version":"4.5.0", "interface":"none", "promisc":0,
  ↪ "output":"out/0_0_linux_firefox_6.gz", "outputdir":".", "username":"none",
  ↪ "info":"none", "count":0, "upload":"none", "keyfile":"none", "retain":0,
  ↪ "bidir":1, "num_pkts":50, "zeros":0, "retrans":0, "dist":0,
  ↪ "cdist":"none", "entropy":0, "hd":0, "classify":0, "idp":0, "exe":0,
  ↪ "anon":"none", "useranon":"none", "bpf":"none", "verbosity":4,
  ↪ "threads":1, "updater":0, "wht":0, "example":0, "dns":1, "ssh":0, "tls":1,
  ↪ "dhcp":0, "dhcpv6":0, "http":1, "ike":0, "payload":0, "salt":0, "ppi":0,
  ↪ "fpx":0, "end-config":1}
2 {"sa":"192.168.57.222", "da":"91.198.174.192", "pr":6, "sp":41100, "dp":443,
  ↪ "bytes_out":3259, "num_pkts_out":73, "bytes_in":13672, "num_pkts_in":3,
  ↪ "time_start":1590569606.840233, "time_end":1590569608.714130,
  ↪ "packets":[{"b":75, "dir":">", "ipt":0},{"b":172, "dir":"<",
  ↪ "ipt":388},{"b":1428, "dir":"<", "ipt":202},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":1},{"b":152, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":359, "dir":"<", "ipt":0},{"b":288, "dir":">",
  ↪ "ipt":23},{"b":113, "dir":">", "ipt":0},{"b":110, "dir":">",
  ↪ "ipt":0},{"b":89, "dir":">", "ipt":0},{"b":96, "dir":">",
  ↪ "ipt":3},{"b":1428, "dir":"<", "ipt":71},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":361, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":2},{"b":697, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1349, "dir":"<", "ipt":0},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":277, "dir":"<", "ipt":0},{"b":167, "dir":">",
  ↪ "ipt":69},{"b":238, "dir":">", "ipt":6},{"b":738, "dir":">",
  ↪ "ipt":5},{"b":318, "dir":">", "ipt":27},{"b":117, "dir":">",
  ↪ "ipt":4},{"b":125, "dir":">", "ipt":4},{"b":117, "dir":">",
  ↪ "ipt":3},{"b":113, "dir":">", "ipt":4},{"b":92, "dir":">",
  ↪ "ipt":3},{"b":611, "dir":"<", "ipt":69},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":1},{"b":1428, "dir":"<",
  ↪ "ipt":0},{"b":1428, "dir":"<", "ipt":0},{"b":86, "dir":">",
  ↪ "ipt":129},{"b":83, "dir":">", "ipt":1},{"b":119, "dir":">",
  ↪ "ipt":769},{"b":175, "dir":">", "ipt":36}], "ip":{"out":{"ttl":64,
  ↪ "id":[12384, 12385, 12386, 12387, 12388, 12389, 12390, 12391, 12392,
  ↪ 12393, 12394, ]}, "in":{"ttl":52, "id":[65380, 65381, 65382, 65383, 65384,
  ↪ 65385, 65386, 65387, 65388, 65389, ]}}}
3 {"sa":"192.168.57.222", "da":"216.58.201.67", "pr":6, "sp":59646, "dp":443,
  ↪ "bytes_out":0, "num_pkts_out":1, "bytes_in":0, "num_pkts_in":1,
  ↪ "time_start":1590569610.650196, "time_end":1590569610.663474,
  ↪ "packets":[], "ip":{"out":{"ttl":64, "id":[40203]}, "in":{"ttl":120,
  ↪ "id":[35518]}}}
```

Zdrojový kód B.2: Ukázka souboru `0_0_linux_firefox_6`. Tento soubor obsahuje dva orientované toky příznaků zachycené generátorem datových sad. Je to soubor, ke kterému jsou metadata na sedmém řádku předchozí ukázky B.1.

Příloha C

Ukázka transformovaného datasetu

```
group_id;session_id;os;browser;query_num;url;num_images;num_scripts;bytes_out;num_pkts_out;bytes_in;num_pkts_in;
↪  duration;traffic;inter_pkt_times
0;0;linux;firefox;0;https://en.wikipedia.org/wiki/138;3;6;[0, 0, 0, 4222, 1051, 808];[2, 2, 2, 107, 13, 12];[0, 0, 0, 15402,
↪  5525, 4522];[2, 2, 2, 6, 10, 11];[9.690688848495483, 9.871349096298218, 9.871723890304565, 0.6506810188293457,
↪  0.05282902717590332, 0.04787588119506836];[[],[],[],[517, -1448, -1448, -1200, -99, 80, 170, 253, -303, -43, -31, 31,
↪  -1448, -1448, -1448, -1448, -1448, -1345, 320, 117, 109, 101, 95, -1448, -1448, -1448, -1448, -1448, -573, -1224, -1448,
↪  -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448,
↪  -1448, -1448, -1448, -1448, -28, 265, 116, 116, 112, 91, -1448, -1448, -1448, -155, -667, -1448, -1448, 166, 238, 728, 85,
↪  82, 156, 155, 119],[517, -1448, -1448, -1300, 80, 170, 253, -303, -43, -31, 31, -952],[517, -1448, -1448, -1200, -99, 80,
↪  187, -303, -24, 24]];[[],[],[],[15, -14, 0, 0, 0, 3, 0, 0, -13, 0, 0, 0, -16, 0, 0, 0, 0, 0, 112, 0, 0, 1, 0, -12, 0, 0,
↪  0, -2, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, -1, 0, 0, 0, -7, 0, 0, 0, -1, 0, 0, 37, 3, 0, 0, 0, -10, 0, 0, 0, -3,
↪  0, 0, 42, 5, 0, 113, 0, 123, 75, 5],[16, -14, 0, -1, 4, 0, 0, -13, 0, 0, 0, 0],[15, -14, 0, 0, 0, 2, 0, -14, 0, 0]]
0;0;linux;firefox;1;https://en.wikipedia.org/wiki/138;3;6;[0, 0, 0, 0, 2774, 138, 46, 1043, 13555];[1, 2, 2, 2, 96, 2, 2, 17,
↪  59];[0, 0, 0, 0, 8511, 630, 46, 5562, 15487];[1, 2, 2, 2, 253, 2, 2, 14, 83];[0.0006840229034423828, 9.868415117263794,
↪  9.87765383720398, 9.871001958847046, 0.45357799530029297, 0.014837980270385742, 0.0013971328735351562,
↪  0.05462002754211426, 2.8623130321502686];[[],[],[],[],[71, -1448, -1448, -1448, -1448, -832, -1431, 206, 113, 110, 89, 96,
↪  -1448, -1448, -1448, -1448, -1448, -1413, -1448, -1448, -1448, -1448, -574, -1448, -1448, -1448, -1448, -1448, -1448,
↪  -1448, -1448, -1448, -1448, -1448, -1448, -1448, -1448, -733, -1448, -1448, -1309, 105, 117, 117, 113, 92, -1448, -1448,
↪  -1448, -67, -265, -331, -307, -1448, -994, -1448, -1448, -1448, -1448, -1448, -1448, 167, 239, 729, 86, 83, 122,
↪  119],[138, -630],[46, -46],[517, -1418, -1418, -956, 64, 92, -580, 256, 44, 31, -31, -514, -221, -126, -259, -39,
↪  39],[517, -1418, -1418, -74, 64, 92, 637, 1701, -580, -31, 31, -960, -1418, -657, -205, -39, 39, 71, 1047, -228, -445,
↪  -33, -39, 39, 71, 992, -232, -386, -228, -39, 39, 71, 955, -227, -402, -286, -39, 39, 71, 971, -228, -297, -168, -39, 39,
↪  71, 964, -224, -416, -109, -39, 39, 70, 951, -226, -415, -236, -39, 39, 71, 938, -224, -361, -139, -39, 39, 71, 946, 169,
↪  567, -224, -399, -93, -39, -525, 39, -754, -134, 71, 985, -229, -253, -215, -39, 39]];[[],[],[],[],[0, -16, 0, 0, 0, 0, 0,
↪  37, 0, 0, 0, 0, -14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 1, 0,
↪  -12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 7, 0, 124, 2, 188, 8],[0, -14],[0, -1],[1, -15, 0, 0, 10, 0, 0, 0, 0, 0,
↪  0, -19, 0, 0, 0, 0, 3],[1, -22, 0, 0, 6, 0, 0, 0, 0, 0, 0, -29, 0, 0, 0, 0, 2, 526, 0, -20, 0, 0, 0, 0, 1069, 0, -31, -9,
↪  0, 0, 0, 52, 0, -20, 0, 0, 0, 1, 159, 0, -19, 0, 0, 0, 3, 221, 0, -116, 0, 0, 0, 3, 190, 0, -22, 0, 0, 0, 3, 43, 0, -20,
↪  0, 0, 0, 2, 89, 0, 0, 0, -18, 0, 0, 0, -1, 0, 0, 0, 120, 0, -19, 0, 0, 0, 2]]
```


Zdrojový kód C.1: Ukázka transformace transformovaného datasetu do jednoho CSV souboru. Tato ukázka obsahuje úvodní hlavičku s názvy sloupců, a dva řádky, které obsahují metadata a příznaky dvou komunikací.

Příloha D

Klasifikátory a jejich parametry

Klasifikátor	Parametr	Hodnoty
Extra Trees Ensemble	n_estimators	50, 85, 100, 120, 150, 200
	max_features	auto, sqrt, log2
	max_depth	None, 3, 5, 10, 20, 50, 80, 100, 150
	criterion	gini, entropy
	min_samples_leaf	1, 2, 4
	min_samples_split	2, 5, 10
Random Forest	n_estimators	50, 85, 100, 120, 150, 200
	max_features	auto, sqrt, log2
	max_depth	None, 3, 5, 10, 20, 50, 80, 100, 150
	criterion	gini, entropy
	min_samples_leaf	1, 2, 4
	min_samples_split	2, 5, 10
KNN	n_neighbors	`list(range(1,31))`
	p	1, 2, 3, 4, 5
	leaf_size	5, 10, 15, 20, 25, 30, 35, 40, 50
Multi-layer Perceptron	max_iter	80, 100, 500, 800
	hidden_layer_sizes	(50,50,50), (50,100,50), (100,)
	activation	tanh, relu
	solve	sgd, adam
	alpha	0,0001; 0,05
	learning_rate	constant, adaptive
AdaBoost	base_estimator	None, Dec.TreeClass.(max_depth=d)
	d	10, 50, 100
	n_estimators	50, 80, 100, 120, 150
	learning_rate	0,01; 0,05; 0,1; 0,3; 1
Lin. Discriminant Analysis		
Logistic Regression	C	100, 10, 1.0, 0.1, 0.01
	multi_class	multinomial

Tabulka D.1: Přehled klasifikátorů a hyperparametrů použitých k ladění klasifikátorů.

Příloha E

Regresory a jejich parametry

Klasifikátor	Parametr	Hodnoty
Linear Regression	n_jobs	-1
Ridge Regression	alpha	0,001; 0,01; 0,1; 0,5; 1; 2; 5; 10; 20; 50
SGD Regressor	alpha loss penalty learning_rate	0,001; 0,01; 0,1; 0,5; 1; 2; 5; 10; 20; 50 squared_loss, huber, epsilon_insensitive l2, l1, elasticnet constant, optimal, invscaling
ElasticNet Regressor	max_iter alpha l1_ratio	1, 5, 10, 50, 100 0,001; 0,01; 0,1; 0,5; 1; 2; 5; 10; 20; 50 0; 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9
Random Forest Regressor	n_estimators max_features max_depth min_samples_split min_samples_leaf n_jobs	50, 85, 100, 120, 150, 200 auto, sqrt, log2 None, 3, 5, 10, 20, 50, 80, 100, 150 2, 5, 10 1, 2, 5, 10 -1
Extra Trees Regressor	n_estimators max_features max_depth min_samples_split min_samples_leaf n_jobs	50, 85, 100, 120, 150, 200 auto, sqrt, log2 None, 3, 5, 10, 20, 50, 80, 100, 150 2, 5, 10 1, 2, 5, 10 -1
AdaBoost Regressor	n_estimators learning_rate loss	50, 100 0,01; 0,05; 0,1; 0,3; 0,6; 1 linear, square, exponential

Tabulka E.1: Přehled regresorů a hyperparametrů použitých k jejich ladění.

Příloha F

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `data`
  - `wiki_cache` ..... předzpracovaný dataset s aktivní cache
  - `wiki_nocache` ..... předzpracovaný dataset s deaktivovanou cache
- `src`
  - `impl_generator` .... zdrojové kódy implementace generátoru datasetů
  - `impl_model` ..... zdrojové kódy zpracování datasetů a tvorby modelů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF