UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

KLASIFIKÁCIA MALVÉRU VYUŽITÍM SELEKCIE ATRIBÚTOV

2020 **Bc. Peter CHOMIČ**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

KLASIFIKÁCIA MALVÉRU VYUŽITÍM SELEKCIE ATRIBÚTOV

DIPLOMOVÁ PRÁCA

Študijný program:	Informatika
Pracovisko (katedra/ústav):	Ústav informatiky
Vedúci diplomovej práce:	JUDr. RNDr. Pavol Sokol, PhD.
Konzultant diplomovej práce:	Mgr. Ladislav Bačo

Košice 2020

Bc. Peter CHOMIČ

Univerzita P. J. Šafárika v Košiciach
Prírodovedecká fakulta

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Chomič
Študijný program: Informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: Informatika
Typ záverečnej práce: Diplomová práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Klasifikácia malvéru využitím selekcie atribútov

Názov EN: Malware classification using the feature selection

Cieľ:
1. Vytvorenie dátovej sady pre klasifikáciu malvéru
2. Porovnanie prístupov ku klasifikácii a určovaniu podobnosti malvéru
3. Vytvorenie modelu pre klasifikáciu malvéru využitím selekcie atribútov, jeho implementácia a vyhodnotenie

Literatúra: Literatúra:

[1] Saxe, J., Sanders, H.: Malware data science - attack detection and attribution, San Francisco, No starch press. 2018.
[2] Monnappa, K.A.: Learning malware analysis, Packt. 2018.
[3] UCCI, Daniele; ANIELLO, Leonardo; BALDONI, Roberto. Survey of Machine Learning Techniques for Malware Analysis. Computers & Security, 2018.
[4] YE, Yanfang, et al. A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR), 2017, 50.3: 41.

Vedúci: RNDr. JUDr. Pavol Sokol, PhD.
Konzultant: Mgr. Ladislav Bačo
Oponent: RNDr. Ľubomír Antoni, PhD.
Ústav : ÚINF - Ústav informatiky
Riaditeľ ústavu: RNDr. Ondrej Krídlo, PhD.

Dátum schválenia: 31.03.2020

Abstrakt v štátnom jazyku

V práci sa venujeme multinominálnej klasifikácii škodlivého kódu (malvéru). V rámci výskumnej práce sme si vytvorili vlastný dataset, ktorý je tvorený veľkým množstvom charakteristík vzoriek. Cieľom tejto práce je pomocou algoritmov na selekciu atribútov vylúčiť tie atribúty, ktoré nie sú dôležité pri klasifikácii. Chceme ukázať, že selekcia má pri klasifikácii malvéru kladný dopad na výsledky aj v prípade, keď sa selektuje len veľmi malá časť atribútov. Keďže atribútov je enormne veľké množstvo a mnohé metódy sú výpočtovo náročné, v rámci práce selekciu atribútov delíme na hrubú selekciu a jemnú selekciu. Pri hrubej selekcii používame výpočtovo nenáročné metódy, ktoré vylúčia atribúty, ktoré majú minimálny alebo žiadny vplyv na klasifikáciu. Po nej nasleduje jemná selekcia, pri ktorej porovnávame viacero metód. Výsledky jemnej selekcie porovnávame na viacerých metódach klasifikácie s cieľom nájsť najlepšiu kombináciu klasifikátora a selektora a zistiť, ktoré skupiny atribútov boli použité pri dosiahnutí najlepšieho výsledku.

Kľúčové slová: malvér, selekcia atribútov, klasifikácia

Abstrakt v cudzom jazyku

This thesis deals with multinomial classification of malicious software (malware). We create our own dataset which consists of many static traits of samples. Our goal is to remove unimportant features with algorithms for feature selection. We want to show, that feature selection has positive impact on results of malware classification even when only few features are selected. Because number of features is enormous and many methods are computationally expensive, we divide selection into gross and fine. We use computationally inexpensive methods, removing features with almost no importance for classification in gross selection. After that comes fine selection where we compare several methods. Results of fine classification are compared on several methods of classification with a goal of finding the best combination of classifier and selector and finding out, which groups of features were used to achieve the best result.

Key words: malware, feature selection, classification

Obsah

Úvod

V súčasnosti sa rapídne zväčšuje množstvo zachyteného škodlivého kódu (malvéru). Podľa štúdie [1] urobenej v auguste 2019 spoločnosťou McAfee sa zachytilo v prvom kvartáli 2019 viac ako 65 miliónov vzoriek nového malvéru. Za každý kvartál v rokoch 2017 a 2018 spoločnosť McAfee zachytila viac ako 40 miliónov vzoriek nového malvéru. Celkové množstvo malvéru, ktorý spoločnosť McAfee zachytila, stúplo z takmer 600 miliónov vzoriek v druhom kvartáli 2017 na viac ako 900 miliónov v prvom kvartáli 2019 a blíži sa k miliarde. Ochrana koncových zariadení (napr. antivírusové programy) používa tzv. signatúry, získané z malvéru extrakciou vhodných atribútov, ktoré charakterizujú jeho rodinu. Tento prístup so sebou prináša aj určité nevýhody. Vhodnosť atribútov pre každú rodinu musí posúdiť expert a aj pri určitých malých zmenách sa môže signatúra výrazne zmeniť. Z tohto dôvodu bolo v posledných rokoch navrhnutých niekoľko prístupov na automatickú klasifikáciu malvéru [2].

Pri použití metód strojového učenia pri klasifikácii je potrebné brať do úvahy, že klasifikačné metódy majú vo všeobecnosti problémy s veľmi vysokou dimenzionalitou vstupu [88]. Ak klasifikačný algoritmus dostane na vstupe príliš veľké množstvo atribútov, môže sa stať, že nebude schopný vytvoriť dobre fungujúci model, keďže nebude vedieť správne určiť dôležitosť jednotlivých atribútov a dôjde k overfittingu [4]. Overfitting nastáva vtedy, keď sa model naučil tak, že dáva veľmi dobré výsledky na tréningovej vzorke ale utrpela jeho generalizácia – mimo trénovacieho datasetu dáva veľmi zlé výsledky [65]. Z tohto dôvodu sa pred klasifikáciou pri datasetoch s veľkým množstvom atribútov vykonáva selekcia. Vzorky malvéru predstavujú spustiteľné programy, a preto sú pomerne komplexné a je možné získať z nich enormné množstvo atribútov.

Našim cieľom je ukázať, že selekcia atribútov má pri klasifikácii malvéru kladný dopad na výsledky aj v prípade, keď sa selektuje len veľmi malá časť atribútov. Dataset zložený z malvéru je špecifický tým, že zo vzoriek malvéru je možné získať enormné množstvo atribútov (aj rádovo viac ako miliardu). Z tohto dôvodu je nevyhnutné obmedziť atribúty. K tomuto účelu si vytvárame dataset, ktorý pokrýva množstvo statických skupín atribútov, keďže jedným z našich cieľov je zistiť, či použitie len statických atribútov postačuje pre klasifikáciu malvéru. K tvorbe datasetu patrí aj označenie jeho prvkov (labeling), preto porovnávame viaceré možnosti určovania

podobnosti a označovania malvéru a vybranú metódu potom aplikujeme na označenie datasetu. Na tomto datasete aplikujeme a porovnávame metódy na selekciu atribútov. Pri porovnávaní metód sa zameriavame na výpočtovú a pamäťovú zložitosť algoritmov, ako aj presnosť pri klasifikácii, ktorú nimi selektované atribúty dosiahli. Keďže presnosť klasifikácie vybraných atribútov je hlavný parameter porovnania metód selekcie, pred použitím selekcie porovnávame prístupy ku klasifikácii a vyberáme metódy klasifikácie, na ktorých potom porovnávame metódy selekcie. Výsledky porovnaní metód selekcie ovplyvňuje aj počet atribútov, ktoré sa selektujú, preto robíme pre každú metódu viacero selekcií s rôznym počtom selektovaných atribútov a sledujeme, aký počet atribútov bol najčastejšie potrebný pre optimálny výsledok.

Keďže porovnáme viacero kombinácií metód na klasifikáciu a selekciu, zaujíma nás, ktoré kombinácie klasifikátora s selektora dosahujú konzistentne dobré výsledky. Špeciálne nás zaujíma to, aké výsledky budú dosiahnuté s atribútmi vybranými embedded metódami selekcie (metódy, ktoré využívajú špecifiká štruktúry klasifikátora, vďaka čomu počas učenia klasifikátor vykonáva aj selekciu) pri klasifikovaní iným typom klasifikátora. Tieto kroky sú časťou nášho cieľa tvorby, implementácie a vyhodnotenia modelu na klasifikáciu malvéru využitím selekcie atribútov.

Okrem toho selekciu a následnú klasifikáciu robíme aj osobitne na skupinách atribútov, ktoré majú málo položiek (najprv vyberáme skupiny, ktoré majú menej ako tisíc prvkov po prvotnej selekcii, potom tie, ktoré ich majú menej ako sto). Vďaka tomu chceme zistiť, či sú pre klasifikáciu postačujúce len tieto málo početné skupiny atribútov. Ak by boli postačujúce, tak by nebolo potrebné mnohopočetné skupiny atribútov extrahovať, čo by ušetrilo veľké množstvo času a pamäte pri klasifikácii malvéru. To by umožnilo spúšťať klasifikáciu malvéru aj na zariadeniach s obmedzenými systémovými zdrojmi. Okrem toho porovnávame presnosť klasifikácie pre dataset s čistými vzorkami a dataset, v ktorom by sa mali vyskytovať len zabalené, šifrované a obfuskované vzorky (obfuskácia je snaha urobiť kód ťažšie pochopiteľným). Sledujeme aj to, ktoré skupiny atribútov sú úspešné len na datasete, ktorý neobsahuje čisté vzorky. Chceme zistiť, či je možné so statickými atribútmi dosiahnuť dobré výsledky aj napriek obfuskácii, šifrovaniu a zabaleniu, a ako tieto skutočnosti ovplyvňujú výsledky.

Prácu sme rozdelili na štyri kapitoly. V prvej kapitole popisujeme prehľad súčasného stavu pri klasifikácii malvéru a selekcii jeho atribútov. Sústredíme sa na používané metódy a nimi dosiahnuté výsledky. Kapitola obsahuje aj úvod do

problematiky malvéru. V druhej kapitole popisujeme náš postup z teoretického hľadiska. Vysvetľujeme funkcionalitu metód, ktoré sme použili v jednotlivých etapách, ich časovú a pamäťovú zložitosť. Poukazujeme na rôzne možnosti riešenia etáp a nástroje, ktoré je možné použiť a odôvodňujeme nami vybrané postupy. V tretej kapitole máme popísaný celý postup spracovania vzoriek, extrakcie atribútov, selekcie, klasifikácie a spracovania výsledkov z implementačného hľadiska. Popisujeme praktické problémy, ktorým sme čelili a naše riešenia, prípadne zmeny postupu, ktoré z nich vyplynuli. Vo štvrtej kapitole analyzujeme získané výsledky. Zisťujeme, aký počet atribútov je minimálne potrebný a pri akom počte sa zastaví zvyšovanie presnosti klasifikácie pri zvýšení počtu atribútov. Porovnávame viacero algoritmov na niekoľkých datasetoch a viacerých prahoch selekcie pre viaceré skupiny klasifikačných algoritmov. Okrem výsledkov klasifikácií empiricky porovnávame aj časy behu algoritmov a množstvo použitej operačnej pamäte. Z výsledkov potom vyvodzujeme závery.

1 Selekcia atribútov pre klasifikáciu malvéru

V rámci tejto kapitoly sa zameriame na podobné, resp. súvisiace práce. Vzhľadom na ciele práce a navrhnutý metodologický postup sme sa rozhodli prehľad súčasného stavu rozdeliť na práce venujúce sa selekcii atribútov pre klasifikáciu malvéru, klasifikácii malvéru, metrikám klasifikácie malvéru a označovaniu (labeling).

1.1 Úvod do problematiky malvéru

Malvér je kód, ktorý vykonáva škodlivú činnosť. Môže byť vo forme spustiteľného programu, kódu, skriptu, alebo iného softvéru. Útočníci využívajú malvér na krádež citlivých informácií, sledovanie postihnutého systému alebo na prebratie kontroly nad systémom. Malvér možno podľa funkcionality a spôsobu útoku klasifikovať do nasledujúcich skupín [171]:

- Vírus a červ: Malvér, ktorý sa dokáže replikovať a šíriť sa do ďalších počítačov. Vírus potrebuje súčinnosť používateľa, červ ho nepotrebuje.
- Trójsky kôň: Malvér, ktorý sa maskuje ako legitímny program s cieľom nalákať používateľa k jeho inštalácii.
- Backdoor / Trójsky kôň so vzdialeným prístupom (RAT): Typ trójskeho koňa, ktorý umožňuje útočníkovi vzdialený prístup do systému s možnosťou spúšťania príkazov.
- Advér: Malvér, ktorý zobrazuje používateľovi nechcené reklamy.
- Botnet: Skupina počítačov (tzv. boti) infikovaná rovnakým malvérom, ktorý komunikuje s riadiacim (command-and-control) serverom, riadeným útočníkom. Útočník dokáže botom posielať príkazy, pomocou ktorých môže vykonať škodlivé činnosti ako je napríklad zamietnutie služby (denial of service - DDOS) alebo posielanie spamu.
- Zlodej informácií (information stealer): Malvér, vytvorený na krádež citlivých dát, napríklad prístupových údajov do banky alebo stlačených tlačidiel na klávesnici. Príklady takého malvéru zahŕňajú: key logger, spyware, sniffer, a form grabber.
- Ransomvér: malvér, ktorý zašifruje súbory a za ich odšifrovanie je požadované výkupné.

- Rootkit: malvér, ktorý umožňuje útočníkovi privilegovaný prístup na infikovaný systém a má schopnosť skryť seba alebo iný softvér.
- Downloader alebo dropper: malvér, ktorý sťahuje alebo inštaluje ďalší malvér.

1.1.1 Analýza malvéru

Analýza malvéru je štúdium správania sa malvéru, ktoré zahŕňa analýzu spustiteľného súboru v bezpečnom prostredí. Cieľom analýzy je detegovať malvér, zistiť jeho schopnosti, pochopiť ako pracuje a zastaviť ho. Analýzu možno rozdeliť do nasledujúcich techník [171]:

- statická analýza,
- dynamická analýza,
- analýza kódu,
- analýza pamäte.

Statická analýza je proces analýzy spustiteľného súboru bez jeho spustenia. Používa sa ako prvotná analýza. Pomocou statickej analýzy možno zistiť typ súboru, získať použité textové reťazce, metadáta, importované a exportované funkcie. Okrem toho možno preskúmať celý súbor, ktorý je rozdelený na sekcie. Súčasťou analýzy môže byť skenovanie antivírusovými programami. Útočníci na ochranu pred statickou analýzou používajú balíčkovací a šifrovací softvér. Pri zabalení sa súbor komprimuje a výstupom je nový komprimovaný spustiteľný súbor, ktorý pri spustení najprv vykoná proces opačný ku komprimácii a potom spustí pôvodný súbor. Pri šifrovaní je proces podobný, ale namiesto kompresie sa používa šifrovacia funkcia a namiesto dekompresie je dešifrovacia funkcia.

Dynamická analýza je proces spustenia súboru v izolovanom prostredí a monitorovanie jeho správania sa a interakcie so systémom. Zvyčajne sa monitorujú nasledujúce časti systému [171]:

- Procesy: Pri spustení programu sa vytvorí jeho proces, ktorý môže spúšťať ďalšie procesy. Monitorovať je potrebné aj ostatné spustené procesy, do ktorých môže malvér vložiť škodlivý kód (code injection) alebo dynamicky linkovanú knižnicu.

- Súborový systém: Monitoruje sa, ktoré súbory malvér čítal, vytváral alebo do nich zapisoval.
- Registre: Monitorovanie kľúčov registrov, ku ktorým malvér pristupoval, alebo ich modifikoval a dát, ktoré zapisoval alebo mazal z registrov.
- Sieť: Monitorovanie sieťovej prevádzky z a do systému, na ktorom je spustený malvér. Kvôli bezpečnosti sa používa iba lokálna sieť na virtuálnom stroji (host-only network) a prístup k Internetu a ostatným sieťovým službám sa simuluje.

Útočníci používajú rôzne techniky na to, aby zistili, či sa malvér nachádza vo virtuálnom stroji, alebo simulovanej sieti. Ak malvér deteguje virtuálne prostredie alebo simuláciu Internetu, môže zmeniť svoje správanie a nevykonať škodlivú činnosť alebo ukončiť svoju činnosť.

Analýza kódu sa delí na statickú a dynamickú. Statická analýza kódu zahŕňa disasemblovanie spustiteľného súboru. Tým sa zo strojového kódu získa kód v assembly jazyku a následne sa skúma kód s cieľom porozumieť jeho funkcionalite. Disasemblovanie môže byť lineárne alebo nelineárne. Pri lineárnom disasemblovaní sa strojový kód prekladá na kód v assembly jazyku v takom poradí, v akom je zapísaný v súbore. Toto ale nemusí odrážať poradie, v akom sú časti kódu spustené pri behu programu. Z tohto dôvodu sa používa aj pokročilejšie nelineárne, prúdovo orientované (flow oriented) disasemblovanie [4]. Nelineárny disasembler sa snaží simulovať spustenie kódu, aby bolo možné zistiť, ktoré inštrukcie je možné dosiahnuť ako výsledok série podmienených vetvení [4]. Okrem disasemblera je možné použiť aj dekompilér, ktorý prekladá strojový kód do vysokoúrovňového jazyka (pseudokód) [171].

Útočníci sa snažia na obranu proti disasemblovaniu obfuskovať činnosti malvéru. Príkladom obfuskačnej techniky je modifikujúci sa kód – program, ktorý modifikuje sám seba po spustení. Iný spôsob ako skryť činnosť malvéru pred disasemblerom je dynamické sťahovanie kódu. Útočník vtedy šíri jednoduchý program, ktorý po spustení stiahne samotný malvér a spustí ho [4]. Obranou proti technikám obfuskácie je dynamická analýza.

Dynamickú analýzu kódu predstavuje debugovanie spustiteľného súboru. Debugovanie je technika, ktorou je možné spúšťať program kontrolovaným spôsobom. Pomocou nástroja na debugovanie (debugera) je možné spúšťať postupne jednotlivé inštrukcie postupne. Je možné spustiť aj viacero vybraných inštrukcií naraz alebo časť

programu preskočiť. Počas debugovania možno po vykonaní každej inštrukcie sledovať hodnoty v registroch a je možné ich aj zmeniť.

Útočníci používajú rôzne techniky na zistenie toho, či je malvér debugovaný. Malvér počas spustenia napríklad zisťuje, koľko času prešlo medzi vykonaním dvoch vzdialených inštrukcií. Pri použití debugera bude čas vyšší ako bez neho. Zistiť prítomnosť debugera je možné aj použitím funkcie jadra operačného systému *IsDebuggerPresent()* alebo sledovaním rôznych situácií v operačnom systéme, resp. v samotnom programe [172]. Pri zistení prítomností debugera môže malvér narušiť svoje vykonávanie alebo sa pokúsiť využiť zraniteľnosti v debugeri, aby spôsobil jeho zlyhanie [173].

Analýza pamäte RAM po spustení kódu je typicky forenzná technika. Jej integrácia do analýzy malvéru je veľmi užitočná pri zisťovaní schopností malvéru skryť svoju činnosť alebo vyhnúť sa ochrane systému (antivírusovým programom) [171]. Z pamäte sa dá získať zoznam bežiacich procesov. Pre každý proces je možné zistiť, aké sieťové spojenia nadväzoval a k akým systémovým prostriedkom (procesy, súbory, registre) pristupoval. Ak chce proces pristúpiť k objektu, musí pristúpiť k jeho handle a cez handle pristúpi k samotnému objektu a vykoná na ňom operácie. Handles pre každý proces sa dajú získať z pamäte. Procesy majú v pamäti načítané knižnice (DLL), aj samotné spustiteľné súbory procesov, ktoré sa dajú získať z pamäte. Z pamäte sa dajú získať aj kľúče registrov, ku ktorým sa pristupovalo.

1.1.2 Štruktúra PE formátu

V tejto kapitole sa sústredíme na malvér vo formáte PE (portable executable) [10, 174]. Tento typ spustiteľných súborov bol predstavený spolu s operačným systémom (OS) Windows NT a dnes je špecifický pre OS Windows. V tejto kapitole popíšeme časti PE súboru v poradí, v akom sa nachádzajú v súbore.

PE súbory začínajú MS-DOS hlavičkou, ktorá informuje operačné systémy MS-DOS, Windows 3.1 a staršie, že program je s daným OS nekompatibilný. Nasleduje real-mode stub (MS-DOS stub) program, ktorý je tiež určený pre staršie OS a spustí sa v nich namiesto samotného programu. Táto časť má za úlohu vypísať chybovú hlášku o nekompatibilite OS.

Hlavička PE súboru obsahuje metadáta (napríklad počet sekcií, čas tvorby súboru). Po nej nasledujú voliteľné hlavičky, ktoré sú povinné a obsahujú ďalšie metadáta a charakteristiky súboru. Samotný spustiteľný súbor sa delí na sekcie. Každá sekcia má svoju hlavičku. Tieto hlavičky sa nachádzajú za hlavičkami samotného súboru. Obsahujú metadáta a informácie o jednotlivých sekciách, napríklad veľkosť, adresa v súbore. Za hlavičkami sekcií sa nachádzajú už samotné sekcie. Aplikácia pre Windows NT má zvyčajne 9 preddefinovaných sekcií - .text, .bss, .rdata, .data, .rsrc, .edata, .idata, .pdata a .debug [174]. Medzi špeciálne sekcie, ktoré majú určený zmysel a sú rezervované, patria napríklad aj sekcie .tls, a .reloc [10].

Sekcia .text obsahuje kód programu. Označuje sa aj ako spustiteľná sekcia, keďže jej obsah sa po načítaní programu vykonáva. Sekcia .bss slúži na ukladanie staticky alokovaných objektov, ktoré nie sú explicitne inicializované (sekcia neinicializovaných dát). Sekcia v spustiteľnom súbore neobsahuje žiadne dáta, len virtuálnu veľkosť, ktorú po načítaní do pamäte operačný systém alokuje neinicializovaným objektom [167]. Statické dáta určené len na čítanie sú v sekcii .rdata. Medzi tieto dáta patria textové reťazce v programe (napr. chybové hlášky) a konštanty. Všetky ostatné premenné (inicializované dáta) sú uložené v sekcii .data. Sekcia .pdata obsahuje pole funkcií, ktoré sa používajú pri riešení výnimiek v programe.

Programy využívajú rôzne zdroje, napríklad obrázky, logá, zvukové súbory. Zdroje, ktoré program využíva, sú uložené v sekcii zdrojov .rsrc. Táto sekcia je rozdelená do stromovej štruktúry adresárov. Program môže exportovať funkcie a umožniť iným programom využiť ich. Tabuľka exportov je v sekcii .edata. Programy zväčša využívajú mnoho funkcií z knižníc operačného systému cez Windows API (aplikačné rozhranie). Tieto funkcie musí pred použitím importovať, a preto potrebuje informáciu o tom, ktoré funkcie sa po spustení importujú. Tabuľka importov sa nachádza v sekcii .idata.

Sekcia .debug obsahuje informácie, vytvorené počas debugovania. Sekcia .tls (thread local storage) umožňuje viacerým vláknam, na ktorých beží program, ukladať lokálne hodnoty pre jednotlivé premenné. Počas behu programu sa môže adresa objektov aj celého programu meniť. Aby jednotlivé časti programu mohli pristupovať k sebe navzájom, musia poznať tieto adresy. Z toho dôvodu sa používa relokačná tabuľka, ktorá sa aktualizuje po každej zmene adresy. Táto tabuľka sa uchováva v sekcii .reloc.

1.2 Úvod do selekcie atribútov malvéru

Prvým krokom pri všetkých metódach strojového učenia je extrakcia atribútov. V rámci nášho výskumu prebieha extrakcia atribútov zo vzoriek malvéru. Atribúty je možné deliť na dve skupiny podľa toho, akým spôsobom boli získané [3]. Podľa tohto spôsobu poznáme statické a dynamické atribúty.

Statické atribúty sú získané priamo zo súboru a ich výhodou je, že ich získanie je rýchle aj pri veľkom množstve malvéru. Avšak vďaka rôznym metódam šifrovania a obfuskácie sa útočníci dokážu vyhnúť správnej klasifikácii nimi vytvoreného malvéru. Z dôvodu použitia balíčkovania (packingu) zabraňujú útočníci aj reverznému inžinierstvu. Pri balíčkovaní je binárny program zabalený pomocou istej funkcie a potrebuje reverznú funkciu na to, aby sa dostal do spustiteľného stavu.

Dynamické atribúty vznikajú monitorovaním operačného systému a procesov (spusteným programom). Vďaka tomu sa možno vyhnúť problémom, spojeným so statickými atribútmi. Spúšťanie vzoriek malvéru je ale časovo náročné a získané údaje majú pomerne vysoké priestorové nároky. Navyše útočníci podmieňujú spustenie programu podmienkami, ktoré v kontrolovanom prostredí nemusia nastať [3].

Do niektorých skupín atribútov sa môže zaradiť veľmi veľké množstvo atribútov. Príkladom sú n-gramy, ktoré zachytávajú sekvencie znakov/slov pohybom posuvného okna cez súbor. Napríklad pri anglickej abecede je možných 456 976 4-gramov písmen, čo je pri klasifikácii vlastne vektor tejto veľkosti. N-gramy možno získať z viacerých zdrojov, napríklad n-gram bajtov. Takto môže vzniknúť vektor, ktorého dĺžka je rádovo v miliónoch, dokonca aj miliardách (4-gram bajtov). Robiť klasifikáciu na takomto vstupe je výpočtovo veľmi náročné (požiadavky na čas a pamäť). Navyše algoritmy na klasifikáciu majú vo všeobecnosti problémy s veľmi vysokou dimenzionalitou vstupu. Tento fakt je známy ako „kliatba dimenzionality“ (curse of dimensionality) [88]. Počet atribútov by mal byť relatívne malý oproti veľkosti tréningovej vzorky. V opačnom prípade nie je možné vytvoriť model, keďže algoritmus nevie určiť význam jednotlivých atribútov pre klasifikáciu a je náchylný k overfittingu [4].

Pred samotným učením je preto potrebné vybrať malé množstvo najdôležitejších atribútov z mnohodimenzionálnych skupín atribútov ako sú n-gramy. Tento proces sa označuje ako selekcia atribútov (feature selection). Po jej aplikácii vzniknú z n-gramov použiteľné atribúty. Tento proces je potrebné aplikovať pre každú skupinu atribútov,

ktorá sa veľkosťou blíži počtu vzoriek, prípadne je podozrenie, že obsahuje redundantné atribúty alebo irelevantné atribúty.

Existuje niekoľko prístupov k selekcii atribútov. Li, et al. [24] spomínajú až 40 rôznych metód. Metódy na selekciu sa delia na 3 hlavné skupiny [6]. Tieto skupiny možno podľa typu použitého algoritmu deliť na ďalšie podskupiny, napríklad podľa Li, et al. [24] existuje 5 skupín. Hlavné skupiny metód sú nasledujúce:

- Filtrové metódy – podľa danej metriky ohodnotia všetky atribúty a vyberú tie, ktoré sa nachádzajú nad špecifikovaným prahom. Sú nezávislé od klasifikátora (algoritmus, ktorý vykonáva klasifikáciu). Nezávislosť od klasifikátora im dáva výhodu generalizovateľnosti – možno ich použiť pre akýkoľvek klasifikátor [24]. Metódy sa líšia použitou metrikou. Nevýhoda je, že do úvahy sa berie ohodnotenie len pre jednotlivé atribúty a nie pre ich podmnožiny. Existujú aj verzie niektorých algoritmov pre ohodnotenie podmnožín [19,26] a aj keď sú menej náročné ako wrapper metódy, sú omnoho náročnejšie ako pôvodné. Tieto metódy možno rozdeliť do nasledujúcich kategórií [24]:
    - založené na štatistickej teórii (napríklad chi-square).
    - založené na teórii informácií (odvodené od entropie).
    - založené na podobnosti (napríklad Fisher skóre).
- Wrapper metódy – zo začiatočnej množiny atribútov budujú najlepšiu podmnožinu tak, že postupne do nej pridávajú alebo odoberajú atribúty a vzniknuté množiny atribútov použijú pri klasifikácii. Podľa výsledkov klasifikácie sa rozhodne, či nové atribúty v množine ostanú alebo sa odstránia. Tieto metódy sú omnoho pomalšie, lebo je potrebné vykonať množstvo iterácií časovo náročnej klasifikácie. Pre klasifikátor, ktorý sa použije, sú ale presnejšie.
- Embedded metódy – pri trénovaní sa vykonáva priamo aj selekcia atribútov. Nie je možné ich použiť pre všetky klasifikátory. Existujú metódy pre lineárnu regresiu, SVM – support vector machine [6] a pre rozhodovacie stromy (decision tree) [7]. Sú výpočtovo omnoho menej náročné ako wrapper metódy, keďže si vyžadujú len jedno trénovanie. Rozhodovacie stromy používajú filtrové metódy pri určení toho, ako rozdeliť strom na podstromy [7]. Takto sa aj stromy samotné dajú využiť pre selekciu atribútov. Z tohto dôvodu sa algoritmy na tvorbu stromov pomocou filtrových metód považujú za vložené (embedded) metódy [7].

Okrem selekcie atribútov sa používajú aj techniky na redukciu rozmerov (dimenzionality). Na rozdiel od selekcie nevyberú najlepšie atribúty, ale prevedú priestor atribútov na menej dimenzionálny tak, že atribúty transformujú. Výsledkom je, že vzniknú nové atribúty, popisujúce staré. Inými slovami sú ich kombináciou. Z toho dôvodu nie je možné určiť, ktoré pôvodné atribúty boli dôležité.

1.3 Klasifikácia metód selekcie atribútov

V tejto sekcii porovnáme metódy na selekciu atribútov, ktoré boli využívané v prácach, zaoberajúcich sa malvérom.

1.3.1 Filtrové metódy

Pokiaľ pri selekcii predpokladáme veľký počet atribútov, je vhodné rozdeliť selekciu na niekoľko krokov a na začiatku použiť metódy, ktoré sú výpočtovo menej náročné [5]. Pokiaľ je počet atribútov taký veľký, že je nemožné akokoľvek ich spracovať, je potrebné hneď na začiatku selekcie (pri extrakcii) niektoré atribúty vylúčiť.

Príkladom nespracovateľného množstva atribútov je dataset, ktorý vytvorili Raff, et al. [5], kde mali takmer 36 miliárd 6-gramov. Na ich načítanie by bolo potrebných 791 GB RAM pri 64-bit reprezentácii. Pre ich redukciu odstránili atribúty, ktoré sa vyskytujú len v malom počte súborov. Už pri hranici jedného percenta súborov bola redukcia viac ako 99.9 % z 1.6 milióna položiek. Tento postup sa používa pomerne často, použili ho napríklad Islam, et al. [2] a Menahem, et al. [11].

Spôsob ohodnotenia atribútov podľa frekvencie ich výskytu v celom datasete sa používa pri spracovaní textu. Nazýva sa document frequency (DF) – frekvencia dokumentov [11]. Wang, et al. [8] vybrali v prvom kroku 1-gramy, ktoré sa vyskytovali minimálne 200-krát v aspoň jednej vzorke. Navyše zaznamenávali len 4-gramy, ktoré obsahovali tieto 1-gramy. Až na týchto 4-gramoch neskôr urobili ďalšiu selekciu. V práci [9] Hwanga, et al. vybrali tiež v prvom kroku 1-gramy s 200 násobným výskytom a tvorili n-gramy len z nich. Na týchto n-gramoch potom vykonali ďalšiu selekciu.

Ohodnotenie, pri ktorom atribúty označíme podľa počtu výskytov v jednej vzorke, sa označuje ako term frequency (TF) – frekvencia výrazov. V našom datasete predstavujú výrazy atribúty a dokumenty sú vzorky malvéru. Santos, et al. v článku [12] používajú normalizovaný TF, teda vydelený celkovým počtom výrazov vo vzorke. Lysenko v rámci

[28] tiež použil TF, ale pre každú triedu osobitne. Je to najmä z dôvodu zabezpečenia zachytenia atribútov aj pre triedy, ktoré majú oproti ostatným triedam malú frekvenciu najčastejších n-gramov a do globálneho rebríčka by sa nedostali. Tian, et al. v článku [59] si rozdelili každú rodinu malvéru na dve množiny – tréningovú a testovaciu. Z tréningovej množiny extrahovali reťazce. Následne brali do globálnej množiny len tie reťazce, ktoré sa vyskytovali aspoň v desiatich percentách vzoriek z tréningovej množiny. Teda globálnu množinu tvorili reťazce, ktoré sa vyskytovali pomerne často v každej rodine.

Okrem jednoduchých ohodnotení na frekvenciu vo vzorke, či v datasete, sú aj komplexnejšie variácie. Výskum [13] používa tzv. Classwise document frequency (CDF) – frekvencia dokumentov vzhľadom na triedu (1).

$$\sum_{val \in \{0,1\}} \sum_{C \in \{M,B\}} P(val, C) \frac{P(val, C)}{P(val)P(C)} \tag{1}$$

Premenná *val* označuje, či sa n-gram v danej triede (aspoň jednej jej vzorke) nachádza. *P(val, C)* je množstvo vzoriek v danej triede, kde sa n-gram nachádza. *P(val)* je počet vzoriek celého datasetu obsahujúcich daný n-gram.

Veľmi často používaná metóda ohodnotenia je TF-IDF (term frequency – inverse document frequency), hlavne pri klasifikácii textových reťazcov. Danú metódu použili napríklad v prácach [5,14,15]. IDF je logaritmus podielu veľkosti datasetu a počtu vzoriek, ktoré obsahujú výraz. Je potrebné pričítať ku deliteľu konštantu, aby sa nedelilo nulou. TF-IDF je potom násobok TF a IDF. Existuje ale viacero možností ako definovať TF a IDF. Príkladom môže byť článok [46], ktorý používa iné verzie (2,3,4,5).

$$TF(d,t) = 1 + \log\big(1 + \log\big(freq(d,t)\big)\big) \tag{2}$$

$$TF(d,t) = 1 + \log\big(1 + \log\big(freq(d,t)\big)\big) \tag{3}$$

$$IDF(t) = \frac{\log(1 + |d|)}{|dt|} \tag{4}$$

$$TF - IDF(d,t) = TF(D,T) * IDF(T) \tag{5}$$

Premenná *d* je množina dokumentov, *t* je výraz a *dt* je množina dokumentov, obsahujúca daný výraz.

Raff, et al. v článku [5] ako jediní použili modifikovanú verziu Gini koeficientu s pridanou konštantou (6). Gini koeficient sa v selekcii používa na určenie distribúcie atribútov v datasete [89]. Vysoký koeficient znamená, že rozdelenie je veľmi nerovnomerné. Gini index uprednostňuje atribúty, ktoré nadobúdajú viac hodnôt [145].

$$Gini_c(gj) = \frac{2(m_j + c)(b_j + c)}{(m_j + b_j + 2c)^2} \tag{6}$$

Vo vzorci (6) c predstavuje konštantu, bez ktorej veľké množstvo n-gramov dosahovalo maximálne skóre. Premenné m_j a b_j znamenajú počet výskytov n-gramu pre triedu *m* a triedu *j*.

Shabtai, et al. v rámci článkov [16,17] využili Fisher skóre. Existuje aj jeho verzia pre výber najlepšej podmnožiny atribútov, ktorá odstraňuje redundantné atribúty – generalizované Fisher skóre [19]. Fisher skóre dáva vysoké hodnotenie atribútom, ktoré dosahujú podobné hodnoty vo vzorkách, patriacich do rovnakej triedy a zároveň rôzne hodnoty pre vzorky z rôznych tried [66]. Skóre *i*-tého atribútu Si sa počíta cez vzorec (7), kde premenné u_{ij} a σ_{ij} sú priemer a variancia hodnôt *i*-tého atribútu v *j*-tej triede, n_j je počet vzoriek v *j*-tej triede a u_i je celkový priemer *i*-tého atribútu.

$$S_i = \frac{\sum_{j=1}^{C} n_j (u_{ij} - u_i)^2}{\sum_{j=1}^{C} n_j * \sigma_{ij}^2} \tag{7}$$

Častejšie používané ohodnotenie bolo mutual information (MI) – vzájomná informácia, ktorá vyjadruje, aká veľká závislosť je medzi rozdelením jednotlivých atribútov a tried v datasete [66]. Pre rovnaký vzorec sa v inom kontexte používa aj označenie informačný zisk (information gain - IG). Vo všeobecnosti meria množstvo informácie, ktorú dve premenné zdieľajú [24]. Santos. Et al. [12] použili MI pre selekciu frekvencií inštrukcií. Tang, et al. [66] popisujú vzorec (8), založený na entropii, pre vstup f_i, C kde f_i je *i*-tý atribút a C je rozdelenie tried v datasete. Vo vzorci sa odčíta entropia atribútu od jeho podmienenej entropie po pozorovaní rozdelenia tried. Týmto sa získa rozdiel informácie v atribúte a informácie, ktorú atribút zdieľa s rozdelením tried. Po vyjadrení entropie je vzorec pre celý dataset nasledovný (9) [12].

$$IG(f_i, C) = H(f_i) - H(f_i|C) \tag{8}$$

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y)\, log\left(\frac{P(x,y)}{P(x)P(y)}\right) \tag{9}$$

MIG je veľmi obľúbená metóda vďaka jej jednoduchej interpretácii a nízkej výpočtovej zložitosti [66]. Použili ju Santos, et al. [12,20], Karampatziakis, et al. [21], Raff, et al. [5], Kang, et al. [22], Wang, et al. [8], Masud, et al. [56] a Hwanga, et al. [9]. Wang, et al. a Masud, et al. [8,56] použili MIG pre selekciu prvých 500 atribútov pre každú triedu. MIG dáva lepšie výsledky pre atribúty s väčším množstvom možných hodnôt. Toto odstraňuje použitie symetrickej neistoty (symmetrical uncertainty) [23].

Yan, et al. v článku [6] vyskúšali aj ďalšie dve filtrové metódy, a to ReliefF, ktorý počíta pomer vzdialenosti atribútu ku „*k*“ atribútom z vlastnej triedy (vzdialenosť vo vnútri triedy) a ostatných tried (vzdialenosť von z triedy) a F1-statistics [25]. F1-statistics je popísaná vo vzorci (10) kde *K* je počet tried, u je priemer hodnôt daného atribútu v celom datasete, n_k je počet vzoriek v triede k a u_k a σ_k sú priemerná a štandardná odchýlka atribútu v triede k.

$$F = \frac{\sum_{k=1}^{K} \frac{n_k}{K-1}(u_k - u)^2}{\frac{1}{n-K}\sum_{k=1}^{K}(n_k - 1)\sigma_k^2} \tag{10}$$

Yan, et al. a Fernández-Delgado, et al. [6,31] použili aj Chi-square skóre. Táto metóda funguje ako test nezávislosti. Pri selekcii atribútov sa zisťuje, či je distribúcia hodnôt atribútu v datasete závislá od označenia tried. Vyššie skóre znamená, že atribút je dôležitejší [24]. Pre atribút f_i , ktorý nadobúda *r* rôznych hodnôt, sa skóre počíta pomocou vzorca (12), kde n_{js} je počet vzoriek, v ktorých atribút nadobúda hodnotu *j* v triede *s*. Parameter u_{js} sa počíta pomocou vzorca (11) kde n_{j*} je počet vzoriek, v ktorých atribút nadobúda hodnotu *j* v celom datasete a n_{*s} predstavuje počet vzoriek v triede *s*.

$$u_{js} = \frac{n_{*s} n_{j*}}{n} \tag{11}$$

$$CSC(f_i) = \sum_{j=1}^{r} \sum_{s=1}^{c} \frac{(n_{js} - u_{js})^2}{u_{js}} \tag{12}$$

Chen, et al. použili v článku [63] vlastnú metódu na hodnotenie atribútov, Discriminating power measure (DPM). Pre každý atribút a každú triedu sa vypočíta absolútna hodnota rozdielu DF pre vzorky v danej triede a vzorky vo zvyšku datasetu. Pre daný atribút sa potom sčítajú tieto rozdiely pre každú triedu. Takto vznikne DPM atribútu. Vyššie DPM je lepšie, lebo znamená že atribút sa pre nejakú triedu vyskytuje často a zároveň vo všetkých ostatných triedach sa vyskytuje v malom počte.

1.3.2 Embedded metódy

Regularizácia bráni tvorbe príliš komplikovaného modelu (klasifikátora) tým, že pri trénovaní pridáva chybu, ktorá sa zväčšuje s komplexitou modelu. Regularizácia sa používa najmä z dôvodu, že komplexný model často značí, že dochádza k overfittingu - model je príliš ovplyvnený konkrétnym datasetom, čím sa zníži jeho generalizačná schopnosť a na iných dátach dosahuje slabé výsledky. Aj keď pôvodný účel regularizácie je zabránenie tvorbe komplexných modelov, ktoré nie sú generalizované, dá sa využiť pre selekciu atribútov. L1-regularizačná metóda na výber atribútov sa označuje aj ako LASSO (least absolute shrinkage and selection operator). Regularizácia mení koeficienty pri korelovaných atribútoch na hodnoty blízke nule. L1-norma dokonca aj na nulu (čím dané atribúty úplne vypadnú z modelu) [24,66]. Pri selekcii stačí potom vybrať tie atribúty, ktoré majú najväčšie koeficienty, prípadne ich majú nenulové.

Yan, et al. a Trofimov [6,29] použili L1-regularizovaný lineárny SVM model a Yan, et al. [6] aj L1-regularizovanú logistickú regresiu. V [5] Raff, et al. použili dve metódy pre logistickú regresiu, a to Lasso a Elastic net, ktorá používa lineárnu kombináciu L1 a L2 regularizácie. Podľa Yan, et al. [6] je vo väčšine prípadov veľmi malý rozdiel v presnosti klasifikátorov pri použití rôznych metód selekcie. V jednom prípade, kde bol rozdiel medzi Chi-squared a L1-regularizáciou, dokázali dosiahnuť dobré výsledky pri menšom počte atribútov (menej ako 50) v porovnaní s F statistics alebo Relief (viac ako 100 atribútov).

Wang, et al. a Trofimov v rámci článkov [8,29] používajú náhodný les (random forest) na nájdenie stromu s najlepším výsledkom. Vyberú sa tie atribúty, ktoré použil náhodný les na rozdeľovanie.

1.3.3 Wrapper metódy

Metódy sa líšia použitou optimalizáciou na tvorbu najlepšej podmnožiny atribútov, keďže všetky podmnožiny je často nemožné vyskúšať (exponenciálna zložitosť). V článku [27] Ahmadi, et al. používajú greedy metódu forward stepwise feature selection (postupný výber atribútov). Táto metóda začína s prázdnou množinou atribútov a v každom kroku sa do nej pridá jeden atribút. Backward verzia zase začína s množinou všetkých atribútov a postupne sa z tejto množiny odoberajú atribúty. V obojsmernej verzii možno v každom kroku pridať alebo odobrať atribút, podľa toho, ako sa použitý algoritmus rozhodne. Z dôvodu vysokej časovej a výpočtovej náročnosti Ahmadi, et al.

za jeden atribút považujú celú skupinu atribútov z jedného zdroja (napr. n-gramy z binárneho súboru) [27].

1.3.4 Redukcia dimenzionality

Trofimov v rámci [29] vyskúšal analýzu hlavných komponentov (Principal Component Analysis – PCA), faktorizáciu nezáporných matíc (Non-negative matrix factorization – NMF) a analýzu nezávislých komponentov (Independent Component Analysis – ICA). NMF dávalo najlepšie výsledky. Wojnowicz, et al. a Dahl, et al. v článkoch [47,49] používajú RPCA – náhodné PCA, ktoré má nižšiu výpočtovú zložitosť, ale nemusí dávať najlepšie výsledky. Gibert, et al. a Mariconti, et al. v rámci [32,48] tiež použili PCA. V článku [14] Lin, et al. použili PCA a KPCA – Kernel PCA.

PCA ortogonálne transformuje atribúty do iného, menšieho priestoru pomocou dekompozície matice atribútov v datasete [98]. Vzniknú tak nové atribúty, ktoré sú od seba nezávislé a sú kombináciou pôvodných. Popisujú všetky pôvodné atribúty, aj keď ich je menej. Navyše sú zotriedené podľa variancie. Tieto nové atribúty sa nazývajú hlavné komponenty. Dimenzie, ktoré sa vyhodia, najmenej ovplyvňujú varianciu [97]. NMF faktorizuje maticu atribútov v datasete podobne ako PCA ale s tým rozdielom, že matice vzniknuté po faktorizácii nemajú v sebe negatívne hodnoty. Nové vektory atribútov tak predstavujú len aditívnu kombináciu pôvodných vektorov, čo je intuitívnejšie [98]. ICA na rozdiel od PCA nepožaduje pri transformácii ortogonalitu, namiesto toho požaduje nezávislosť medzi komponentami bázy matice [99].

Ďalším prístupom k redukcii dimenzionality je feature hashing, kde sa pomocou nejakej funkcie prevedú atribúty do menšieho priestoru – vstupom funkcie sú pôvodné atribúty, výstupom sú nové. Feature hashing použili Jung, et al., Andersons a Roth, Jang, et al. v článkoch [33,58,60]. Funkcie môžu mať rôznu komplexitu, napríklad Jung, et al. [33] pri frekvencii bajtov zaznamenávali len jednu hodnotu pre bajty, ktoré prekročili určitú hranicu. Sú aj funkcie, pri ktorých sa aj niekoľko atribútov zlúči do jednej hodnoty. Napríklad Jung, et al. v článku [33] používajú súčet polynómov pre vyjadrenie 4-gramu do jedného čísla.

1.4 Označovanie malvéru (labeling)

Pred samotnou klasifikáciou je pri tvorbe vlastného datasetu potrebné každej vzorke určiť triedu (označiť ju). Označiť dataset malvéru možno viacerými spôsobmi. Prvé riešenie je zhlukovanie datasetu, kde vzniknuté zhluky budú predstavovať jednotlivé triedy. Druhá možnosť je analyzovať prvky datasetu cez ochranu koncových zariadení a prideliť mu triedu na základe toho, do akej triedy ho zaradil program. Trieda môže závisieť od konsenzu pri použití viacerých programov na ochranu koncových zariadení. Možný je aj kombinovaný spôsob – na zhlukovanie sa nepoužijú dáta priamo z datasetu, ale dáta z ochrany koncových zariadení. Tretia možnosť je časovo najnáročnejšia – dôsledná analýza každej vzorky a manuálne priradenie triedy na základe výsledkov analýzy.

Spôsobov zhlukovania je mnoho. Xu a Tian [140] porovnávajú 71 algoritmov. Okrem výberu algoritmu je dôležité vybrať aj metriku pre vzdialenosť, prípadne podobnosť jednotlivých vzoriek. V rámci toho istého článku sa uvádza 8 najčastejších metrík. Takto môžu vzniknúť stovky kombinácií zhlukovania. Niektoré hlavné skupiny algoritmov podľa Xu a Tian [140] sú:

- partition based: sú efektívne, ale nevhodne reagujú na outliere (hodnoty ktoré sa vo veľkej miere odlišujú od zvyšku dát). Nevedia rozdeliť nekonvexné oblasti. Jadrové (kernel) verzie to dokážu pomocou jadrového (kernel) triku. Ide o transformáciu do viacrozmerného priestoru, kde sa snažia nekonvexné zhluky previesť na konvexné. Príkladom sú K-means, K-medoids, K-medians.
- hierarchické: vedia nájsť zhluky vo všetkých typoch údajov, ale sú výpočtovo náročné. Príkladom sú BIRCH (balanced iterative reducing and clustering using hierarchies), CURE (Clustering Using REpresentatives), ROCK (RObust Clustering using linKs), Chameleon.
- density based: efektívne zvládnu všetky tvary zhlukov. Na druhej strane majú vysoké požiadavky na pamäť. Výsledky môžu byť nepresné, ak nie je rovnomerná hustota vzoriek. Príkladom sú DBSCAN (Density-based spatial clustering of applications with noise), OPTICS (Ordering points to identify the clustering structure), mean-shift.

- grafové: vzorky predstavujú vrcholy grafu a vzťahy medzi nimi predstavujú hrany. Príkladom sú CLICK (CLuster Identification via Connectivity Kernels) a MST (algoritmus založený na Minimum Spanning Tree).
- grid based: priestor sa rozdelí do mriežky, ktorá sa dá zhlukovať paralelne. Sú rýchle, ale nie veľmi presné. Príkladom sú CLIQUE (CLustering in QUEst), STING (STatistical INformation Grid).
- swarm intelligence based: Simulujú meniace sa procesy v biologickej populácii (kolónia mravcov, včiel). Príkladom sú kategórie algoritmov PSO (Particle Swarm Optimization), ACO (Ant Colony Optimization), SFLA (Shuffled frog leaping Algorithm), ABC (Artificial bee colony algorithm).
- affinity propagation based: Všetky vzorky sú považované za potenciálne centrá a negatívna hodnota Euklidovskej vzdialenosti medzi nimi predstavuje ich afinitu. Vzorka, ktorá má vyšší súčet afinity, má väčšiu šancu stať sa stredom zhluku. Do tejto kategórie patria napríklad rôzne greedy algoritmy.

Rôzne metódy zhlukovania sa využívajú aj pre označovanie datasetov malvéru. Annachhatre, et al. v článku [45] získali sekvencie operačného kódu (operation code), ktoré použili ako vstup pre Hidden Markov model. Z takýchto atribútov dosiahli pomocou K-means zhlukovania 82% presnosť klasifikácie.

Sahu, et al. v článku [46] používajú metódu zhlukovania Kernel k-means na atribútoch frekvencie inštrukcií. Kernel k-means využíva kernel trick známy pre SVM – premapuje atribúty do viacrozmerného priestoru, kde už môžu byť separovateľné, keďže k-means nedokáže rozdeliť lineárne neseparovateľný dataset. Takto získali presnosť až 78%.

Označovanie pomocou ochrany koncových zariadení sa pri datasetoch malvéru tiež často využíva, keďže je pre tento typ datasetu špecifické. Yan, et al. v článku [6] použili službu VirusTotal [90], ktorá v sebe zahŕňa analýzu pomocou viac ako 70 antivírusových programov (AV). Označenia jednotlivých programov rozdelili do slov a odstránili príliš generické slová. Potom vynechali správy (reporty) od AV, ktoré nepoužili žiadnu zo známych tried malvéru. Pre zvyšné programy si zistili, aké aliasy používajú pre dané rodiny a pomocou nich už kontrolovali výstupy programov (aspoň 4 správne označenia z 5). Alias vzniká pri situácii, keď rôzne antivírusové programy označia tú istú triedu iným (často podobným) názvom.

Li, et al. v článku [40] urobili množinu slov z označení AV, odstránili málo informatívne slová (napr. agent, malware). Odstránili vzorky, ktorých označenia naznačovali zabalenie a obfuskáciu (packers, packed, obfuscators) a potom spočítali frekvenciu každého slova. Najčastejšie slovo pre vzorku označili ako jej triedu za predpokladu, že tvorilo ¾ označení AV, ktoré vzorku označili ako malvér. Nakoniec odstránili príliš malé rodiny.

Canzanese, et al. v rámci článku [41] si vybrali niekoľko AV a používajú označenie väčšiny. Podobne postupovali Nataraj, et al. aj v článku [42], ale ako triedu brali označenie, na ktorom sa zhodli 2 zo 6 vybraných AV. Na druhej strane, v článku [43] Zhao, et al. za triedu považovali označenie, na ktorom sa zhodlo 5 AV zo všetkých AV. Takto im ostalo 30% pôvodného datasetu.

Kombinácia dát z ochrany koncových zariadení a zhlukovania je tiež používaná. Kolosnjaji, et al. v rámci článkov [37,39,44] robili binárny vektor zo všetkých označení vybraných AV (antivírusov). Tieto vektory zhlukujú pomocou metódy DBSCAN [38] cez kosínusovú vzdialenosť (podiel skalárneho súčinu vektorov a súčinu ich veľkostí).

Tvorcovia služby Holmes Processing [50] si určili kľúčové slová a z označení malvéru vyhodili všetky prefixy, sufixy a slová, ktoré neboli kľúčové. Potom spočítali výskyt kľúčových slov pre dataset a nechali len tie, ktoré sa vyskytovali aspoň 50-krát. Pre každý malvér urobili binárny vektor podľa toho, či sa v ňom vyskytovali tieto slová. Tento vektor použili pre označovanie zhlukovaním.

1.5 Klasifikácia malvéru

Existuje veľké množstvo metód klasifikácie. Fernández-Delgado, et al. v rámci svojho výskumu [31] porovnali 179 klasifikátorov zo 17 rodín na 121 datasetoch. Medzi rodiny klasifikátorov ktoré porovnali patria nasledujúce [31]:

- support vector machine (SVM),
- neurónové siete,
- bayesovské klasifikátory,
- rozhodovacie stromy (decision trees - DT).

Z pohľadu klasifikácie malvéru sme porovnali 55 článkov za posledných 10 rokov [2,3,6,8,22,27,32-34,37,39,41-43,47,49,55,59,61,97,101-102,104-111,115-126,128-

139]. Pri Microsoft Malware Classification Challenge používali účastníci na najvyšších priečkach [8,28,29] XGBoost [30], čo je state-of-art (2015) ensemble RF. Medzi článkami, ktoré sme porovnávali, mal XGBoost dve najvyššie priečky v presnosti (accuracy) – 99.83% [8] a 99.77% [27]. XGBoost používa aj regularizáciu (embedded metóda) [30].

Podľa Fernández-Delgado, et al. [31] (porovnanie 179 klasifikátorov) bol najlepší klasifikátor Random forest – RF, druhý bol SVM. Najlepšie rodiny boli tiež RF a SVM. Pri klasifikácii malvéru sa v poslednom roku používali hlavne konvolučné siete – CNN, napríklad v článkoch [32,33,34]. Globálne sa najviac používal DT, resp. RF – spolu v 30 článkoch, samotný DT bol v 17. Druhé najpočetnejšie boli SVM – 16 výskytov. Druhý bol aj K-nearest neighbours classification [35] – 16 výskytov, až potom CNN – 12 a nakoniec Naive Bayes [36] – 8, ktorý mal aj globálne najhoršie výsledky. Porovnanie článkov je v prílohe B. Pri úspešnosti je potrebné brať do úvahy, že nie všetky články používali rovnaké metriky. Niektorí robili krosvalidáciu, iní mali testovací dataset. Niektorí neuvádzali, ako získali výsledok, takže je možné, že to bol výsledok trénovania, nie testovania, preto je úspešnosť len orientačná.

1.5.1 Metriky klasifikácie

Výsledky trénovania klasifikátora je potrebné ohodnotiť vhodnou metrikou. Plnú informáciu o presnosti pri trénovaní zachováva len confusion matrix. Jej riadky aj stĺpce predstavujú jednotlivé triedy. V každom políčku je počet vzoriek, ktoré majú triedu prislúchajúcu stĺpcu, ale klasifikátor ich zaradil do triedy prislúchajúcej riadku (alebo naopak). Matica, prislúchajúca dokonalému klasifikátoru, by mala nenulové hodnoty len na diagonále. Pri binárnej klasifikácii má matica štyri polia:

- TP – pozitívne klasifikované výsledky, ktoré sú pozitívne.
- FP – pozitívne klasifikované výsledky, ktoré sú reálne negatívne.
- TN – negatívne klasifikované výsledky, ktoré sú negatívne.
- FN – negatívne klasifikované výsledky, ktoré sú pozitívne.

Z matice je často náročné porovnať výsledky dvoch klasifikátorov, obzvlášť pri veľkom počte tried. Preto sa používajú namiesto nej rôzne metriky, ktoré sa snažia informácie z matice zhrnúť do jedného čísla. Toto číslo by si malo zachovať čo najväčšie množstvo informácie. Pri väčšine prípadoch by malo byť vyššie pre lepší klasifikátor.

Niektoré z týchto metrík sa dajú použiť len pre binárnu klasifikáciu. Ak ich chceme použiť pri multinominálnej klasifikácii, je potrebné brať ju ako sériu binárnych klasifikácií, na ktorých sa aplikuje metrika a z jej výsledkov sa urobí priemer [96].

Najjednoduchšia metrika je presnosť (accuracy), ktorá predstavuje podiel správne klasifikovaných vzoriek (TP a TN) a celého datasetu (TP+TN+FP+FN). Ďalšia metrika je precíznosť (precision). Je vyjadrením toho, aký podiel nájdených výsledkov je relevantných. Inými slovami, koľko reálne pozitívnych výsledkov (TP) bolo medzi tými, ktoré boli označené ako pozitívne (TP+FP). Recall (označovaný aj ako true positive rate - TPR) je podiel TP ku všetkým reálne pozitívnym vzorkám (TP+FN). Hovorí, aké kompletné sú výsledky – teda koľko z reálne pozitívnych vzoriek bolo správne klasifikovaných. Okrem TPR sa dá merať aj FPR (fallout). Ide o podiel FP ku všetkým negatívnym vzorkám (FP+TN). Hovorí o tom, ako veľmi je klasifikátor naklonený nesprávne označovať vzorky ako negatívne. F-skóre (F faktor) berie do úvahy precíznosť aj recall a je ich harmonickým priemerom.

Všetky tieto metriky sú ale neobjektívne (biased), čo znamená, že existujú prípady, keď horší klasifikátor dosiahne lepšie skóre [100]. Pre recall existuje objektívna verzia - informovanosť (informedness), ktorá kvantifikuje ako veľmi je klasifikátor informovaný o danej podmienke (ako sú rozdelené triedy). Popisuje, aká je pravdepodobnosť, že je o nej informovaný (oproti tomu, že klasifikuje náhodne) a vyjadruje sa rozdielom TPR (recall) a FPR. Objektívna verzia existuje aj pre presnosť a označuje sa ako zreteľnosť (markedness). Číselne vyjadruje, ako veľmi je daná podmienka zreteľná pre klasifikátor. Súčasne špecifikuje pravdepodobnosť, že daná podmienka bola klasifikátorom označená ako indikátor výsledku (marker). Pred tým, ako popíšeme spôsob jej výpočtu, uvedieme sekundárny názov pre precíznosť – True Positive Accuracy (TPA), keďže predstavuje podiel reálne pozitívnych vzoriek (TP) a všetkých vzoriek označených ako pozitívne (TP + FP) (teda má rovnaký vzorec ako presnosť, ale ignoruje negatívne označené vzorky). Obdobne sa dá definovať False Negative Accuracy (FNA) – podiel FP a všetkých vzoriek označených ako negatívne (FN + TN). Zreteľnosť potom je rozdiel TPA a FNA [100].

Ďalšia metóda používa ROC krivku (receiver operating characteristic curve). Krivka je vytvorená ako funkcia, ktorá odráža zmeny medzi TPR a FPR. TPR predstavuje hodnoty na *Y* osi a FPR hodnoty na *X* osi. Diagonála reprezentuje náhodu, krivka pod diagonálou znamená klasifikáciu horšiu ako náhoda, krivka nad ňou naopak lepšiu. Pre

vyjadrenie jedným číslom sa používa obsah oblasti pod krivkou (area under the curve – AUC) – čím vyšší obsah, tým lepší klasifikátor [100].

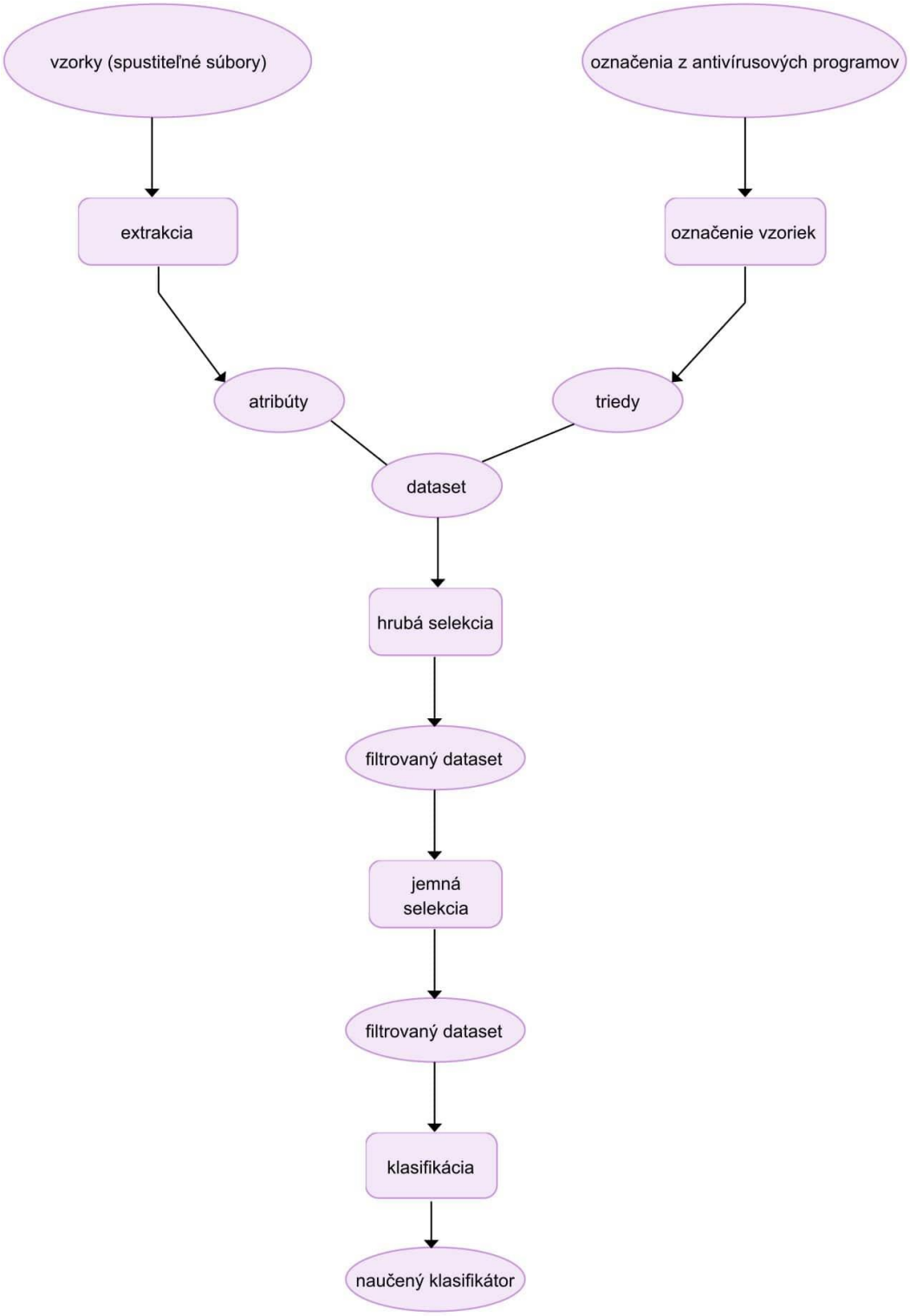
2 Metodológia

Proces klasifikácie malvéru s použitím selekcie atribútov musíme rozdeliť na niekoľko etáp:

- označovanie vzoriek (labeling),
- extrakcia atribútov,
- hrubá selekcia atribútov (výpočtovo nenáročné metódy),
- jemná selekcia (filtrové a embedded metódy),
- samotná klasifikácia.

Diagram etáp klasifikácie je zobrazený na obrázku (Obr. 1).

Obr. 1 Etapy klasifikácie malvéru.



Na začiatku vytvárame čo najväčšiu sadu skupín atribútov, nad ktorými vykonávame postupnú selekciu a určíme, ktoré atribúty sú najdôležitejšie. Našim cieľom je nájsť najmenšiu podmnožinu atribútov, ktorej výsledky sú porovnateľné s výsledkami,

získanými s použitím všetkých atribútov (výsledná presnosť nie je nižšia o viac ako percento).

Naším cieľom je zistiť aj to, či je možné nájsť takúto podmnožinu len pre málodimenzionálne skupiny atribútov. Ak by to bolo možné, nebolo by potrebné mnohodimenzionálne skupiny atribútov ani len extrahovať zo vzoriek, čo by ušetrilo množstvo času a pamäte. Podľa nám dostupnej literatúry existuje jediná práca [27], ktorá sa sústreďuje na porovnanie celých skupín atribútov. Ahmadi, et al. v danej práci nevykonali prvotnú selekciu atribútov. Vykonali len selekciu celých skupín atribútov, v ktorých mohlo byť veľa redundantných atribútov, spôsobujúcich horšiu presnosť. Navyše nemali k dispozícii samotné vzorky, len hexadecimálnu reprezentáciu súborov a disasemblovaný súbor bez hlavičky. Z tohto dôvodu niektoré skupiny atribútov neboli schopní získať.

Samozrejme sa malvér časom mení a o pár rokov by nami vybrané atribúty stratili rozdeľovaciu schopnosť kvôli novým skupinám malvéru. V prípade, že by začala klesať presnosť predikcie, stačilo by vytvoriť nové rodiny malvéru, urobiť znovu selekciu jednotlivých atribútov a riešenie by fungovalo naďalej. Samotné modely klasifikácie je tiež nutné časom preučiť a pri preučení je vhodné vykonať aj novú selekciu.

2.1 Označovanie (labeling)

Triedy malvéru sme sa rozhodli získavať pomocou značenia antivírusových programov (AV) cez službu VirusTotal [90]. Keďže viaceré AV pri zabalených a obfuskovaných vzorkách používajú generické názvy ako „packed“, „obfuscator“, tak sme sa rozhodli podobne ako v práci [40] v našom prvom datasete takéto vzorky odstrániť. Dôvodom bolo najmä, aby nám nevznikli pri zhlukovaní osobitné triedy pre zabalený a obfuskovaný malvér, ktoré by nemali zmysel. Tieto vzorky sme odstránili pomocou entropie. Podľa Lyda a Hamrock v článku [57] je s presnosťou 99% dôveryhodný (confidence) interval entropie pre binárny súbor od 4.941 do 5.258. Pre zabalený súbor to je interval od 6.677 do 6.926 a pre šifrovaný je interval od 7.174 do 7.177. Pri počítaní entropie brali do úvahy len bloky, v ktorých je aspoň polovica bajtov nenulových. Z toho dôvodu reálna entropia bude ešte nižšia. Preto sme sa rozhodli vylúčiť vzorky s entropiou väčšou alebo rovnou ako 6.66. Pre porovnanie sme si nechali aj druhý dataset, v ktorom ostali aj tieto vzorky. Takto sme mali možnosť vidieť zmeny v selekcii atribútov

a presnosti klasifikácie pre čisté vzorky a miešaný dataset. Okrem toho sme si nechali aj dataset, v ktorom boli len obfuskované, šifrované a zabalené vzorky. Vďaka tomu budeme vedieť porovnať zmeny vo vybraných skupinách atribútov, keď nie sú vzorky v čistom stave.

Druhá možnosť by bola urobiť rozbalenie (unpacking) na všetkých vzorkách detegovaných ako zabalených a až potom urobiť filtrovanie pomocou entropie. Toto by zahŕňalo pre každú vzorku nájdenie použitého balíčkovacieho nástroja (packera) (cez nástroje ako je PEiD [86] alebo Packerid [87]), ktorý by sa použil na jej rozbalenie (unpacking). Vyskúšali sme aj túto možnosť. Použili sme UniExtract, (Universal extractor) čo je nástroj na rozpoznanie a extrakciu súboru z veľkého množstva archívov. Jeho výhoda je, že má batch mode, takže dokáže naraz rozbaliť veľké množstvo súborov [155]. Na siedmych skúšobných vzorkách nám rozbaľovanie trvalo to takmer 3 minúty a dostali sme jeden výsledok „not packed“, a ostatné boli „failed“ alebo „unknown“. Z tohto dôvodu sme sa rozhodli nerozbaľovať vzorky.

Nami navrhované riešenie označenia vzoriek spočívalo v spojení označení rôznych antivírusových programov vzorky do jedného dlhého slova a zhlukovaní týchto slov. Náš predpoklad bol, že vzorky malvéru, ktoré by mali patriť do rovnakej triedy, jeden AV označí vždy rovnako, prípadne veľmi podobne (budú mať minimálne rovnakú alebo podobnú príponu/predponu). Ak by to platilo, tak spojením názvov by sme získali triedy, na ktorých všetky AV súhlasia. Vzorky, ktoré veľa AV označí podobne, budú patriť do jednej triedy. Výhoda je, že tieto spojené slová nemusia byť rovnaké, stačí, ak sú podobné. Nevýhodou tohto prístupu je, že vzorky, ktoré majú len generický názov, ktorý neobsahuje triedu, (AV ich nevedeli zaradiť) sa zhluknú do jednej triedy, prípadne do viacerých generických tried. Kvôli eliminácii generických tried je preto potrebné pred zhlukovaním odstrániť z datasetu vzorky, ktoré majú len generický názov. Za metriku sme si vybrali Levenshtein distance [52]. Uvažovali sme aj nad Damerau-Levenshtein distance [53], keďže obe metriky sú symetrické a spĺňajú trojuholníkovú nerovnosť, čo je podmienkou pre metriku vzdialenosti. Levenshtein distance predstavuje rozdiel medzi slovami – minimálny počet zmien jedného písmena, ktoré z jedného slova vytvoria iné. Povolené zmeny sú vkladanie, mazanie a výmena písmena za iné. Damerau-Levenshtein distance pridáva výmenu pozícií pre dve písmena vedľa seba, čo sa môže použiť pri preklepoch.

Pre zhlukovanie sme sa rozhodli použiť metódu s prístupom, založeným na hustote (density-based), pretože tieto metódy efektívne zvládnu spracovať všetky tvary zhlukov, aj nekonvexné [140]. Z tejto skupiny metód sme si vybrali algoritmus HDBSCAN. Ide o relatívne nový (2013) algoritmus, ktorý spája výhody hierarchického prístupu a prístupu, založeného na hustote a je rýchlejší ako podobný algoritmus OPTICS. Jeho časová zložitosť je *O(n*log(n))* [38]. HDBSCAN na rozdiel od DBSCAN nepoužíva parameter pre minimálnu vzdialenosť bodov v zhluku, ale namiesto toho má parameter len pre minimálnu veľkosť zhluku. Takto odstraňujú nevýhodu algoritmov, založených na hustote, ktoré majú problém, ak zhluky majú rôznu hustotu (hustota nie je rovnomerná). Tento algoritmus dokáže správne zhlukovať aj rôzne husté zhluky [51].

Druhá možnosť je za triedu považovať najčastejšie sa vyskytujúce sa slovo pri označení rôznych AV - konsenzus. Ak neodstránime málo informatívne slová, je možné, že generické slová typu „trojan“ prevládnu kvôli tomu, že rôzne AV majú rôzne aliasy pre tú istú rodinu. Napríklad Yan, et al. v článku [6] tieto aliasy našli a pri určovaní najčastejšieho slova ich všetky pokladali za jedno slovo, lebo pri piatich AV dostávali aj tri rôzne mená, ktoré boli často veľmi podobné (niekedy len rozdiel v jednom písmene). Preto najprv identifikujeme príliš generické slová. Tieto slová z tried vynechávame a potom na zvyšných triedach hľadáme aliasy, ktoré počítame za jednu triedu. Vzorka malvéru, ktorá má rovnakú triedu (aj s aliasmi) na stanovenom počte z vybraných AV, dostane túto triedu ako svoju značku (label) pri klasifikácii. Pri klasifikácii porovnávame výsledky pre zhlukovanie a konsenzus.

Vyskúšali sme obidva prístupy k označovaniu (labelingu) a ich presnosť sme ohodnotili pri klasifikácii. Zhlukovanie malo lepšie výsledky, ale len minimálne (približne pol percenta). Z tohto dôvodu sme robili selekciu atribútov pre obe riešenia. Podrobnejšie sa budeme zaoberať výsledkami testov v nasledujúcej kapitole. Robili sme pokusy aj s kombinovaným prístupom – získanie možných tried cez konsenzus a vytváranie skupín cez Levenshtein distance. Ukázalo sa, že toto riešenie je menej robustné, ako keď sa do úvahy berú celé názvy. Mnoho skupín malo veľmi krátke názvy a skôr, ako sa začali zlučovať skupiny s dlhšími názvami, ktoré k sebe naozaj patrili, (napr. „wannacry“ a „wannacryptor“), už boli zlúčené skupiny s krátkymi názvami (napr. „dorv“, „worm“). Je to najmä z dôvodu, že pri Levenshtein distance je vzdialenosť minimálne rozdiel dĺžky slov. Toto kombinované riešenie sme nakoniec nepoužili.

2.2 Extrakcia atribútov

Zahrnuli sme čo najviac statických atribútov, použitých v nami analyzovaných 55 článkoch o klasifikácii malvéru a v riešeniach najlepších riešiteľov Microsoft Malware Classification Challenge. Keďže náš dataset obsahuje len súbory v PE (portable executable) formáte, tak sme mohli zahrnúť aj atribúty, špecifické pre tento formát. Tento formát je špecifický pre operačný systém Windows. Vynechali sme atribúty, ktoré vznikli transformáciou binárneho súboru na obrázok, pretože predstavujú ďalšie výpočtové zaťaženie, a atribúty, ktoré tvorili graf, lebo sa nedali bez ďalšej transformácie použiť pri klasických klasifikátoroch.

Pri n-gramoch sme sa obmedzili maximálne na 2-gramy (teda berieme len unigramy a bigramy) kvôli pamäťovým obmedzeniam, aj keď sa často používali aj 3-gramy a 4-gramy a niektoré práce pracovali až s 10-gramami (napríklad [22,29]). Pôvodne sme extrahovali aj 3-gramy, ale keďže po prvotnej selekcii nám ostal takmer milión atribútov, čo bolo viac ako sme dokázali spracovať, rozhodli sme sa ich vynechať. Efektívne sú všetky n-gramy pre n>=2. Rôzne štúdie sa líšia v identifikácii najlepšieho „n“ [5]. N-gramy môžu byť binárne-existenčné a frekvenčné [22]. Hoci 1-gramy sú neefektívne, pri frekvenčnej verzii, čo je vlastne frekvencia jednotlivých položiek vo vzorke, to neplatí. Aj samotné 1-gramy majú rozlišovaciu schopnosť medzi malvérom a benignwarom (neškodný softvér) [22]. Odteraz myslíme pod extrakciou n-gramu získanie oboch verzií.

2.2.1 Popis atribútov

Zdrojmi statických údajov sú najčastejšie hexadecimálne binárne súbory, disasemblované súbory a samotná vzorka.

Hexadecimálny súbor možno získať priamo cez powershell pomocou format-hex cmdlet, hexdump [74], dumpbin [75], ktorý je vo Visual Studio, a mastiff [76]. Disasemblovaný súbor (s inštrukciami) tvorí IDA [77], dumpbin, objdump [78] a radare2 [79]. Údaje priamo zo vzorky možno získať množstvom programov s otvoreným kódom (open source programov). Príkladmi sú pescanner [80], pev [81], multiscanner [127] a peframe [82]. Medzi ďalšie nástroje patrí pype32 [113], pyew [165] a generic parser [166]. Viacero takýchto programov sa vyskytuje v Linux distribúcii REMnux [85]. Existujú aj knižnice pre python – pefile [83], ktorú používa viacero programov na

získanie údajov zo vzoriek a LIEF [84], ktorú extenzívne Anderson, et al. využili v článku [58]. Nástroje, ktoré sme vyskúšali, a ich výstupy sú spracované v prílohe C. Niektoré nástroje mali minimálnu dokumentáciu, preto zoznam výstupov nemusí byť úplný. V prílohe sú len nástroje, ktoré majú možnosť vrátiť súbor s výstupom.

Z hexadecimálneho binárneho súboru získavame n-gramy, pričom 1-gram predstavuje jeden bajt, čo sú pri hexadecimálnej reprezentácii dve znaky (táto reprezentácia sa skladá zo štvoríc po 16 bitov) [13]. Ďalej zaznamenávame frekvencie výskytov každého n-gramu. V normalizovanej verzii sú frekvencie vydelené veľkosťou súboru v bajtoch. Zaznamenávame aj veľkosť tohto súboru. Ďalší atribút vychádza zo samotnej štruktúry súboru – histogram entropie bajtov (byte-entropy histogram) [54]. Konkrétne Saxe a Berlin v [54] mali posuvné okno po 1024 bajtov s posunom po 256 bajtov (skokový 1024-gram), na ktorom počítali entropiu a frekvenciu jednotlivých bajtov. Potom urobili histogram, rozdelený na oboch osiach na 16 hodnôt. Os X zachytávala entropiu a os Y počet jednotlivých bajtov, obe rozdelené na 16 políčok na histograme. Rovnaké riešenie použili Anderson, et al. [58], ale s oknom o veľkosti 2048 a posunom okna 1024. Lyda a Hamrock [57] mali veľkosť okna 256 bajtov a už 512 bolo podľa nich veľa, lebo malé okná nízkej entropie ovplyvňovali celkovú entropiu. Ahmadi, et al. [27] použili veľkosť okna 10000 bajtov pre hexadecimálnu reprezentáciu na získanie štatistík o entropii. Vyskúšali sme viacero veľkostí okna, no pri žiadnej konfigurácii sa po prvotnej selekcii nevylúčili žiadne atribúty histogramu. Preto sme usúdili, že veľkosť okna nie je dôležitým parametrom. Nechali sme okno s veľkosťou 1024 bajtov a s posunom 256 bajtov. Histogram sme mali rovnaký ako Saxe a Berlin [54], keďže sme použili ich zdrojový kód.

V disasemblovanej vzorke n-gramy predstavujú postupnosť assembly inštrukcií. Keďže inštrukcie môžu mať parametre, ktoré môžu odkazovať na miesta v súbore tak do n-gramov, extrahujeme inštrukcie bez parametrov, ináč by ich vzniklo enormné množstvo. Registrov procesora nie je až tak veľa, a preto môže ako atribút slúžiť frekvencia použitia jednotlivých registrov [27]. My sme robili pre registre priamo n-gramy, v rámci ktorých je frekvencia priamo zahrnutá. Ďalšie atribúty, ktoré pomôžu zachytiť viac informácie o inštrukciách, sú frekvencia dĺžok riadkov a ich priemerná dĺžka, počet riadkov, použitých inštrukcií (aj operačného kódu) a registrov. Pre zistenie, koľko priestoru si vzorka malvéru reálne rezervuje, používame ďalší atribút – súčet výskytov inštrukcií dd, db, dw a pomer ich súčtu k súčtu frekvencií všetkých inštrukcií

[27]. Tieto inštrukcie sa používajú na inicializáciu – rezervujú miesto v operačnej pamäti. To isté sme urobili pre všetky jump inštrukcie [8].

V disasemblovanej vzorke sa vyskytujú informácie aj na úrovni strojového kódu. Tieto informácie pozostávajú z operačných kódov (opcodes), uložených v hexadecimálnom tvare. Operačné kódy korešpondujú s inštrukciami a sú určené pre procesor – určujú, akú operáciu má procesor vykonať [173]. Na týchto operačných kódoch sme pôvodne tiež robili n-gramy, ale bolo ich príliš veľa (viac ako 5 miliónov 1-gramov). Preto sme sa rozhodli robiť n-gramy len na bajtoch, nie na celých operačných kódoch. Poslednými atribútmi sú veľkosť disasemblovaného súboru a pomer jeho veľkosti s veľkosťou pôvodného súboru a hexadecimálnej verzie súboru. Tieto pomery sme urobili pre všetky kombinácie týchto troch súborov.

Zo samotnej vzorky (spustiteľného súboru) je možné získať veľké množstvo atribútov. Programy (aj malvér) využívajú pri práci funkcie z knižníc operačného systému. My sme zaznamenali počet importovaných funkcií pre každú knižnicu, výskyt každej funkcie vo vzorke a počet importovaných knižníc, výskyt jednotlivých exportov vzorky a ich počet.

V ďalšej časti budeme popisovať atribúty špecifické pre PE (Portable Executable) typ spustiteľných súborov, keďže všetky naše vzorky sú tohto typu. Prvá skupina atribútov sa získava z metadát v hlavičke PE súboru. My sme vybrali nasledujúce metadáta [10]:

- CodeSize (suma veľkostí sekcií),
- LinkerVersion (verzia linkera, ktorý načíta program s knižnicami do pamäte),
- SubsystemVersion (verzia Windows subsystému, ktorý bude spúšťať program. Napríklad subsystém IMAGE_SUBSYSTEM_WINDOWS_GUI pre programy s grafickým rozhraním je uvedený pod číslom 2),
- InitializedDataSize (suma veľkostí inicializovaných dátových sekcií),
- UninitializedDataSize (veľkosť neinicializovanej dátovej sekcie BSS),
- OSVersion,
- ImageVersion (verzia programu),
- TimeStamp (čas kompilácie),
- EntryPoint (miesto v programe, v ktorom je prvá inštrukcia programu),

- MachineType (typ CPU),
- Size (SizeOfImage – veľkosť programu po načítaní do pamäte).

Ku PE súborom možno pridávať dáta bez zmeny v hlavičke. Tieto dáta sa označujú ako overlay. Overlay sa pri načítaní programu automaticky nenačítava. Malvér v nich môže skrývať škodlivú časť kódu. Preto zaznamenávame offset, veľkosť a entropiu týchto dát. Ak vzorka nemá overlay, všetky hodnoty nastavíme na nulu.

Ďalšia skupina atribútov sa viaže na sekcie, na ktoré je PE súbor rozdelený. Existujú všeobecne známe sekcie: .text, .data, .bss, .rdata, .edata, .idata, .rsrc, .tls, .reloc. [27], ale pri tvorbe programu je možné definovať si vlastné sekcie. Keďže pri každej vzorke musíme mať pri tréningu rovnaký počet atribútov, tak pri atribútoch, ktoré sa viažu na jednotlivé sekcie (napríklad veľkosť sekcie), berieme do úvahy len známe sekcie. Aby sme zachovali aspoň nejaké informácie o neznámych sekciách, pridávame atribúty, ktoré zachytávajú neznáme sekcie ako celok (napríklad ich počet). Atribúty, viažúce sa k sekciám, sú nasledovné:

- počet sekcií a počet sekcií s prázdnym menom,
- počet známych a počet neznámych sekcií,
- entropia, veľkosti a virtuálne veľkosti známych sekcií,
- súčet veľkostí známych a neznámych sekcií a pomer ich veľkostí, tiež pomery ich veľkostí k veľkosti celého súboru,
- proporcia veľkostí známych sekcií k veľkosti súboru, aby sme mali informáciu o abnormálne veľkých/malých sekciách,
- počet zdrojov v resource sekcii a počet zdrojov jednotlivých typov (data, text, image...),

Posledná skupina atribútov zachytáva informácie z textových reťazcov (printable strings) ktoré sa vyskytujú vo vzorke. Anderson, et al. v [58] berú len reťazce dĺžky aspoň 5 znakov. Tiam, et al. a Islam, et al. v [59,3] berú reťazce z IDA, ktorá predvolene extrahuje tiež len reťazce s dĺžkou aspoň 5 znakov. Skúšali sme aj zoznam kratších reťazcov, ale obsahovali priveľa krátkych nezmyselných zhlukov znakov. Navyše, podľa Tiam, et al. [59], sú krátke reťazce extrémne bežné, a preto nemá zmysel ich extrahovať. Preto sme si zvolili aj my minimálnu dĺžku 5 znakov. Pri reťazcoch je dobré zamerať sa okrem všeobecných atribútov aj na atribúty, zachytávajúce počet špecifických reťazcov, ktoré bližšie popisujú činnosť vzorky. Medzi reťazce, ktorým venujeme zvýšenú pozornosť, patria tie, ktoré v sebe obsahujú umiestnenie na disku („C:\”), webové stránky

(“http”), registre (“HKEY_”) a reťazec „MZ“. Reťazec „MZ“ sa vyskytuje na začiatku každého PE súboru. Ak je ich vo vzorke viac, môže to indikovať že sú v nej zabalené ďalšie spustiteľné programy [58]. Sledujeme aj počet reťazcov, ktoré tvarom zodpovedajú IP adresám (IPv4). Po získaní reťazcov z nich extrahujeme nasledujúce všeobecné atribúty:

- n-gramy znakov a slov,
- frekvencie dĺžky reťazcov v intervaloch a priemerná dĺžka reťazcov,
- počet reťazcov a veľkosť súboru, v ktorom sú reťazce,
- entropia celého zoznamu reťazcov.

2.3 Hrubá selekcia

Ešte pred samotnou selekciou už pri extrakcii atribútov možno robiť redukciu dimenzionality. Ak nejaký atribút naberá príliš veľa hodnôt, tak zlúčime blízke hodnoty do intervalov a takto nám vznikne histogram [55]. Pokiaľ nebudeme zlučovať príliš veľa hodnôt, tak väčšina informácie sa zachová, pričom dimenzionalita bude násobne nižšia a zbavíme sa problému, pri ktorom by väčšina dimenzií mala nulovú hodnotu. Intervaly sme nakoniec využili len pri frekvencii dĺžok reťazcov. Viac o tom píšeme v tretej kapitole.

Prvým krokom je odstránenie atribútov, ktoré sa vyskytujú len v malom počte vzoriek z celého datasetu (document frequency), aby sme pri nasledujúcich výpočtoch mali menej dimenzií. Malý počet berieme vzhľadom na veľkosť najmenšej triedy, aby sme nevylúčili atribúty, ktoré v malej triede predstavujú veľkú časť a môžu mať rozdeľovaciu silu vzhľadom na túto triedu. Z tohto dôvodu vylučujeme len atribúty, ktoré sa v globálnom datasete vyskytujú v počte vzoriek menšom ako je jedna desatina veľkosti najmenšej triedy.

Dôvodom, prečo sme sa sústredili na frekvenciu dokumentov je aj to, že má nižšie pamäťové nároky ako frekvencia výrazov. Pri DF stačí držať v pamäti len boolean hodnoty pre jednotlivé dokumenty namiesto sčítavania frekvencií, ktorých môže byť enormné množstvo. Navyše to, že TF atribútu je vo vzorkách nejakej triedy malé, neznamená, že nemôže byť použitý pri klasifikácii, naopak, práve táto skutočnosť môže byť pri klasifikácii využitá. DF má časovú zložitosť $O(V*A)$, kde V je počet vzoriek a A je počet atribútov, keďže musí skontrolovať každú hodnotu každého atribútu. Keďže sa

v jednotlivých krokoch nerobia žiadne výpočty, tak DF je oproti ostatným algoritmom rýchlejší.

Okrem odstránenia atribútov, ktoré sa vyskytujú v príliš malom počte vzoriek, je dobré odstrániť aj atribúty, ktoré majú v príliš veľkom počte vzoriek rovnakú hodnotu. Ak má atribút rovnakú hodnotu vo väčšine vzoriek, tak ich nemôže dobre rozdeliť. Tento fakt využíva selekcia podľa nízkej variancie (low variance selection). Vzťah na výpočet variancie je vo vzorci (13). *E* predstavuje priemer. Variancia meria rozpätie hodnôt premennej. Pokiaľ je rovná nule, tak premenná nadobúda v celom datasete rovnakú hodnotu [24]. My vylúčime atribúty s varianciou menšou ako prah blízky nule, teda také, ktoré nadobúdajú veľmi malé množstvo hodnôt. Prah sme určili ako 0.01. Ak variancia atribútu bola nižšia ako prah, atribút sme odstránili. Pri binárnej premennej sa taký prah dosiahne, ak je 99% rovnakých hodnôt. Pri premenných, kde sú rozdiely medzi hodnotami väčšie, je potrebných viac rovnakých hodnôt. Keďže nám ostávalo príliš veľa atribútov, tak pre skupiny atribútov, v ktorých bolo viac ako tisíc položiek, sme prah zvýšili na 0.05 (95%). Ak bolo nad desaťtisíc položiek, tak prah bol 0.1 (90%). Pri počítaní variancie je potrebné prechádzať dáta dvakrát, ale existujú algoritmy na jeden prechod, ktoré dokážu vypočítať varianciu s malou chybou. Variancia má časovú zložitosť *O(V*A),* kde *V* je počet vzoriek a *A* je počet atribútov [141].

$$VAR(X) = E[(X - E[X])^2] \tag{13}$$

Aj po vykonaní všetkých týchto selekcií môže byť veľa atribútov, ktoré sú redundantné, lebo aj keď sa vyskytujú prevažne len v jednej triede alebo sú časté vo všetkých triedach, tak iné atribúty sú im veľmi podobné a majú rovnakú rozhodovaciu schopnosť. Tiež nám mohli ostať atribúty, ktoré sú irelevantné, ich nadobúdané hodnoty nekorelujú s ich priradením ku triedam. Tieto dve metódy sú ale veľmi dobré pre prvotnú selekciu, lebo vylúčia atribúty, ktoré sú nepoužiteľné bez ohľadu na klasifikáciu.

2.4 Jemná selekcia

Pôvodne sme počítali s tým, že nám po hrubej selekcii ostane dosť málo atribútov na to, aby sme zvládli z časového hľadiska použiť aj pomalšie algoritmy. Ostalo nám omnoho viac atribútov ako sme predpokladali. Z tohto dôvodu sme museli buď pomalšie algoritmy (napr. väčšinu algoritmov založených na teórií informácií) vylúčiť alebo rozdeliť jemnú selekciu na dve etapy. Rozdelenie na etapy má nevýhodu, že algoritmus

použitý v prvej etape môže vylúčiť atribúty, ktoré by lepší algoritmus v druhej etape nevylúčil. Z tohto dôvodu môže mať horšie skóre oproti ostatným algoritmom v druhej etape, ktoré pracujú na inom princípe. Skúsiť kombináciu všetkých dvojíc algoritmov pre obe etapy je časovo príliš náročné a výsledky môžu byť ťažko interpretovateľné. Preto sme sa rozhodli pre všetky atribúty použiť len algoritmy, ktorých časová a pamäťová zložitosť nám umožní takéto overenie. Časová zložitosť algoritmov musí byť maximálne lineárna vzhľadom na počet atribútov. Z hľadiska pamäte si metódy musia vystačiť s 32 GB RAM pre 500 000 atribútov, čo sme kontrolovali empiricky. Ako sme v úvode spomínali, máme aj osobitné dva datasety skupín atribútov, v ktorých ostalo po prvotnej selekcii len malé množstvo atribútov. Prvý dataset obsahuje skupiny atribútov, ktoré majú maximálne 1000 prvkov. Označujeme ho ako jednoduchý kvôli tomu, že obsahuje prevažne nekomplexné skupiny atribútov. Druhý dataset obsahuje skupiny, v ktorých je maximálne 100 prvkov a označujeme ho ako veľmi jednoduchý, keďže obsahuje prevažne najjednoduchšie skupiny atribútov, ktoré majú pevný počet prvkov. Na týchto dvoch datasetoch použijeme aj ostatné metódy. Takto zabezpečíme, že všetky metódy porovnáme na rovnakých dátach bez ovplyvnenia inými algoritmami. Navyše získame výsledky na dátach, ktoré sú pre nás zaujímavé, keďže môžu viesť k rýchlej extrakcii.

2.4.1 Rýchle metódy

Na tomto mieste sme vybrali prevažne algoritmy, založené na podobnosti a štatistickej teórii, ktoré majú vo všeobecnosti veľmi nízku výpočtovú zložitosť, ale nevedia sa vyhnúť redundancii. Z tohto dôvodu sa využívajú ako pred-selekcia skôr, ako sa použije zložitejší algoritmus [24].

Viaceré algoritmy, založené na štatistickej teórii, môžu pracovať len s diskrétnymi číslami. Toto obmedzenie sa týka aj algoritmov, založených na teórii informácií, ktoré chceme použiť na jemnú selekciu [24]. Niektoré z týchto algoritmov ale môžu pracovať so spojitými dátami za predpokladu, že sa zmení ich korelačná funkcia, napríklad za Pearsonovu koreláciu. Následne je potrebné osobitne riešiť prípad korelácie spojitého a diskrétneho atribútu [142]. Druhá možnosť je diskretizovať atribúty pred použitím selekcie. Diskretizácia prebieha tak, že dáta sa rozdelia do uniformných skupín a hodnoty v skupine sa zmenia na jednu hodnotu (priemer, medián). Týmto by sme stratili väčšie množstvo informácií. Preto sme sa rozhodli diskretizovať tak, že obmedzíme počet desatinných miest a každý atribút vynásobíme tak, aby sa stal celočíselným.

Metódy, ktoré môžeme pre ich rýchlosť využiť pre všetky atribúty, sú tieto:

- Chi-square [144],
- ANOVA F-skóre (Analysis of Variance) [143]
- Gini index skóre [24]
- Fisher skóre [19]
- Laplacian skóre [146]
- Trace ratio [147]
- SPEC (Spectral Feature Selection) [148]
- ReliefF [149]
- Mutual Information (Information gain) – informačný zisk [66]

Všetky tieto metódy majú maximálne lineárnu zložitosť vzhľadom na počet atribútov, ako si ukážeme pri popise jednotlivých metód. Z týchto metód patrí 5 medzi metódy založené na podobnosti (Fisher skóre, Laplacian skóre, SPEC, ReliefF, Trace Ratio). Gini index a Chi-square patri medzi štatistické metódy. ANOVA, ktorá generalizuje *t*-test, tiež patrí medzi štatistické metódy. Metóda informačného zisku patrí medzi metódy, založené na teórií informácií [24].

Verzia Chi-square, ktorú použijeme v knižnici sklearn, má zložitosť *O(C*A),* kde *C* je počet tried a *A* je počet atribútov [144]. Popis Chi-square sme už poskytli v prvej kapitole.

Najjednoduchšia verzia ANOVA, ktorú využijeme aj v našej práci v knižnici sklearn, je One way Anova for independent samples [143]. Jej zložitosť sme vypočítali ako *O(A*V),* kde *A* je počet atribútov a *V* je počet vzoriek. Táto metóda predstavuje pomer agregovaných rozdielov medzi priemermi atribútu v jednotlivých triedach a náhodnej variability, ktorá existuje medzi prvkami v rámci jednotlivých tried [143]. Popisuje ju vzorec (14). SS_{bg} a SS_{wg} sú popísané vo vzorci (15). Prvá premenná (SS_{bg}) predstavuje varianciu atribútu medzi skupinami. Je to suma kvadratických odchýlok priemerov jednotlivých skupín od priemeru datasetu, násobená ich veľkosťou. Druhá premenná (SS_{wg}) je suma kvadratických odchýlok prvkov jednotlivých skupín od priemeru skupiny. df_{bg} je stupeň slobody medzi skupinami, tu je rovný počtu skupín mínus jedna. df_{wg} je stupeň slobody vnútri skupín. Ide o rozdiel súčtu počtu prvkov jednotlivých skupín atribútov, rozdelených podľa tried a počtu tried.

Pri F-value metóde sa testuje nulová hypotéza, že prvky v jednotlivých triedach boli vybrané náhodne zo spoločného zdroja [143]. Hodnoty MS_{bg} a $MSwg$ sú odhadom variancie tohto zdroja. Ak je hypotéza pravdivá, tak F je rovné alebo menšie ako 1. Ak je hypotéze nepravdivá, tak hodnota F je väčšia ako 1. Aby mohla byť táto metóda použitá, je potrebné splniť určité predpoklady [143]. Škála, na ktorej sa meria závislá premenná (trieda), musí mať vlastnosti škály s rovnakým intervalom. Triedy musia byť nezávisle a náhodne vybrané zo zdrojovej populácie. Zdrojový dataset musí mať normálnu distribúciu a jednotlivé triedy podobnú varianciu. ANOVA je robustná voči variancii a preto dáva dobré výsledky aj keď nie je splnený posledný predpoklad, obzvlášť, ak triedy majú rovnakú veľkosť.

$$F = \frac{\frac{SS_{bg}}{df_{bg}}}{\frac{SS_{wg}}{df_{wg}}} = \frac{MS_{bg}}{MSwg} \tag{14}$$

$$SS_{bg} = \sum_g N_g \left(Mean_g - Mean_{total}\right)^2, SS_{wg} = \sum_X \sum_i (X_i - Mean_X)^2 \tag{15}$$

Gini index skóre [24] metóda umožňuje zistiť, či atribút vie rozdeliť vzorky, pochádzajúce z rôznych tried. Menší index znamená dôležitejší atribút [24]. Nulový index znamená, že všetky vzorky v skupine majú rovnakú triedu. Pre atribút f_i s r hodnotami sa postupne pre každú hodnotu j rozdelí do dvoch skupín W a $\overline{W}$ pre hodnoty atribútu menšie alebo rovné ako j a hodnoty väčšie ako j [24]. Skóre je popísaný vo vzorci (16).

$$GINI(f_i) = min_W \left[p(W) \left(1 - \sum_{s=1}^{C} p(s|W)^2 \right) + p(\overline{W}) \left(1 - \sum_{s=1}^{C} p\left(s|\overline{W}\right)^2 \right) \right] \tag{16}$$

C predstavuje počet tried. Pre všetky triedy sa počíta podmienená pravdepodobnosť triedy za predpokladu vzniknutej skupiny W, resp. $\overline{W}$. To, čo Jundong Li, et.al. nazývajú Gini index skóre, korešponduje s Gini nečistotou, ktorú si predstavíme pri rozhodovacích stromoch. Zložitosť implementácie, ktorú sme použili, sme vypočítali ako *O(A*V*C),* kde A je počet atribútov, V je počet vzoriek a C je počet tried. Keďže počet tried je zvyčajne veľmi malý, môžeme za zložitosť považovať *O(A*V).* Pre n tried možno postupne deliť vzorky až do n skupín, ak má atribút dosť hodnôt. Pritom sa ale zvýši zložitosť, lebo je potrebné vytvoriť všetky kombinácie hodnôt atribútu až po počet tried. Počet rozdelení by potom bol: $\frac{n!}{1!(n-1)!} + \cdots + \frac{n!}{k!(n-k)!}$, kde n je počet rôznych hodnôt atribútu a k je počet tried. Výpočtovo rýchlejšie riešenie je po nájdení najlepšieho

rozdelenia datasetu rekurzívne hľadať najlepšie rozdelenia pre vzniknuté skupiny vzoriek, čo sa robí pri stromoch.

Laplacian skóre patrí medzi metódy, založené na podobnosti. Algoritmus je založený na myšlienke, že dva body pravdepodobne súvisia, ak sú blízko seba. Nižšie skóre znamená, že atribút je dôležitejší. Časovú zložitosť nami použitej implementácie tohto skóre sme vypočítali ako $O(C*A*Vc^2)$, kde A je počet atribútov, C je počet tried a Vc je maximálny počet vzoriek v jednej triede. Rovnakú zložitosť má aj nami použitá implementácia Fisher skóre. Algoritmus prebieha nasledujúcim spôsobom[146]: najprv sa urobí graf susednosti medzi hodnotami atribútu vo všetkých vzorkách. Ak nemáme označený dataset, tak za suseda sa považuje k najbližších susedov podľa hodnôt atribútu. Ak poznáme rozdelenie tried, tak najbližší susedia sa berú len z rovnakej triedy. Potom sa urobí matica, kde pre spojené hodnoty x_i a x_j políčko nadobudne pre vhodnú konštantu t hodnotu (17). Pre políčka bez susedov bude mať nulovú hodnotu.

$$s_{ij} = e^{-\frac{\|\|x_i - x_j\|\|^2}{t}} \tag{17}$$

L_r je skóre r-tého atribútu, f_{ri} je hodnota atribútu v i-tej vzorke r-tého atribútu. pričom máme m vzoriek. Pre r-tý atribút sa definujú začiatočné argumenty (18):

$$\boldsymbol{f}_r = [f_{r1}, \dots, f_{rm}]^T, \mathbf{1} = [1, \dots 1]^T, D = diag(S\mathbf{1}), L = D - S \tag{18}$$

L sa nazýva Laplacian matica. Z argumentov sa vypočíta medzivýsledok (19).

$$\widetilde{\boldsymbol{f}_r} = \boldsymbol{f}_r - \frac{\boldsymbol{f}_r^T D \mathbf{1}}{\mathbf{1}^T D \mathbf{1}} \mathbf{1} \tag{19}$$

Nakoniec sa vypočíta samotné Laplacian skóre (20).

$$L_r = \frac{\widetilde{\boldsymbol{f}_r^T} L \widetilde{\boldsymbol{f}_r}}{\widetilde{\boldsymbol{f}_r^T} D \widetilde{\boldsymbol{f}_r}} \tag{20}$$

Trace ratio metóda vyberá najlepšiu podmnožinu atribútov pomocou Fisher alebo Laplacian skóre. Algoritmus je nasledovný [147]: vstup je d atribútov v matici X, z ktorých sa napokon vyberie m. Vytvorí sa selektovacia matica W o veľkosti $d*m$, ktorej i-tý stĺpec w_i má jednotku len na i-tom mieste, inde nuly. Vytvoria sa dva grafy susednosti ako pri Laplacian (respektíve Fisher) skóre. Jeden graf bude predstavovať lokálnu vzdialenosť vo vnútri tried. Súčasne má väčšie hodnoty pre prvky, ktoré majú podobné hodnoty alebo sú v rovnakej triede. Druhý graf predstavuje globálnu vzdialenosť prvkov atribútu medzi triedami. Má väčšie hodnoty pre prvky, ktoré sa líšia atribútmi alebo sú v

rôznych triedach. Z nich sa vytvoria Laplacian matice L_w *a* L_b. Skóre *i*-tého atribútu je vo vzorci (21).

$$score(f_i) = \frac{\boldsymbol{w}_i^T \boldsymbol{X} \boldsymbol{L}_b \boldsymbol{X}^T \boldsymbol{w}_i}{\boldsymbol{w}_i^T \boldsymbol{X} \boldsymbol{L}_w \boldsymbol{X}^T \boldsymbol{w}_i} \tag{21}$$

Myšlienka metódy je maximalizovať podobnosť prvkov atribútu vnútri tried a zároveň minimalizovať podobnosť medzi triedami [24]. Nami použitá implementácia konverguje ku globálne najlepšej podmnožine tým, že opakuje iterácie algoritmu s rovnakou zložitosťou, akú majú Fisher skóre a Laplacian skóre ($O(C*A*Vc^2)$). Skóre najlepších atribútov predošlej iterácie potom využíva pri výpočte novej iterácie [147].

SPEC (Spectral Feature Selection) je rozšírenie Laplacian skóre. Namiesto Laplacian matice má jej normalizovanú verziu (22).

$$\boldsymbol{L} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{D} - \boldsymbol{S})\boldsymbol{D}^{-\frac{1}{2}}$$
$$\widetilde{\boldsymbol{f}}_i = \boldsymbol{f}_i - \frac{\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{f}_i}{\left\|\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{f}_i\right\|} \tag{22}$$

SPEC má tri možné funkcie na skóre. Oproti Laplacian skóre pridávajú robustnosť proti šumu v dátach. Prvá z týchto funkcií je opísaná vo vzorci (23).

$$score(f_i) = \widetilde{\mathbf{f}_1^{\mathrm{T}}}\gamma(\boldsymbol{L})\widetilde{\boldsymbol{f}}_i \tag{23}$$

γ je funkcia, ktorá penalizuje vysokú frekvenciu hodnôt atribútu za účelom redukcie šumu [24]. Ak sa použije RBF pre vypočítanie vzdialenosti a nepoužije sa γ, tak zložitosť SPEC je $O(A*V^2)$ pre *A* atribútov a *V* vzoriek [148]. Keďže SPEC je len rozšírením Laplacian skóre a obe umožňujú selekciu s učiteľom aj bez učiteľa, rozhodli sme sa pri Laplacian skóre použiť verziu s učiteľom a pri SPEC verziu bez učiteľa. Týmto sme do použitých algoritmov selekcie zahrnuli aj reprezentanta metód, ktoré používajú učenie bez učiteľa. Vďaka tomu budeme môcť zistiť, či atribúty, ktoré najlepšie prirodzene rozdeľujú vzorky, majú porovnateľné výsledky s atribútmi, ktoré rozdeľujú najlepšie do vopred pripravených tried. Selekcia s učením bez učiteľa nám dá informáciu aj o kvalite datasetu, či ohodnotenia tried. Ak by bol veľmi veľký rozdiel medzi výsledkami selekcie s učiteľom a bez učiteľa (v tomto prípade SPEC a Laplacian), tak je možné, že atribúty zle popisujú vzorky alebo triedy sú pridelené nesprávne.

Metóda ReliefF má zložitosť *O(M*V*A),* kde *M* je počet opakovaní, *V* je počet vzoriek a *A* je počet atribútov [149]. *M* je zvyčajne veľmi malé, preto je reálna zložitosť *O(V*A)*. Táto metóda je založená na tom, že k váhe atribútu sa odčítavajú jeho rozdiely

v hodnotách vo vzorkách tej istej triedy a pripočítavajú rozdiely v rôznych triedach. Z tohto dôvodu má najlepší atribút podobné hodnoty v rámci jednotlivých tried ale relatívne rozdielne hodnoty medzi triedami. Algoritmus postupuje nasledovne [149]:

Každému atribútu A sa nastaví váha $W[A] = 0$. Potom sa m krát upravujú váhy atribútov nasledujúcimi krokmi. Najprv sa vyberie náhodná vzorka R_i. Pre túto vzorku sa nájde k najbližších susedov v jeho triede (H_j) a potom pre každú ďalšiu triedu C sa nájde ďalších k najbližších susedov $(M_j(C))$. Každému atribútu sa potom upraví váha vzhľadom na susedov (23) pomocou rozdielovej funkcie. Rozdielové funkcie môžu byť rôzne, pôvodná je popísaná vo vzorci (24).

$$W[A] = W[A] - \sum_{j=1}^{k} \frac{diff\left(A[R_i], A\left[H_j\right]\right)}{m * k} + \\ + \sum_{C \neq class(R_i)} \frac{\left[\frac{p(C)}{1 - p\left(class(R_i)\right)} \sum_{j=1}^{k} diff\left(A[R_i], A\left[M_j(C)\right]\right)\right]}{m * k} \tag{23}$$

$$diff(A[I_1], A[I_2]) = \frac{|A[I_1] - A[I_2]|}{\max(A) - \min(A)} \tag{24}$$

Funkcionalitu metódy informačného zisku (Mutual information) sme vysvetlili už v prvej kapitole. Algoritmus, ktorý používa knižnica sklearn pre výpočet informačného zisku, získa entropiu odhadom od vzdialeností k najbližších susedov. Takéto algoritmy majú pri použití triediaceho algoritmu (napr. quicksort) komplexitu $O\left(V\sqrt{kV}\right)$ až $O\left(\sqrt{k}V\right)$ pri hladkej distribúcii [150]. Naivná verzia tejto metódy má zložitosť $O(V^2)$.

2.4.2 Pomalé metódy

Rozhodli sme sa vôbec nepoužívať wrapper metódy. Dôvodom je, že trénovanie je časovo náročné a podmnožín atribútov je exponenciálne veľa. Aj po použití optimalizačných techník by sme museli vykonať veľké množstvo trénovaní. Ďalší problém je, že hodnota atribútu je viazaná na klasifikátor. Inými slovami, môže sa podstatne zmeniť pri zmene parametrov alebo funkcií klasifikátora (napríklad kernelu pri SVM). Z tohto dôvodu nemusí mať až takú veľkú výpovednú hodnotu pre ostatné klasifikátory.

Rovnaký problém majú aj embedded metódy. Tieto metódy ale priamo využívajú štruktúru klasifikátora a jeho trénovací algoritmus. Teda neberú ho ako blackbox. Navyše majú iné výhody. Na rozdiel od filtrových metód ensemble DT metódy vedia zachytiť aj interakcie medzi viacerými atribútmi (nielen dvojicami) pri svojom použití filtrových metód [69]. Z tohto dôvodu sme sa rozhodli ich používať. Ich problémom je, že neodstraňujú redundantné atribúty. Interakcie vedia zachytiť aj regularizačné metódy. Medzi nich patrí Lasso – L1 regularizácia, Ridge – L2 regularizácia či Elastic net – L1 aj L2. HSIC lasso (Hilbert-Schmidt Independence Criterion Least Absolute Shrinkage and Selection Operator) vie zachytávať nelineárne vzťahy [67,68]. Z toho dôvodu ho zahrnieme medzi metódy, ktoré budeme testovať. Embedded regularizáciu ponúka SVM (L1, L2 aj elastic net). Aj stromy môžu priamo používať regularizáciu. Ako príklad môžeme uviesť XGBoost, ktorý využíva gradient boosting [30]. Takisto ju používa aj Regularized Greedy Forest (RGF) [73].

Pri filtrových algoritmoch sme sa zamerali na tie, ktoré odstraňujú irelevantné aj redundantné atribúty, čo znižuje nevýhodu ich pomalosti. Za pomalé považujeme metódy, ktorých časová zložitosť je horšia ako lineárna vzhľadom na počet atribútov. Nevýhoda týchto metód je, že nájdu koreláciu medzi dvojicami (pairwise correlation), ale vzťahy medzi viacerými atribútmi, ktoré ani nemusia ani byť lineárne, nezachytia. Medzi tieto algoritmy môžeme zaradiť:

- FCBF (Fast Correlation Based Filter for Feature Selection) [23],
- CFS (Correlation-based Feature Selection) [26],
- Mutual Information Feature Selection (MIFS) [24],
- Minimum Redundancy Maximum Relevance (MRMR) [24],
- Conditional Infomax Feature Extraction (CIFE) [24],
- Joint Mutual Information (JMI) [24],
- Conditional Mutual Information Maximization (CMIM) [24],
- Double Input Symmetrical Relevance (DISR) [24].

Všetky algoritmy v tomto zozname okrem CFS patria medzi algoritmy, založené na teórií informácií, a všetky okrem CFS a FCBF možno previesť do jedného teoretického frameworku (conditional likelihood maximization framework) [24]. Túto skutočnosť využíva aj nami používaná knižnica scikit-feature [170]. V tejto implementácii sme vypočítali časovú zložitosť týchto metód ako $O(V*A^2)$, kde A je počet atribútov a V je

počet vzoriek. Najpomalšia metóda z tohto frameworku je DISR, ktorý síce má rovnakú zložitosť, ale vykonáva najviac operácií o zložitosti *O(V)*.

CFS, ktoré patrí medzi štatistické metódy, berie do úvahy koreláciu medzi atribútom a triedou pre nájdenie irelevantných atribútov a medzi dvojicami atribútov pre nájdenie redundantných. Ako koreláciu používa symetrickú neistotu (26), ktorá nezvýhodňuje atribúty s vyšším množstvom možných hodnôt ako informačný zisk [24]. Na začiatku si vypočíta korelácie pre každý jeden atribút a od najlepšieho atribútu postupne buduje najlepšiu množinu. Najlepšiu množinu buduje heuristickým hľadaním s možnosťou vybrať si stratégiu hľadania: forward selection, backward elimination a best first [26]. CFS je popísané vo vzorci (25). *S* je *k*-prvková podmnožina atribútov, r_{cf} je priemerná korelácia medzi triedami a atribútmi a r_{ff} je priemerná korelácia medzi atribútmi navzájom. Čitateľ predstavuje prediktívnu silu množiny a menovateľ predstavuje, koľko redundancie je v množine [24]. Zložitosť algoritmu je kvadratická vzhľadom na počet atribútov – $O(A^4)$ [23]. Táto metóda má najvyššiu zložitosť z metód, ktoré sme používali.

$$CFS(S) = \frac{k * r_{cf}}{\sqrt{k - k(k-1)r_{ff}}} \tag{25}$$

$$SU(X_S, Y) = 2\frac{IG(X_S, Y)}{H(X_S) + H(Y)} \tag{26}$$

Na podobnom princípe (korelácia medzi atribútmi a triedami a korelácia medzi atribútmi navzájom) pracuje FCBF. Najprv si algoritmus vyberie množinu atribútov *S*, ktorá je vysoko korelovaná s rozdelením tried. Táto korelácia je nad stanoveným prahom δ. Ako koreláciu používa symetrickú neistotu (SU). SU medzi atribútom X_s a rozdelením tried *Y* sa počíta pomocou vzorca (14), kde *H* je entropia a *IG* je informačný zisk. Korelácia medzi atribútom F_i a triedou *C* sa nazve predominantná, ak $SU(F_i, C)$ je väčšie ako prah a zároveň pre žiaden iný, už vybraný atribút F_j neplatí, že $SU(F_i, F_j) \geq SU(F_i, C)$. Tie atribúty, pre ktoré táto nerovnosť platí, sú označené ako redundantné, vzhľadom na F_i a uložia sa do množiny Sp_i. Samotný atribút F_i je predominantný, ak jeho korelácia ku triede je predominantná alebo sa takou stane po odstránení jeho redundantných atribútov. Redundantné atribúty v množine sa potom rozdelia. Tie, pre ktoré $SU(F_j, C) > SU(F_i, C)$, sa uložia do Sp_i+. Tie, pre ktoré platí opačná nerovnosť, sa uložia do Sp_i-. Zo všetkých troch skupín sa potom heuristicky odstránia niektoré atribúty. Štartovacia heuristika vyberie za predominantný atribút ten F_p, ktorý má najvyššiu koreláciu s rozdelením tried. Potom sa skontrolujú atribúty s nižšou koreláciou

s triedami. Všetky, ktoré sú redundantné pre F_p, sa odstránia. Nové F_p bude atribút s najvyššou koreláciu s triedami, ktorý sa neodstránil. Proces sa opakuje, kým neostanú žiadne atribúty, ktoré by sa mohli v priemere vyhodiť [23]. Vzhľadom na to, ako pracuje, FCBF sa nedá použiť na ohodnotenie všetkých atribútov, ale jeho cieľom je nájdenie najmenšej dobrej množiny atribútov. Zložitosť je *O(V*A*log(A))* pre *A* atribútov a *V* vzoriek [23].

DISR je popísaný vo vzorci (27). Atribúty, ktoré dopĺňajú už vybrané atribúty, majú podľa jeho autorov vyššiu šancu byť vybrané, ako pri ostatných kritériách [153].

$$DISR(k) = \sum_{j \in S} \frac{IG(jk, Y)}{H(jk, Y)} \tag{27}$$

Ostatné algoritmy tiež pracujú na princípe korelácie medzi atribútmi a triedami, aj atribútmi navzájom. Algoritmus MIFS pre nový atribút *k,* ktorý chce pridať medzi vybrané atribúty *S*, je popísaný vo vzorci (28). MIFS pridáva ku IG penalizáciu za redundanciu. Metóda penalizuje atribúty, ktoré korelujú s už vybranými atribútmi, čo možno vidieť vo vzorci (28), kde sa od skóre odčíta penalta. MRMR len špecifikuje β z MIFS ako $\frac{1}{|S|}$, vďaka čomu sa so zvyšujúcim počtom vybraných atribútov znižuje penalta za redundanciu. Zložitosť je $O(A^{|S|})$ kde *A* je počet atribútov, *S* je množina vybraných atribútov [151]. Táto zložitosť je pre presné vyhľadávanie, ale je možné dosiahnuť takmer optimálny výsledok cez inkrementálne hľadanie (postupne sa pridá do *S* atribút s najlepším skóre) so zložitosťou *O(|S|*A)* [151,152]. CIFE ku MIFS pridáva maximalizáciu podmienenej redundancie medzi vybranými atribútmi a nevybraným atribútom vzhľadom na triedu [24]. Toto zabezpečuje pripočítaním podmieneného informačného zisku medzi atribútmi za predpokladu rozdelenia tried *Y*. β má rovné jednotke.

JMI počíta len podmienený informačný zisk. Myšlienkou je pridávať nové atribúty, ktoré dopĺňajú už vybrané atribúty vzhľadom na triedu. Vyberá sa atribút, ktorý najviac zvýši komplementárnu informáciu, zdieľanú s už vybranými atribútmi za predpokladu poznania triedy [24]. CMIM metóda, popísaná vo vzorci (29), počíta minimum podmieneného informačného zisku medzi novým atribútom a rozdelením tried za predpokladu už vybraných atribútov. Potom sa berie atribút, ktorý má toto minimum najvyššie, teda má silnú koreláciu s triedami a zároveň nie je redundantný vzhľadom na už vybrané atribúty.

$$MIFS(k) = I(k,Y) - \beta \sum_{j \in S} I(k,j) \tag{28}$$

$$CMIM(k) = \min_{j \in S} (I(k,Y|j)) \tag{29}$$

2.5 Klasifikácia a embedded metódy

Keďže podľa Fernández-Delgado, et al. [31] sú najpresnejšie klasifikátory (aj rodiny klasifikátorov) DT a SVM a súčasne medzi nami porovnávanými článkami boli aj najpoužívanejšie, rozhodli sme sa ich použiť. Konkrétne pri DT používame ich ensemble – Random forest. Pri DT používame aj ensemble založený na gradient boostingu – XGBoost, keďže mal najlepšie skóre z porovnávaných klasifikátorov. Jeho časová zložitosť je *O(K*d*m*log(n))* pre exact greedy verziu algoritmu [30], kde *d* je maximálna hĺbka stromu, *K* je počet stromov, *m* je počet atribútov a *n* je počet vzoriek. Ponúka aj blokovú štruktúru uloženia dát, pri ktorej sa atribúty ukladajú v usporiadaných blokoch. Pri tejto štruktúre sa čas zníži na *O(K*d*m+ m*log(n))* [30].

Okrem XGBoost používame aj dve novšie gradient boosting algoritmy, ktoré tiež dosahujú veľmi dobré výsledky: LGBM (LightGBM) [71] a Catboost [72], ktorý má rozšírenú podporu pre kategorické atribúty. LGBM pre zrýchlenie trénovania používa Gradient-based One-Side Sampling (GOSS), kde sa pre počítanie informačného zisku berú do úvahy hlavne vzorky malvéru s vysokým gradientom. Teda tie vzorky, ktoré potrebujú najviac zmenšiť svoju chybu. Pre zrýchlenie výpočtu pri riedkych (sparse) dátach sa používa Exclusive Feature Bundling. V tomto prípade sa zlučujú atribúty, ktoré nenadobúdajú naraz nenulové hodnoty, prípadne ich naraz nadobúdajú málokedy [95].

Gradient boosting metóda funguje nasledovne [30]: pre dataset *X* s *m* atribútmi, cieľovým atribútom *Y* (trieda) a *n* príkladmi (vzorkami) stromový ensemble používa *K* aditívnych funkcií na predikciu výsledku (30).

$$\hat{y}_i = \theta(x_i) = \sum_{k=1}^{K} f_k(x_i), f_k \in F \tag{30}$$

V tomto vzorci je $F = \{f(x) = w_{q(x)}\}(q: R^m \to T, w \in R^T)$ je priestor regresných stromov. q reprezentuje štruktúru každého stromu, ktorá mapuje príklad na index korešpondujúceho listu (T je počet listov). Každé f_k korešponduje s jedným stromom q a váhami jeho listov w. Na rozdiel od rozhodovacích stromov má každý strom skóre na každom liste a w_i je skóre i-tého listu. Pre každý príklad sa pomocou rozhodovacích

pravidiel q stromy rozhodnú klasifikovať príklad do jedného zo svojich listov. Potom sa počíta finálna predikcia sčítaním skóre v listoch, korešpondujúcich s príkladom.

Na naučenie sa funkcií, použitých v modeli, sa minimalizuje regularizovaná cieľová funkcia (31).

$$L(\theta) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{31}$$
$$kde\ \Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$$

Vo funkcii (31) je l diferencovateľná konvexná chybová funkcia na počítanie rozdielu medzi predikciou $\hat{y}_i$ a reálnou cieľovou hodnotou y_i. Druhá suma slúži ako regularizátor – penalizuje komplexitu modelu pre zabránenie overfittingu. Takto bude model uprednostňovať jednoduchšie funkcie. Podobná, ale menej jednoduchá regularizácia je využitá v Ragularized Greedy Forest (RGF), ktorý dokonca ponúka na výber štyri možné regularizácie [73]. Regularizácia pri gradient boosting ale nie je nutná.

Gradient boosting sa učí iteratívne. Pri iterácii *t* s pre *i*-tý príklad vypočíta predikcia $\hat{y}_i^{(t)}$ nasledovnou minimalizáciou (32):

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i{}^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{32}$$

Takto sa vždy vyberie funkcia, ktorá pre nasledujúce kolo najviac zmenší chybu predikcie. Na zrýchlenie môže byť použitá aproximácia druhého rádu (33).

$$L^{(t)} \cong \sum_{i=1}^{n} [l\left(y_i, \hat{y}_i{}^{(t-1)}\right) + g_i\, f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{33}$$

V tomto vzorci sú premenné $g_i = \partial_{\hat{y}^{t-1}}\, l\left(y_i, \hat{y}_i{}^{(t-1)}\right)$ a $h_i = \partial^2_{\hat{y}^{t-1}}\, l\left(y_i, \hat{y}_i{}^{(t-1)}\right)$ gradienty chybovej funkcie prvého a druhého stupňa.

Keďže regularizačný parameter je pri stromoch rovný $\gamma T + \frac{1}{2}\lambda||w||^2$, možno pre boosting vypočitať optimálnu váhu w_j^* každého listu *j,* kde I_j je množina príkladov v tomto liste (34).

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{34}$$

Z tejto hodnoty možno potom vypočítať skóre celého stromu v danej iterácii (35).

$$L^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{35}$$

Toto skóre sa používa na nájdenie najlepších stromov. Jednotlivé stromy sa skórujú podobným spôsobom ako rozhodovacie stromy v náhodnom lese, kde sa používa metrika nečistoty. Keďže množina všetkých možných stromov pre daný dataset je veľmi veľká, tak sa používa buď greedy algoritmus alebo rýchle aproximačné algoritmy. Greedy algoritmus začína s jedným listom a iteratívne v strome pridáva vetvy. Vyskúša všetky možné rozdelenia pre každý atribút. Pre pridanie vetvy sa hľadá rozdelenie, pri ktorom strom bude mať najlepšie skóre. Skóre rozdelenia sa počíta nasledovne: ak I_L a I_R sú množiny príkladov v ľavej a pravej vetve po rozdelení a I je ich prienik, tak zníženie chyby po rozdelení je vo vzorci (36).

$$L_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{36}$$

Skóre atribútu môže závisieť od toho, koľkokrát bol použitý pri najlepšom rozdelení (split) alebo o sumu zníženia chýb pri rozdeleniach, v ktorých bol použitý (zisk), prípadne priemerné zníženie chyby na rozdelenie.

Pri **rozhodovacích stromoch** sa používa nečistota pri hľadaní najlepšieho rozdelenia. Nečistota v danom uzle stromu hovorí o tom, ako čisté sú jeho vzorky – do akej miery patria len do jednej triedy. Ak je veľká nečistota, tak v uzle sú rovnomerne prítomné vzorky zo všetkých tried. Ak je nulová, všetky vzorky pochádzajú z rovnakej triedy [154]. Existuje viac typov nečistoty. Nečistota entropie množiny S pre počet tried C, kde S_i sú vzorky z triedy i, je vo vzorci (37) [154], kde $p(S_i)$ je pomer veľkosti S_i a celého S v uzle.

$$E(S) = \sum_{i=1}^{C} -p(S_i)\, log_2\big(p(S_i)\big) \tag{37}$$

Ďalšia miera nečistoty je Gini nečistota, čo je očakávané množstvo chýb, ak by sa trieda vybrala náhodne z distribúcie tried vo vzorkách, ktoré sú prítomné v uzle [93]. Vzorec je nasledujúci (38) [154]:

$$G(S) = 1 - \sum_{i=1}^{C} p(S_i)^2 \tag{38}$$

Pri delení uzlu je cieľom, aby nové uzly mali čo najväčší pokles nečistoty (zisk - gain) bez ohľadu na to, aký typ nečistoty sa používa. Test na hľadanie nového rozdelenia sa robí pre každý atribút a uzol. Zisk sa v tomto teste počíta nasledovne (39) [154]:

$$Gain(S) = Impurity(S) - \sum_{i=1}^{N} p(S_i) * Impurity(S_i) \quad (39)$$

N je počet detí, na ktoré sa delí uzol, čo je zvyčajne 2. Pre vybranú hodnotu (prípadne hodnoty) skúšaného atribútu sa rozdelia vzorky v uzle do jeho potomkov na jednotlivé množiny S_i a vypočíta sa pokles. Keďže prvý výraz vo vzorci – nečistota pred rozdelením je pre všetky naraz porovnávané rozdelenia rovnaký, tak sa zvykne vynechávať a potom sa minimalizuje druhá časť. Toto sa robí hlavne, ak je použitá Gini nečistota [154]. Ak sa použije nečistota entropie, tak ziskom je v tom prípade informačný zisk [93]. Problém so ziskom je, že atribúty, ktoré majú viac možných hodnôt, majú väčšiu šancu, že rozdelia uzol na čistých potomkov. Z tohto dôvodu sa používa Gain ratio, kde sa získaná entropia normalizuje entropiou rozdelenia (40) [154]:

$$GR(S) = \frac{Gain(S)}{-\sum_{i=1}^{N} p(S_i) log_2(p(S_i)} \quad (40)$$

Časová zložitosť rozhodovania je *O(d*m*n)* [154] kde *d* je hĺbka stromu, *m* je počet atribútov a *n* je počet vzoriek. Strom potrebuje pri každom rozdelení vypočítať nečistotu pre každý atribút a pre všetky vzorky v uzle. Na jednej úrovni stromu je v uzloch spolu celý dataset. Z tohto dôvodu sa pre každú úroveň skontroluje každý atribút pre každú vzorku. Hĺbka stromu závisí od počtu vzoriek. Ak je strom vybalansovaný, tak má logaritmickú veľkosť vzhľadom na počet vzoriek. V najhoršom prípade (keď sa do jedného potomka dostane len jedna vzorka) je veľkosť stromu rovná počtu vzoriek.

V náhodnom lese potom jednotlivé stromy hlasujú na priradení triedy. Stromy môžu vyberať náhodné množiny atribútov pre svoje trénovanie alebo si priradia náhodné váhy pre trénovaciu množinu. Spoločná myšlienka je, že každý strom by mal byť nezávislý, ale ich náhodné vektory majú mať rovnaké rozdelenie. Pri dostatočnom počte stromov bude celý les konvergovať k spoločnému riešeniu [92].

SVM hľadá v priestore atribútov podmnožinu, ktorá ho rozdelí s najväčšou vzdialenosťou k najbližším vzorkám [93]. Vďaka kernelom vie SVM pracovať aj s dátami, ktoré nie sú lineárne separovateľné [154]. Príkladmi kernelov sú lineárny, polynomiálny, RBF (radial basis function) a sigmoidálny. Základný lineárny SVM pracuje nasledovne [154]: majme dataset *n* príkladov $< \boldsymbol{x}, y >$ kde X má veľkosť *p* –

počet atribútov a *y* je binárne rozdelenie tried (pozitívne a negatívne). Jednotlivé triedy musia byť lineárne separovateľné – existuje lineárna separujúca funkcia $\mathrm{f}(\mathbf{x}) = < \mathbf{w}, \mathbf{x} > +\mathrm{b}$, ktorej pozitivita korešponduje s pozitivitou *y*. **w** sú váhy jednotlivých atribútov. Tam, kde funkcia nadobúda nulovú hodnotu, sa nachádza rozdeľovacia nadrovina. Pre nájdenie najlepšej nadroviny sa použije nasledujúca optimalizácia pre separujúce funkcie (41) [154]:

$$max_{\boldsymbol{w},b} min_{1 \le i \le n} \frac{|< \boldsymbol{w}, \boldsymbol{x}_i > +b|}{\|\boldsymbol{w}\|} \tag{41}$$

Násobok optimálneho riešenia je tiež optimálne riešenie. Z tohto dôvodu sa pre úpravu škály používa minimalizácia len druhej mocniny menovateľa a nie celého zlomku. Táto minimalizácia sa rieši kvadratickým programovaním s lineárnymi obmedzeniami, ktoré je efektívne [154].

V prípade, že dataset nie je lineárne separovateľný, je možné riešenie aj bez kernelu s tým, že sa povolí chybová oblasť a pri optimalizácii sa za jej použitie pridá penalta. Pôvodne mala byť funkcia separujúca: $y_i(< \boldsymbol{w}, \boldsymbol{x}_i > +b \geq 1$. Teraz môže byť od separujúcej posunutá o malú kladnú chybu: $y_i(< \boldsymbol{w}, \boldsymbol{x}_i > +b \geq 1 - \xi_i$. Toto obmedzenie sa použije pri novej optimalizácii, kde sa berie do úvahy aj veľkosť chyby (42).

$$min_{\boldsymbol{w},b,\xi_i} \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \xi_i \tag{42}$$

V tomto vzorci je λ regularizačný trade-off faktor. Určuje, ako veľmi bude penalizácia zložitosti (regularizácia) vplývať na chybu. Teraz je minimalizačný problém v primárnej forme, ale obmedzenia v primárnej forme je ťažšie kontrolovať. Preto sa využíva duálna forma. Langrangeovský duál, derivovaný z tejto primárnej formy, je vyjadrený vo vzorci (43) [154]:

$$min_{\boldsymbol{w}} \frac{1}{2\lambda} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j w_i w_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1}^{n} w_i \tag{43}$$

Tento duál sa stále počíta kvadratickým programovaním, ale už má iné, jednoduchšie obmedzenia: $w_i \in [0, n^{-1}], \sum_{i=1}^{n} y_i w_i = 0$.

Pre *d* atribútov (dimenzií) a *n* vzoriek (bodov) má primárna optimalizácia zložitosť $O(nd^2+d^3)$ a duálna má zložitosť $O(dn^2+n^3)$ [154]. Z tohto dôvodu by sa mala vyberať optimalizácia podľa toho, či je viac atribútov alebo vzoriek. Pri vybratí správnej

optimalizácie je potom zložitosť $O(max(n,d)*min(n,d)^2)$ [154]. Avšak pomocou Woodburyho formuly možno ukázať, že obe metódy majú túto zložitosť bez ohľadu na to, kedy sú použité [62].

SVM rozdeľuje rovinu na nadroviny, takže my použijeme jeho verziu pre viactriednu klasifikáciu – SVC. Tá môže pracovať s dvoma stratégiami. Prvá stratégia je one-versus-rest, keď sa postupne snaží rozdeliť jednotlivé triedy od zvyšku datasetu. V druhej stratégii one-versus-one sa snaží oddeliť postupne dvojice tried.

Vo vzorci (42) sme spomenuli regularizáciu**. Regularizácia** sa dá použiť pre všetky typy lineárnych modelov, nielen pre SVM. Pri týchto modeloch sa za zložitý model považuje ten, ktorý používa veľké váhy. Vo všeobecnosti ℓ_p regularizácia pre p rovné aspoň jedna pridáva penaltu na základe váhy nasledovne (44) [154]:

$$\|\boldsymbol{w}\|_p = \left(\sum_i |w_i|^p\right)^{\frac{1}{p}} \tag{44}$$

SVM vo vzorci (42) prirodzene využíva ℓ_2-normu, ale používa sa aj ℓ_1-norma.

Regularizácia sa používa aj pri selekcii atribútov z dôvodu, že regularizátor spôsobuje, že koeficienty (váhy) mnohých atribútov sú malé alebo až nulové, čo úplne eliminuje dané atribúty [24]. Pri selekcii potom stačí vybrať atribúty, ktoré neboli eliminované alebo atribúty, ktorých koeficient bol nad určitým prahom (viac prispeli do modelu). ℓ_0-norma priamo hľadá optimálnu množinu nenulových atribútov. Optimalizačný problém sa stáva problém celočíselného programovania, ktorý je náročné riešiť (NP úplný problém). z tohto dôvodu sa viac používa ℓ_1-norm regularizácia ktorá je najužšou konvexnou relaxáciou ℓ_0-normy [24].

Pri multi-class klasifikácii (prvok môže patriť do viacerých tried) alebo multivariate regresii sa využívajú $\ell_{p,q}$-normy [24]. V tomto prípade je aj výstup jedného príkladu vektor a nie číslo. To sa odráža aj v norme. $\ell_{p,q}$-norma je definovaná vo vzorci (45), kde C je veľkosť výstupného vektora (počet tried pre multi-class selekciu) a d je počet atribútov [24].

$$\|\boldsymbol{w}\|_{p,q} = \left(\sum_{j=1}^{C}\left(\sum_{i=1}^{d}|w_{i,j}|^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}} \tag{45}$$

Na selekciu atribútov pomocou ℓ_1-normy možno využiť **Lasso** (least absolute shrinkage and selection operator). Lasso uprednostní atribúty, ktoré majú vysokú lineárnu

závislosť na cieľový atribút rozdelenia tried. Jeho nevýhodou je, že nezachytí nelineárnu závislosť. Ak máme dataset *X, y* s *n* príkladmi, kde každý má *d* atribútov, tak metóda Lasso predstavuje optimalizáciu vo vzorci (46) [68], kde $\boldsymbol{w}$ je vektor koeficientov regresie (váhy), $\|.\|_1$ a $\|.\|_2$ sú ℓ_1- a ℓ_2- normy. λ je regularizačný parameter.

$$min_{\boldsymbol{w}\in R^d}\ \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{X}^T\boldsymbol{w}\|_2^2+\lambda\|\boldsymbol{w}\|_1 \tag{46}$$

Lasso produkuje výsledok, v ktorom sú koeficienty irelevantných atribútov často nulové.

Nelineárne **HSIC Lasso** [68] je definované optimalizáciou vo vzorci (47) s obmedzením – vektor $\boldsymbol{w}$ sa skladá len z nezáporných prvkov.

$$min_{\boldsymbol{w}\in R^d}\ \frac{1}{2}\left\|\bar{\boldsymbol{L}}-\sum_{k=1}^{d}w_k\bar{\boldsymbol{K}}^{(k)}\right\|_{Frob}^2+\lambda\|\boldsymbol{w}\|_1 \tag{47}$$

$\|.\|_{Frob}$ je Frobeniova norma čo je $\ell_{2,2}$-norma, $\bar{\boldsymbol{K}}^{(k)}=\boldsymbol{\Gamma}\bar{\boldsymbol{K}}^{(k)}\boldsymbol{\Gamma}$ a $\bar{\boldsymbol{L}}=\boldsymbol{\Gamma}\bar{\boldsymbol{L}}\boldsymbol{\Gamma}$ sú centrované Gram matice , $K_{i,j}^{(k)}=K(x_{k,i},x_{k,j})$ a $L_{i,j}=L(y_i,y_j)$. *K(x,x')* a *L(y,y')* sú kernel funkcie. $\boldsymbol{\Gamma}=\boldsymbol{I}_n-\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$ je centrovacia matica, $\boldsymbol{I}_n$ je *n*-rozmerná identická matica a $\mathbf{1}_n$ je jednotkový vektor dĺžky *n*. Gram (Gramian) matica obsahuje skalárne súčiny postupnosti vektorov [94].

HSIC Lasso používa stratégiu selekcie nazvanú „minimálna redundancia, maximálna relevancia“, ktorá vyberá atribúty, majúce vysokú závislosť od triedy, ale nízku závislosť od ostatných atribútov. Táto stratégia je použitá v mRMR selekcii. To, že využíva túto stratégiu, uvidíme, keď sa prepíše prvá časť rovnice (47) do tvaru rovnice (48).

$$min_{\boldsymbol{w}\in R^d}\ \frac{1}{2}\left\|\bar{\boldsymbol{L}}-\sum_{k=1}^{d}w_k\bar{\boldsymbol{K}}^{(k)}\right\|_{Frob}^2=$$
$$\frac{1}{2}HSIC(\boldsymbol{y},\boldsymbol{y})-\sum_{k=1}^{d}w_kHSIC(\boldsymbol{u}_k,\boldsymbol{y})+\frac{1}{2}\sum_{k=1}^{d}\sum_{l=1}^{d}w_k\,w_lHSIC(\boldsymbol{u}_k,\boldsymbol{u}_l) \tag{48}$$

V tejto rovnici $HSIC(\boldsymbol{u}_k,\boldsymbol{y})=tr\,(\bar{\boldsymbol{K}}^{(k)}\bar{\boldsymbol{L}})$ je meradlo nezávislosti založené na kerneli, ktoré sa označuje Hilbert-Schmidt independence criterion (HSIC) a *tr* znamená trace (suma elementov na hlavnej diagonále). $HSIC(\boldsymbol{y},\boldsymbol{y})$ je konštanta. *HSIC* môže mať len kladné hodnoty a je nulový len v prípade, keď sú dva atribúty štatisticky nezávislé. Keď má atribút $\boldsymbol{u}_k$ veľkú závislosť na rozdelení tried $\boldsymbol{y}$, tak $HSIC(\boldsymbol{u}_k,\boldsymbol{y})$ má tiež vysokú hodnotu. To spôsobí, že w_k bude mať tiež vysokú hodnotu, aby sa celý výraz

minimalizoval. V opačnom prípade (nezávislosť) bude hodnota $HSIC(\boldsymbol{u}_k, \boldsymbol{y})$ blízka nule a w_k bude vo všeobecnosti eliminované regularizácou. Po regularizácii ostanú len relevantné atribúty, ktoré sú závislé na výstupnom atribúte. Podobne $HSIC(\boldsymbol{u}_k, \boldsymbol{u}_l)$ má veľkú hodnotu pri závislých atribútoch, čo znamená, že jeden z nich je redundantný. Keďže tu sa výraz pripočítava, tak $w_k w_l$ bude veľmi malé kvôli minimalizácii celého výrazu. Vo všeobecnosti jeden z $w_{k,} w_l$ nadobudne nulovú hodnotu, teda redundantné atribúty sú eliminované.

HSIC lasso má pamäťovú zložitosť $O(d*n^2)$, kde d je počet atribútov a n je počet vzoriek. HSIC lasso má aj blokovú verziu, kde sa atribúty diskretizujú do B blokov po ich štandardizácii, čím sa zmenší pamäťová zložitosť na $O(d*n+B)$. [68]. S použitím Nyströmovej aproximácie bude pamäťová zložitosť rovná $O(d*b*n)$, kde b určuje presnosť aproximácie. Časová zložitosť vypočítania jadra (kernelu) a jeho násobenia je potom $O(d*b*n+m*b^2d*n) = O(m*b^2d*n)$, kde m je počet vybraných parametrov. Pri naivnej verzii by bola zložitosť $O(m*d*n^3)$ s pamäťou $O(d*n^2)$ [67].

Pri trénovaní pre všetky klasifikátory používame krosvalidáciu, pri ktorej sa rozdelí testovací dataset na k skupín a postupne sa vystriedajú všetky kombinácie $k-1$ skupín pri trénovaní. Skupina, ktorá sa nevybrala, slúži ako testovacia sada. Z nej sa získa výsledok. Takto dostaneme lepší odhad presnosti klasifikátorov a zabránime, aby došlo pri trénovaní k overfittingu [61], teda stavu, keď je klasifikátor príliš upravený pre konkrétny dataset a zle klasifikuje ostatné datasety pre rovnakú úlohu. Pri krosvalidácii využívame stratifikáciu – jednotlivé skupiny majú rovnaké rozdelenie tried ako pôvodný dataset. Ako metriku používame presnosť (accuracy). Pôvodne sme chceli použiť ROC-AUC krivku, ale tá je pre binárnu klasifikáciu a my máme veľa tried. Presnosť je v nami analyzovaných článkoch o klasifikácii malvéru najčastejšia metrika, a preto budeme vedieť porovnať naše výsledky s ostatnými prácami. Naším výsledkom je priemerná presnosť pre 10-fold krosvalidáciu.

3 Implementácia a postup

Celý postup sme robili nad datasetom malvéru, ktorý bol v rokoch 2017 a 2018, získaný zo služby VirusTotal [90] cez ich aplikačné rozhranie. Dataset preto okrem malvéru tvoria aj reporty zo služby VirusTotal, ktoré obsahujú ohodnotenie vzoriek malvéru antivírusovými programami, hash súboru a počet pozitívnych ohodnotení. Keďže bolo použité premium aplikačné rozhranie, report obsahuje aj výstup z nástroja exiftool [70] (metadáta), importy, exporty, informácie o sekciách aj s entropiou, zdrojoch a overlay, výstup z IDS (Intrusion detection system – systém na detekciu prienikov) a sandboxu (otvorené porty, navštívené adresy, vytvorené procesy, manipulácia s registrami a súbormi, načítané knižnice).V tomto datasete je 90355 vzoriek malvéru, ktoré majú veľkosť 109 GB.

3.1 Tvorba datasetu

V pôvodnom datasete je príliš veľa vzoriek na to, aby sme zvládli spracovať všetky. Z tohto dôvodu sme najprv dataset vyčistili. Dataset sme rozdelili na vzorky a reporty a očistili dataset o neúplné údaje. Po tomto kroku sme mali 90355 vzoriek malvéru aj reportov. Potom sme vymazali vzorky, ktorých veľkosť je nad 5 MB. Ostalo nám 85501 vzoriek, ktorých veľkosť bola už len 60 GB. Väčšina vzoriek malvéru, až 84452 boli PE spustiteľné súbory. Keďže jeden typ takto prevažoval, rozhodli sme sa zamerať len na nich, čo nám umožnilo využiť atribúty, ktoré sa viažu na PE súbory. Ostatné vzorky, ktoré boli prevažne označené ako MS DOS typ, sme vymazali.

Potom sme sa zamerali na reporty. Najprv sme odstránili reporty, ktoré neboli kompletné – nebol pri nich použitý exiftool [70], nemali informácie o sekciách alebo o importoch. Predpokladáme, že tieto súbory museli byť poškodené, keďže sa z nich nedali získať tieto informácie. Po tomto kroku nám ostalo 79856 vzoriek. Nasledoval výber antivírusových programov (AV), ktoré neskôr slúžili na tvorbu označení (labels). Pri výbere sme sa sústredili na to, aby vybrané AV používali podobné označenia vzoriek, keďže tieto označenia potom zhlukujeme. Empiricky sme vybrali antivírusové programy od spoločností Kaspersky, Microsoft, McAfee, Eset, BitDefender a Malwarebytes. Najprv sme zistili, koľko vzoriek malvéru neoznačili ako malvér a v koľkých vzorkách sa nevyskytli hodnotenia týchto AV. Počty sú v tabuľke (Tab. 1) (prvé číslo je počet false

hodnotení, druhé je počet vzoriek, v ktorých sa hodnotenie daného AV nevyskytovalo). Hodnotenie false znamená, že daný antivírusový program nepovažoval vzorku za malvér.

Tab. 1 Počty hodnotení, ktoré nepoukazujú na malvér.

Antivírusová ochrana	False	Nevyskytuje sa	Spolu
Kaspersky	10145	366	10551
Microsoft	18951	215	19166
McAfee	4628	203	4831
Eset	6008	39	6047
BitDefender	5097	1009	6106
Malwarebytes	38499	3128	41627

AV Microsoft a Malwarebytes sme sa rozhodli vylúčiť, lebo označili príliš veľa vzoriek malvéru ako negatívnych. Pre zvyšné štyri AV sme odstránili všetky vzorky, ktoré označili ako negatívne alebo pri nich neboli všetky štyri AV použité. Po vyššie uvedených úpravách nám ostalo 63428 vzoriek malvéru. Pôvodne sme odstraňovali aj vzorky, ktoré mali viac negatívnych ako pozitívnych označení pre všetky AV vo VirusTotal, ale vylúčilo sa nám príliš veľa vzoriek malvéru. Z tohto dôvodu sme sa sústredili len na vybrané AV.

Následne sme rozdelili dataset na tri časti. V prvej časti sme sa snažili odstrániť zabalené, zašifrované a obfuskované vzorky. Na zabalené vzorky bola aplikovaná komprimačná funkcia. Na to, aby sa vzorka dostala do pôvodného stavu, sa musí použiť funkcia, ktorá je reverzná ku komprimačnej. Obfuskácia v tomto kontexte je snaha urobiť strojový kód, získaný po disasemblovaní vzorky ťažko pochopiteľným. V druhej časti sme nechali všetky vzorky (63428) – zmiešaný dataset. V treťom datasete sme si nechali len vzorky, ktoré sme v prvom odstránili. V prvom datasete sme v reportoch hľadali reťazce „packe“, „obfuscat“, „encrypt“ na nájdenie zabalených, obfuskovaných a zašifrovaných vzoriek. To nám znížilo počet vzoriek na 40207. Potom sme cez sigcheck [158] skontrolovali ich entropiu a vylúčili všetky, ktoré ju mali nad 6.66. Bližšie sme sa entropii venovali v predchádzajúcej kapitole. To nám odstránilo približne polovicu vzoriek malvéru. Napokon nám ostalo 23595 vzoriek. V treťom datasete sme vylúčili tie vzorky malvéru, ktorých reporty neobsahovali reťazce „packer“, „obfuscat“,

„encrypted“, „packed“, „unpack“. Ostalo nám 23452 vzoriek malvéru. V nasledujúcich kapitolách si popíšeme jednotlivé kroky analýzy vzoriek, ktoré sú rovnaké pre všetky tri datasety.

3.1.1 Označovanie

Pre každý zo štyroch vybraných AV sme zistili, do akých tried zaradzuje vzorku malvéru. Každé označenie vzorky sa skladá z jej hierarchického názvu, ktorý, žiaľ nie je štandardizovaný. Vo všeobecnosti je na začiatku platforma a typ malvéru, potom trieda a prípona. Prípon môže byť niekoľko. V názve sa môže vyskytovať aj predpona pred triedou a podtrieda. Zvyčajne nie sú v názve všetky vymenované časti, ale hierarchia je zachovaná. Príkladom je označenie „Trojan-Ransom.Win32.Locky.wtk“ ktoré obsahuje typ, plaformu, triedu a príponu. Niekedy je ťažké rozlíšiť, ktorá časť názvu je trieda. Príkladom je „W32/Bifrost.M.gen!Eldorado“. Časti názvov sa oddeľujú prevažne bodkou, ale používajú sa aj iné znaky (príkladom je označenie „Ransom:Win32/Locky.A.“). My sme sa po pozorovaní názvov viacerých vzoriek rozhodli použiť tieto znaky: _ :/.!,-. Následne sme oddelili každý výstup, zmenili veľké písmená na malé a pre každý AV osobitne sme spočítali, koľkokrát sa jednotlivé oddelené slová nachádzajú v datasetoch. Hneď na začiatku sme vylúčili slová, ktoré boli kratšie ako štyri znaky, lebo tak krátke boli zvyčajne prípony a predpony. Vylúčili sme aj reťazce, ktoré sa vyskytovali v menej ako päťdesiatich vzorkách. To sme vykonali najmä z dôvodu, že takéto triedy by boli príliš malé v porovnaní s ostatnými. Navyše tieto slová aj tak pravdepodobne boli podtriedy alebo ešte jemnejšie rozdelenie.

Následne sme si vypísali najčastejšie slová a vylúčili sme všetky, ktoré boli príliš generické. Tieto slová sa vyskytujú pred samotnou triedou v názve vzorky od AV, prípadne sa vyskytujú namiesto nej, ak AV nevedel zaradiť vzorku do triedy. Medzi triedami tieto slová nechceme, keďže by sa nám zlúčili vzorky rovnakého typu, ale z rôznych tried do jednej, príliš všeobecnej triedy. Zoznam týchto generických slov je v tabuľke (Tab. 2).

Tab. 2 Zoznam generických slov v názvoch vzoriek.

win32	variant	spyware	trojan	worm
virus	heur	trojandownloader	generic	malware

ransom	ransomware	trojandropper	autorun	agent
adware	downloader	infector	backdoor	hacktool
bitcoinminer	win64	blocker	proxy	email
injector	Inject	softwarebundler	virtool	ddos
exploit	filecoder	dropper	sitehijack	cryptolocker
application	deepscan	keylogger	obfuscated	packed
encrypted	Packer	obfuscator	encryptor	malpack
startpage	servstart	infostealer	crypt	rootkit
passwordstealer	optional	genpack		

Okrem toho sme zlúčili tri aliasy: „wannacry“, „wannacrypt“, „wannacryptor“ do jednej triedy. Urobili sme to preto, aby nám nevzniklo viacero tried, v ktorých sú vzorky, ktoré patria do jednej rodiny. Následne sme hľadali prienik v AV medzi triedami, ktoré ostali. To sme vykonali najmä z dôvodu, že nám stále ostali triedy, ktoré používal len jeden AV. Pre dataset bez obfuskovaných, šifrovaných a zabalených vzoriek sme brali zhodu dvoch AV zo štyroch. Pre druhý dataset sme chceli aspoň tri zhodné názvy, keďže pri nich je väčšia šanca, že AV ich označí nesprávne. Pri treťom datasete sme naopak brali do úvahy len dve zhody, ináč nám ostalo málo tried. Ak existovala vzorka, pri ktorej sa aspoň daný počet AV zhodol na jednej zo zostávajúcich tried, tak sme túto triedu uložili medzi finálne. Keď sme mali finálne triedy, tak sme znovu skontrolovali všetky vzorky. Keď sa v aspoň jednom AV vyskytovala jedna z týchto tried, tak sme ju označili za triedu danej vzorky. To, čo sme robili, nie je úplný konsenzus, keďže sme požadovali len to, aby sa daná trieda vyskytovala vo viacerých AV v ľubovoľnej vzorke, nie v každej. Toto sme robili najmä z dôvodu, že v niektorých triedach bolo veľmi málo vzoriek, ktoré by prešli konsenzom a pri jeho použití by sme museli tieto triedy ignorovať. Pre prvý dataset nám vzniklo 10 tried, pre druhý a tretí dataset bolo tried 8. Rozdelenie tried je v tabuľke (Tab. 3).

Tab. 3 Rozdelenie tried datasetov.

Prvý dataset		Druhý dataset		Tretí dataset	
crytex	1135	sality	1627	allaple	828
msil	917	upatre	1118	msil	576

upatre	895	zbot	962	vbkrypt	479
zbot	553	dealply	649	nsis	331
sality	519	expiro	616	sality	302
vbkrypt	491	locky	488	dealply	221
neshta	487	wannacry	210	parite	199
runouce	326	linkury	181	hupigon	106
linkury	96	spolu	5851	spolu	3042
wannacry	86				
spolu	5506				

Keďže ako metriku používame presnosť, nie je dobré mať príliš veľké rozdiely vo veľkosti tried. Môže sa stať, že klasifikátor sa rozhodne príliš sa zamerať na najväčšiu triedu a najmenšiu zväčša ignoruje, lebo z najväčšej triedy budú najväčšie zisky v presnosti, kým rozdelením najmenšej sa získa minimum. Preto sme v prvom datasete obmedzili veľkosť triedy na 100, v druhom na 200 a v treťom na 120. V prvom datasete nám teraz ostalo 982, v druhom 1581 a v treťom 946 vzoriek. Pri extrakcii atribútov, ktorú sme robili po tomto kroku, sme niektoré vzorky museli vylúčiť. Dôvod je popísaný v kapitole 3.1.2. V prvom datasete sme vylúčili 13, v druhom 22 a v treťom 29 vzoriek. V najmenšej triede zostal rovnaký počet vzoriek v prvom aj druhom datasete, takže nerovnosť medzi triedami sa ešte znížila. V treťom datasete sa zmenšila aj najnižšia trieda na 92 vzoriek. Teraz majú datasety 969, 1559 a 917 vzoriek. Toto sú finálne veľkosti datasetov pre označovanie konsenzom. Teraz sme skontrolovali, koľko potenciálne zabalených, obfuskovaných alebo zašifrovaných vzoriek ostalo v druhom datasete – konkrétne sme počítali v koľkých vzorkách sa vo VT reportoch vyskytli reťazce „packer“, „obfuscat“, „encrypted“, „packed“, „unpack“. Ukázalo sa, že takýchto vzoriek ostalo 330 – približne 21% datasetu.

Potom sme skúsili názvy týchto filtrovaných tried zlučovať cez Levenshtein distance, aby sme spojili triedy, ktoré v rôznych AV majú len drobné odchýlky. Výsledky boli veľmi nepresné,. Najprv sa začali zlučovať úplne nesúvisiace triedy, ktoré mali štyri znaky, čo bolo naše minimum. Až pri vyšších hodnotách sa zlúčili triedy, ktoré sa mali zlúčiť. Problém bol v tom, že dĺžka názvov jednotlivých tried bola príliš odlišná.

Náš predpoklad bol, že vzorky s podobnými označeniami budú mať podobnú aj časť pred triedou, ako sme spomínali v predošlej kapitole. Problém s generickou triedou nemáme, keďže zhlukovanie robíme na datasetoch, ktoré prešli výberom konsenzom a teda ich vzorky neobsahujú generické názvy. Najprv sme vyskúšali na celých názvoch (aj s predponami a príponami) identitu. Hľadali sme vzorky, ktoré majú úplne rovnaký celý názov v aspoň dvoch rôznych AV, ale ostali nám len dve triedy s viac ako 50 vzorkami. Z tohto dôvodu sme totožnosť zavrhli. Potom sme zhlukovali tieto celé názvy cez HDBSCAN. Robili sme to tak, že celé označenia so všetkých štyroch AV sme spojili do jedného slova a Levenshtein distance medzi týmito slovami slúžila ako vzdialenosť. Maticu vzdialenosti sme dali na vstup pre HDBSCAN.

Označovanie cez zhlukovanie robíme až na datasetoch, na ktorých boli vylúčené vzorky, ktoré neprešli konsenzom. Robíme to preto, lebo pôvodné datasety majú rádovo desaťtisíce vzoriek a robiť na tak veľkom datasete zhlukovanie cez Levenshtein distance na slovách, ktoré majú zvyčajne nad sto znakov, je výpočtovo príliš náročné, pretože časová zložitosť Levenshtein distance pre dve slová dĺžky n a m je $O(n*m)$. Pre celý dataset veľkosti d je potom potrebné vypočítať vzdialenosť pre všetky páry, čím sa dostaneme k zložitosti $O(d^2*n^2)$ kde n je dĺžka najdlhšieho slova.

Minimálnu veľkosť zhluku sme v prvom datasete zadali 86, čo je veľkosť najmenšej triedy v predošlom rozdelení. Metóda HDBSCAN vylúčila 57 vzoriek ako outliere (vzorky, ktoré nepatria ani do jedného zhluku). Ostalo ich 912. Zvyšné vzorky metóda zhlukovala do 9 tried. Početnosť tried bola nasledujúca: 128, 114, 102, 101, 100, 100, 95, 86, 86.

V druhom datasete sme za minimálnu veľkosť zhluku zadali 181 – veľkosť najmenšej triedy. Outlierov bolo až 266 – ostalo 1293 vzoriek. Vzorky sa zlúčili len do 5 tried s týmito početnosťami: 341, 340, 215, 199, 198.

V treťom datasete mala najmenšia trieda 92 vzoriek. Po tomto obmedzení veľkosti sme dostali 5 tried s veľkosťami: 247, 212, 141, 120, 102. Outlierov bolo 95, takže dataset sa zmenšil na 822 vzoriek.

Dôvod, prečo vzniklo menej tried, môže byť ten, že niektoré vzorky mali pre rôzne AV rôzne názvy, ale keď sme názvy z rôznych AV spájali namiesto porovnávania, tak tieto triedy sa zlúčili. Príkladom môžu byť vzorky, ktoré boli označené v pôvodných triedach ako runounce alebo b@mm. V pôvodných triedach cez testy (nad 50 vzoriek a 2,

resp. 3 AV) prešlo len runounce. V realite tieto dve triedy predstavujú rovnakú triedu. Rôzne AV ju označujú takto [156]:

- Eset: Win32_Chir.B,
- Kaspersky: Email-Worm.Win32.Runouce.b,
- McAfee: W32/Chir.b@MM.virus,
- Microsoft: Virus:Win32/Chir.B@mm,
- Symantec: W32.Chir.B@mm.

Keď v nových triedach spojíme názvy pre dve vzorky v rovnakej triede, vzniknú nám tieto mená (mriežka je použitá na oddelenie názvov rôznych AV):

Email-Worm.Win32.Runouce.b#W32/Chir.b@MM#Win32/Virut.O#Win32.Runouce.B@mm

Email-Worm.Win32.Runouce.b#W32/Chir.b@MM#Win32/Virut.NBP#Win32.Runouce.B@mm

Ako je možné vidieť, rozdiel je len v prípone v treťom AV (Virut.O a Virut.NBP). Tento dokonca označil vzorku ako „virut“, čo v pôvodných triedach predstavovalo jednu samostatnú triedu, aj keď tu je to pravdepodobne nesprávna klasifikácia. V tomto prípade vzorky, ktoré by dostali od dvoch AV označenia „runounce“, od dvoch „b@mm“ a od jedného „virut“, teraz patria do jedného zhluku.

Pri tomto datasete sme vyžadovali zhodu triedy pre dve AV zo štyroch. Ďalšie dve mohli mať iné názvy. Outlier mohol vzniknúť, ak názvy ostatných dvoch neboli konzistentné. Inkonzistencia môže vzniknúť pri nesprávnej klasifikácii. Ak jeden z AV, ktoré sa nemuseli zhodovať, takmer vždy priradil vzorke rovnakú triedu, tak po spojení označení zo všetkých AV malé množstvo vzoriek, ktoré mali iný názov, bude mimo klastra – vznikne outlier. Toto je ďalšia výhoda zhlukovania. Neprekáža, ak rôzne AV používajú rôzne označenia. Stačí, že navzájom sú ich označenia konzistentné. Ak nie sú konzistentné, tak pravdepodobne došlo k nesprávnemu označeniu vzorky. Tento stav predošlým prístupom nevieme odhaliť.

3.1.2 Extrakcia dát

Po tom, čo sme vyčistili dataset, sme zo vzoriek malvéru začali získavať dáta. Skôr, ako sme mohli získavať samotné atribúty, sme potrebovali zo vzoriek získať čo najviac informácií. Na získanie hexadecimálnej reprezentácie vzoriek sme použili hexdump [74], keďže jeho výstup mal menšiu veľkosť ako výstupy ostatných programov (umožňuje výstup bez ASCII prekladu aj offsetu).

Na získanie disasemblovaného súboru sme použili objdump [78], keďže umožňuje disasemblovať všetky sekcie, nielen tie, v ktorých sa očakáva kód. To sa dá využiť pri vzorkách s premenovanými sekciami a droppermi (malvér uložený v časti programu resources). Pri niektorých vzorkách zobrazoval objdump chybu „Section flag STYP_COPY (0x10) ignored“ a mal na nich prázdny výstup. Vo veľmi malom počte prípadov skončil s chybou „File format not recognized“. Pri niekoľkých vzorkách vyčerpal pamäťové prostriedky, čo sa môže stať pri rekurzii (napr. pri demanglingu [78]).

Z tohto dôvodu sme vyskúšali nástroj radare2 [79]. Radare2 za normálnych okolností otvorí vlastný terminál, v ktorom možno používať príkazy programu na súbore. Na druhej strane nemá žiadne príkazy pre spracovanie viacerých súborov a z vonkajšieho systémového terminálu sa nám nepodarilo zadať príkazy do jeho terminálu. Na prácu s viac súbormi (batch) je možné použiť nástroj r2pipe [157], vďaka čomu možno príkazy pre Radare2 zadávať priamo cez programovací jazyk. R2pipe podporuje množstvo jazykov. My sme použili verziu pre Python. Výstupy boli príliš veľké (230 MB oproti 55 MB pre objdump na 5 MB súbore) a spracovanie bolo o dosť pomalšie. Z tohto dôvodu sme sa rozhodli nepoužiť Radare2 na celom datasete. Keďže radare2 nám nevyhovoval, použili sme objdump a vzorky, na ktorých dal prázdny výstup, sme vylúčili z datasetu. Neskôr (pri treťom datasete) sme zistili, že do niektorých vzoriek zapíše len prvý úvodný riadok. Z tohto dôvodu sme odstránili všetky vzorky malvéru, ktorých disasemblovaný súbor mal na disku veľkosť do štyroch kilobajtov, čo je veľkosť jednej stránky a teda minimálna veľkosť neprázdneho súboru. Potom sme skontrolovali prvé dve datasety na takéto súbory. Prvý dataset mal len jednu takú vzorku, druhý ich mal 15. Kvôli objdump chybám sme odstránili 13 vzoriek z prvého datasetu, 22 z druhého datasetu a 29 z tretieho datasetu. Týmto vzorkám sme sa bližšie venovali v predchádzajúcej kapitole.

Textové reťazce sme získal prostredníctvom nástroja Strings [103], pričom sme brali len reťazce s minimálne 5 znakmi. Entropiu sme získali rovnakým spôsobom ako v prípade čistenia prvého datasetu, pomocou nástroja Sigcheck [112].

3.2 Extrakcia atribútov

Počas extrakcie sme vykonávali aj prvotnú selekciu, keďže sa extrahovalo príliš veľké množstvo atribútov a nechceli sme všetky ukladať a znovu spracovávať pri selekcii. Bližšie máme túto selekciu popísanú v nasledujúcej podkapitole. V rámci tejto kapitoly si popíšeme, koľko atribútov sa nachádzalo v jednotlivých skupinách a koľko z nich neprešlo prvotnou selekciou. Po extrakcii atribútov sme rozdelili v každom datasete atribúty na tri skupiny:

- všetky atribúty datasetu,
- tie atribúty, ktoré pochádzajú zo skupín, majúcich do tisíc atribútov,
- tie atribúty, ktorých skupiny majú len do sto atribútov.

Na jednotlivých skupinách budeme robiť selekciu a klasifikáciu. Takto zistíme, či postačujú aj malé skupiny atribútov.

3.2.1 Prvý dataset

Atribúty, získané v tabuľke, popisujeme v tabuľke (Tab. 4). V tabuľke uvádzame počet extrahovaných atribútov v jednotlivých skupinách, aj ich počet po prvotnej selekcii, ktorú predstavujú metódy Document frequency (DF) a Low variance selection (selekcia nízkou varianciou).

Tab. 4 Atribúty, extrahované z prvého datasetu.

Názov skupiny atribútov	Počet atribútov	Po prvotnej selekcii
informácie o importovaných knižniciach	172	40
informácie o importovaných funkciách	6424	457
informácie o exportoch	103	1
metadáta	11	10
entropia spustiteľného súboru	1	1
informácie o overlay dátach	3	3
informácie o sekciách	56	47

informácie o zdrojoch (resources)	48	19
existencia 1-gramov textových reťazcov	419217	964
frekvencia 1-gramov textových reťazcov	419217	9245
existencia 2-gramov textových reťazcov	647349	304
frekvencia 2-gramov textových reťazcov	647349	13792
existencia 3-gramov textových reťazcov	741590	188
frekvencia 3-gramov textových reťazcov	741590	15712
existencia 1-gramov písmen v textových reťazcoch	97	94
frekvencia 1-gramov písmen v textových reťazcoch	97	94
existencia 2-gramov písmen v textových reťazcoch	9408	6
frekvencia 2-gramov písmen v textových reťazcoch	9408	89
existencia 3-gramov písmen v textových reťazcoch	641597	2
frekvencia 3-gramov písmen v textových reťazcoch	641597	13
informácie o textových reťazcoch	42	36
existencia 1-gramov hexadecimálnej reprezentácie vzorky	256	65
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	256	256
normalizovaná frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	256	256
existencia 2-gramov hexadecimálnej reprezentácie vzorky	65536	65026
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	65536	65536
normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	65536	65536
existencia 3-gramov hexadecimálnej reprezentácie vzorky	14004842	4620815 – nedokončilo sa
frekvencia 3-gramov hexadecimálnej reprezentácie vzorky	14004842	Nerobilo sa

normalizovaná frekvencia 3-gramov hexadecimálnej reprezentácie vzorky	14004842	Nerobilo sa
histogram entropie bajtov	256	256
pomery veľkostí extrahovaných súborov	9	7
existencia 1-gramov operačných kódov	5388075	29701
frekvencia 1-gramov operačných kódov	5388075	92089
existencia 1-gramov bajtov operačných kódov	256	82
frekvencia 1-gramov bajtov operačných kódov	256	256
normalizovaná frekvencia 1-gramov bajtov operačných kódov	256	256
existencia 2-gramov bajtov operačných kódov	65536	65412
frekvencia 2-gramov bajtov operačných kódov	65536	65536
normalizovaná frekvencia 2-gramov bajtov operačných kódov	65536	65536
existencia 3-gramov bajtov operačných kódov	13432761	2812295 – nedokončilo sa
frekvencia 3-gramov bajtov operačných kódov	13432761	Nerobilo sa
normalizovaná frekvencia 3-gramov bajtov operačných kódov	13432761	Nerobilo sa
existencia 1-gramov inštrukcií	1076	838
frekvencia 1-gramov inštrukcií	1076	881
existencia 2-gramov inštrukcií	116270	21603
frekvencia 2-gramov inštrukcií	116270	38257
existencia 3-gramov inštrukcií	2184555	73257
frekvencia 3-gramov inštrukcií	2184555	219712
existencia 1-gramov registrov	108	75
frekvencia 1-gramov registrov	108	100
existencia 2-gramov registrov	3924	2359
frekvencia 2-gramov registrov	3924	2737

existencia 3-gramov registrov	76520	19723
frekvencia 3-gramov registrov	76520	30027
informácie o inštrukciách a registroch	7	6
informácie o disasemblovanom súbore	115	106

Z metadát bol po použití selekcie varianciou odstránený len atribút MachineType. To znamená, že väčšina vzoriek beží na rovnakom type procesora. Z exportov nám po selekcii DF ostal len atribút ich počtu. Vzhľadom na veľkosť tried a hranicu pre DF to znamená, že ani vzorky v rovnakých triedach nemajú spoločné exporty.

Zo sekcie programu .edata nám vypadli všetky atribúty, čo znamená, že daná sekcia sa nezvykne pri malvéri používať. Vo viacerých sekciách nám vypadol pomer ich veľkosti k veľkosti súboru (ich veľkosť nemala odchýlky) a entropia. V sekcii programu .bss vypadla aj samotná veľkosť. Jediné sekcie programov, v ktorých ostali všetky dáta, sú .text, .rsrc a .reloc. Dôvod, prečo sa vylúčil pomer veľkostí sekcií k veľkosti súboru, je ten, že v daných sekciách malo len malé množstvo vzoriek nenulovú veľkosť. Veľkosť ostatných sekcií bola veľmi malá a preto sa jej pomer k veľkosti súboru blížil nule, takže pri výpočte variancie splýval s väčšinovou nulovou hodnotou. Z tohto dôvodu sme sa rozhodli, že budeme deliť veľkosť súboru s veľkosťou sekcií. Pomer medzi sekciami sa zachová, len veľkosti budú vymenené. Po týchto úpravách nám ostalo 47 atribútov (predtým 42). Takýto obrátený pomer používame aj pri ostatných atribútoch, pri ktorých sa delí veľkosťou súboru.

Pri extrahovaní informácií o textových reťazcoch sme chceli mať pre každú použitú dĺžku reťazca atribút a počítať, koľkokrát bola daná dĺžka dosiahnutá. Tieto atribúty sme sa nakoniec rozhodli neextrahovať, keďže maximálna dĺžka reťazca bola 979417 znakov, čo je príliš veľké množstvo. Namiesto toho iba sledujeme, do ktorého určeného intervalu veľkosť reťazca spadá. Intervaly sme rozdelili najprv od päť po desať po jednom čísle (reťazce s dĺžkou menšou ako päť sme neextrahovali). Potom tvoríme intervaly po desiatkach po stovku, potom po stovkách do tisícky a po tisícoch do desaťtisíc. Od desaťtisíc nahor je všetko v jednom intervale. Niektoré intervaly dĺžky reťazcov vypadli po prvotnej selekcii, hlavne tie najväčšie – v tisícoch (napr. tisíc až dvetisíc...).

Z 3-gramov hexadecimálnej reprezentácie súboru po selekcii DF ostalo 4620815 3-gramov, čo je príliš veľa pre počítanie variancie – pri 28 B veľkosti Python integer to je

125 GB dát pre celú maticu datasetu. Aj keď sme sprísnili hranicu DF na 20, tak stále ostalo 2890723 3-gramov, čo je stále príliš veľa. Z tohto dôvodu sme sa rozhodli, že tieto 3-gramy vylúčime. Pri 3-gramoch bajtov operačných kódov po DF ostalo 2812295 3-gramov, čo je stále príliš veľa a preto sme aj tieto 3-gramy vynechali. Tiež sme vzhľadom na enormné množstvo 1-gramov operačných kódov (5388075) (opcode je 1-gram) 2-gramy neextrahovali.

Pri histograme entropie bajtov sme vyskúšali rôzne veľkosti okien a krokov: 512 a 2048, 1024 a 2048, 256 a 1024, a pri žiadnej prvotná selekcia nič nevylúčila, ostalo všetkých 256 atribútov. Rozhodli sme sa ostať pri veľkostiach 512 a 2048.

Celkovo z 20 861 316 atribútov (s tým, že sme ďalších asi 82 312 809 úplne vylúčili – príliš veľké skupiny) ostalo 966 466 atribútov. Toto množstvo bolo stále príliš veľké. Z tohto dôvodu sme sa rozhodli vylúčiť všetky 3-gramy aj 1-gramy operačných kódov, ktorých bolo na začiatku 5 388 075, čo je príliš veľké množstvo. Po úpravách nám ostalo 486 136 atribútov. Po rozdelení na skupiny (zlúčenie skupín atribútov do tisíc položiek a do sto položiek) nám ostalo 5567 atribútov v druhej skupine a 766 v tretej skupine. Prvá skupina predstavuje všetky atribúty.

3.2.2 Druhý a tretí dataset

V druhom a treťom datasete sme už vôbec neextrahovali skupiny atribútov ktoré sme sa v prvom datasete rozhodli vylúčiť – všetky 3-gramy a n-gramy operačných kódov, kde celý operačný kód je 1-gram. Rovnako ako v prvom datasete sme aj v ostatných datasetoch rozdelili dĺžky textových reťazcov do intervalov. Počty extrahovaných skupín atribútov sú popísané v tabuľke (Tab. 5). V tabuľke sú popísané atribúty pre obe datasety – najprv sú uvedené počty extrahovaných skupín atribútov a počty po prvotnej selekcii pre druhý dataset, potom sú uvedené rovnaké údaje pre tretí dataset.

Tab. 5 Atribúty extrahované z druhého a tretieho datasetu.

Názov skupiny atribútov	Počet – 2. dataset	Po selekcii – 2. dataset	Počet – 3. dataset	Po selekcii – 3. dataset
informácie o importovaných knižniciach	312	51	127	44

informácie o importovaných funkciách	7752	645	3395	495
informácie o exportoch	518	1	78	1
metadáta	11	11	11	11
entropia spustiteľného súboru	1	1	1	1
informácie o overlay dátach	3	3	3	3
informácie o sekciách	56	52	56	53
informácie o zdrojoch (resources)	114	21	40	14
existencia 1-gramov textových reťazcov	1987269	3762	1438390	561
frekvencia 1-gramov textových reťazcov	1987269	11733	1438390	1960
existencia 2-gramov textových reťazcov	2576514	3437	1672343	221
frekvencia 2-gramov textových reťazcov	2576514	16290	1672343	858
existencia 1-gramov písmen v textových reťazcoch	97	94	97	94
frekvencia 1-gramov písmen v textových reťazcoch	97	94	97	94
existencia 2-gramov písmen v textových reťazcoch	9408	6	9408	2
frekvencia 2-gramov písmen v textových reťazcoch	9408	86	9408	65
informácie o textových reťazcoch	42	35	42	37
existencia 1-gramov hexadecimálnej reprezentácie vzorky	256	48	256	60
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	256	256	256	256

normalizovaná frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	256	256	256	256
existencia 2-gramov hexadecimálnej reprezentácie vzorky	65536	64464	65536	63184
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	65536	65536	65536	65536
normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	65536	65536	65536	65536
histogram entropie bajtov	256	256	256	256
pomery veľkostí extrahovaných súborov	9	7	9	7
existencia 1-gramov bajtov operačných kódov	256	117	256	59
frekvencia 1-gramov bajtov operačných kódov	256	256	256	256
normalizovaná frekvencia 1-gramov bajtov operačných kódov	256	256	256	256
existencia 2-gramov bajtov operačných kódov	65536	65308	65536	65254
frekvencia 2-gramov bajtov operačných kódov	65536	65536	65536	65536
normalizovaná frekvencia 2-gramov bajtov operačných kódov	65536	65536	65536	65536
existencia 1-gramov inštrukcií	1457	930	1395	887
frekvencia 1-gramov inštrukcií	1457	958	1395	933
existencia 2-gramov inštrukcií	214346	41847	195805	27664
frekvencia 2-gramov inštrukcií	214346	55601	195805	47242
existencia 1-gramov registrov	133	104	133	69
frekvencia 1-gramov registrov	133	124	133	104
existencia 2-gramov registrov	8451	2580	7737	2347

frekvencia 2-gramov registrov	8451	4162	7737	2966
informácie o inštrukciách a registroch	7	6	7	6
informácie o disasemblovanom súbore	119	105	120	109

Rovnako ako v prvom datasete nám po selekcii DF ostal z exportov len atribút ich počtu. Pri metadátach selekcia varianciou neodstránila v oboch datasetoch žiadne atribúty, čo znamená, že na rozdiel od prvého datasetu sa medzi vzorkami atribút MachineType líšil. To mohlo byť spôsobené zabalenými vzorkami.

V treťom datasete sa po prvotnej selekcii vylúčilo rádovo viac n-gramov slov ako v prvých dvoch. To sa ale dalo čakať vzhľadom na to, že v tomto datasete sú obfuskované, šifrované a zabalené vzorky malvéru. Z tohto dôvodu je rozdelenie slov viac náhodné a je menšia šanca, že sa budú opakovať vo viacerých vzorkách. Tiež sa zníži počet slov, ktoré dávajú zmysel a nie sú vytvorené obfuskáciou, šifrovaním alebo zabalením.

V treťom datasete sme pre niektoré disasemblované súbory nevedeli dekódovať bajt „*0x9d*“. Python interpretér na našom stroji používa kódovanie (encoding) cp1252 (Windows-1252). Použité kódovanie sa dá v Pythone zistiť cez príkaz `print(locale.getpreferredencoding()`. Skúsili sme použiť aj kódovanie UTF-8, ale tiež sa daný bajt nedekódoval. Kódovanie Latin-1 bolo schopné dekódovať tento bajt, ale to len z dôvodu, že toto kódovanie je schopné dekódovať všetky bajty. Nakoniec sme sa rozhodli ponechať cp1252 kódovanie a nahradiť výskyty tohto bajtu znakom otáznika. V ostatných datasetoch sa tento problém nevyskytol.

V druhom datasete po prvotnej selekcii ostalo 536 109 atribútov, v druhej skupine (skupiny atribútov do tisíc členov) je 4781 atribútov a v tretej (skupiny do sto atribútov) ostalo 518 atribútov. Tretí dataset má po prvotnej selekcii 478 831 atribútov, z toho je 6070 v druhej skupine a 622 v tretej skupine. Tretí dataset sa odlišoval hlavne počtom n-gramov z textových reťazcov (printable strings), ktorých mal po selekcii výrazne menej, čo bolo pravdepodobne spôsobené tým, že ho tvoria obfuskované, šifrované a zabalené vzorky.

3.3 Selekcia atribútov

3.3.1 Prvotná selekcia a predspracovanie

Ako sme písali v predošlej podkapitole, prvotnú selekciu sme robili počas extrakcie. Skladala sa z frekvencie dokumentov (document frequency – DF) a selekcie nízkou varianciou (low variance selection). Pri niektorých skupinách atribútov ktoré sa musia vyskytovať v každej vzorke sme DF nerobili (entropia, sekcie, metadáta...). Pri DF sme stanovili hranicu na 10% veľkosti najmenšej triedy. Atribút sa vo vzorke vyskytuje, ak jeho hodnota pre ňu nie je nulová. Naše atribúty sú počty alebo číslom vyjadrené vlastnosti (veľkosť, entropia, existencia prvku...). Nulová hodnota teda znamená neprítomnosť v datasete. V prvom datasete má najmenšia trieda len 86 prvkov, čo by znamenalo 9 vzoriek. Na začiatku sme zvolili hranicu 9, ale potom, keď sme sa rozhodli použiť stratifikovanú 10-fold kros-validáciu, sme to zmenili. Stratifikácia zachováva pomer tried. To znamená, že môže nastať situácia, že v jednom folde by bola z každej triedy presne desatina vzoriek, v ktorej je nejaký atribút všade nulový. Vyhnúť tomu sa dá posunutím prahu o 1, čo je len malá zmena. Z tohto dôvodu sme rozhodli, že aspoň 10 prvkov musí mať nenulovú hodnotu, aby atribút prešiel cez DF. V druhom datasete mala najmenšia trieda 181 vzoriek. Za prah sme nastavili číslo 19. V treťom datasete mala najmenšia trieda 92 vzoriek pri stanovenom prahu 10.

Keďže v niektorých skupinách bolo príliš veľa atribútov, tak sme sa rozhodli zmenšiť ich počet už v prvom kroku. Z tohto dôvodu sme pre skupiny, v ktorých je nad 100 000 atribútov, za hranicu určili 20 vzoriek pre prvý dataset a 25 pre druhý dataset. Pre tretí dataset sme neurčili hornú hranicu, keďže obsahoval najmenej atribútov. Okrem toho sme vylúčili v týchto skupinách aj atribúty, ktoré v takmer celom datasete nemali nulovú hodnotu (aspoň 950 vzoriek v prvom datasete a 1540 v druhom datasete). V treťom datasete sme to nevykonali, keďže mal dosť málo atribútov aj bez tejto úpravy. Týmto krokom sme okrem tých atribútov, ktoré sa takmer v žiadnej vzorke nevyskytujú, vylúčili aj tie, ktoré opačne takmer nikde nechýbajú. Tak veľké skupiny majú len n-gramy a náš predpoklad bol, že n-gram, ktorý sa vyskytuje v takmer všetkých vzorkách, bude menej prispievať ku klasifikácii ako ten, ktorý sa v niektorých vzorkách nevyskytuje.

Po DF nasleduje selekcia pomocou nízkej variancie, stále počas extrakcie. Prahy variancie, ako sme už spomínali sú:

- 0.01 – predvolená hranica,

- pre skupiny nad 1000 atribútov to je 0.05,
- pre skupiny nad 10 000 atribútov je prah 0.1.

Hlavný cieľ prvotnej selekcie je redukovať počet atribútov. Z tohto dôvodu sme sa sústredíme na hlavne veľké skupiny. Jadro selekcie prebehne neskôr, pričom po každej metóde už aj skontrolujeme presnosť pri klasifikácii. Takto získame viac informácii o vybraných atribútoch.

Skôr, ako sme začali s hlavnou selekciou, sme robili ešte úpravy na dátach. Pôvodne sme v hlavičke atribútov mali jednoznačný názov každého atribútu, skladajúci sa z predpony – mena skupiny a jeho mena. Pomocou týchto názvov sme chceli zistiť nielen to, ktoré skupiny atribútov si selekcie vyberajú, ale aj jednotlivé atribúty v nich. Neskôr sa ukázalo, že niektoré mená atribútov obsahovali úvodzovky aj čiarky, takže sa pri čítaní zle rozdelili a veľkosť hlavičky sa nezhodovala s počtom atribútov. Z tohto dôvodu sme zmenili názov atribútov na meno ich skupiny, ktorú vieme skontrolovať.

Pre načítanie a prácu s atribútmi sa v Pythone používajú dve knižnice: NumPy [168] a Pandas [169]. NumPy polia sú rýchlejšie. Pandas dataframe ponúka na druhej strane viac funkcionality (napríklad time series). Naše atribúty mali väčšinou typ integer, ale približne 30 atribútov bolo float. Pôvodne sme sa rozhodli využiť príkaz `np.genfromtxt(dtype=None)`, ktorý vyberie typ pre každý stĺpec osobitne [159]. Rovnako sa správa `pandas.read_csv(dtype=None)` [160].

Ak ale nie sú v celom poli rovnaké atribúty, tak NumPy knižnica vytvorí jednorozmerné pole zoznamov jednotlivých riadkov, ktoré sme nemohli použiť. Pandas knižnica tento problém nemala. Ale dve knižnice na selekciu atribútov (pyHSICLasso a skfeature) nepodporujú Pandas DataFrame (pyHSICLasso podporuje, ale len priamo z csv súboru v tvare, ktorý náš dataset nemal). Z tohto dôvodu sme sa rozhodli diskretizovať float atribúty, aby sme mohli využiť priamo NumPy pole.

NumPy pole má výhodu v rýchlosti pri jedinom dátovom type. Ukladá dáta do celistvého bloku v pamäti a pozícia prvku sa počíta pomocou smerníka na začiatok poľa a veľkosti dátového typu ako v jazyku C. Naproti tomu Python pole je zoznam smerníkov na jednotlivé objekty, ktoré môžu byť uložené na rôznych miestach v pamäti. Z tohto dôvodu sú omnoho pomalšie. Okrem rýchlosti má NumPy pole výhodu aj vo veľkosti. Typy np.float64 a np.int64 majú 64 bitov [161], kým float v Pythone má minimálne 24 bajtov, rovnako ako integer, lebo sú to plné objekty s vlastnými metódami. Podľa potreby si vedia alokovať viac pamäte a zväčšiť sa. Veľkosť Python typov záleží aj od operačného

systému. Na konkrétnom počítači sa dá zistiť cez príkazy `print(sys.getsizeof(float()))` a `print(sys.getsizeof(int()).`

Diskretizujeme tak, že pre každý atribút zistíme, koľko mali jeho hodnoty maximálne číslic s desatinnou čiarkou. Potom oddelíme časť za desatinnou čiarkou na maximálne 5 cifier. Ak mal atribút menej ako 5 cifier za desatinnou čiarkou, tak jeho hodnoty vynásobíme 10^{max}, ináč 10^5 a zmeníme ich na np.uint64. Bežne sa robí diskretizácia tak, že podobné čísla sa zlučujú do skupín a potom sa zo skupiny vyberie reprezentant (napríklad zaokrúhlený priemer) a všetky čísla v skupine sa prepíšu na reprezentanta. My sme sa rozhodli nejsť touto cestou, keďže zlučovaním hodnôt by sme stratili časť informácií v atribútoch.

Po diskretizácii robíme aj štandardizáciu. Musíme ju robiť z dôvodu, že pre niektoré funkcie, ako je napríklad RBF pre SVM alebo L1 a L2 regularizácia pri lineárnych modeloch (teda aj lineárne SVM) nie je dobré, ak rôzne atribúty majú príliš veľký rozdiel vo variancii. Rozdiel vo variancii je spôsobený tým, že jeden atribút má veľmi veľké rozpätie hodnôt a druhý veľmi malé. V takom prípade atribút s veľkou varianciou pri učení bude dominovať a tie s malou budú mať minimálny vplyv. Navyše tieto funkcie očakávajú vstup centrovaný na nulu [162]. Z tohto dôvodu, pred tým, ako urobíme selekciu alebo klasifikáciu na SVM alebo HSIC lasso, tak použijeme standard scaler [162]. Ten zmení atribúty tak, aby boli centrované okolo nuly a mali rovnakú varianciu.

Po štandardizácii sme spustili SVM na štandardizovaných aj normalizovaných (<-1,1>) atribútoch. Normalizácia zlepšila výsledky pre RBF, polynomiálne a sigmoidálne SVC (o 11%, 29% a 4%) a zároveň zlepšila časy. Štandardizácia mala rovnaké výsledky, ale ešte viac zlepšila časy. Pri lineárnom SVM sa zlepšovali len časy, aj to nepatrne. Pri lineárnom SVM z liblinear knižnice sklearn sa pri štandardizovaných dátach nepodarilo konvergovať k riešeniu. Pri SGD lineárnom SVM nám ostal výsledok aj čas rovnaký. V dokumentácii sa ale odporúča štandardizácia pred jeho použitím (pre možné zrýchlenie výpočtu).

3.3.2 Výber metód selekcie

Pred tým, ako sme začali robiť samotnú selekciu, sme sa rozhodli vylúčiť niektoré metódy, pretože ich počet časovo komplikuje proces selekcie. Najprv sme potrebovali z rýchlych metód vybrať tie, ktoré zvládnu spracovať všetky atribúty. Všetky rýchle metódy sme spustili na 92 000 a 220 000 atribútoch. Použili sme stroj s 32 GB RAM a 6

jadrovým procesorom AMD 5 3600. Najhorší čas bol 25 minút na 220 000 atribútoch a mala ho metóda SPEC. Na týchto atribútoch bola najrýchlejšia ANOVA, ktorá nebežala ani dve sekundy. Trace ratio metóda neukončila selekciu, dosiahla pridelené pamäťové prostriedky už pri 92 000 atribútoch, čo je pochopiteľné, lebo využíva dve matice atribútov naraz.

Následne sme vyskúšali aj embedded metódy: zisk (gain) a rozdelenie (split) pre XGBoost a LGBM (bližšie sme ich popísali pri gradient boosting). Pre XGBoost sa rozdelenie označuje ako váha (weight). Nepoužili sme totálny zisk (suma všetkých ziskov), ale priemerný zisk na rozdelenie. Rozhodli sme sa tak z dôvodu, že prvé rozdelenia majú zvyčajne vyššie hodnoty zisku ako tie, ktoré boli použité nižšie v strome a aj sú dôležitejšie – vedia najlepšie rozdeliť veľkú časť datasetu. Ak by sme chceli vybrať malé množstvo atribútov pre najlepšiu klasifikáciu, tak to sú tieto atribúty. Ale v prípade, že by nejaké atribúty boli často použité nízko v strome, mohli by akumulovať dosť veľký zisk a predbehnúť atribúty na vrchole. Tomuto zamedzíme, keď berieme do úvahy priemerný zisk. Atribúty, ktoré boli použité len na vrchole, ho budú mať najvyšší. Tie atribúty, ktoré boli (aj keď často) použité len nad listami, ho budú mať najnižší. LGBM má len možnosť pre totálny zisk. Takto aspoň budeme môcť porovnať rozdiely pre metriky zisku a totálneho zisku.

Ďalšie metódy sú CatBoost, RandomForestClassifier (RFC) s RegularizedForest (RGF). RFC používa priemerný pokles nečistoty (mean decrease in impurity - MDI) [163], teda zisk. CatBoost používa vlastnú metriku, ktorá počíta, ako sa v priemere mení hodnota výsledku klasifikácie, keď sa mení hodnota atribútu. Veľká zmena znamená veľké skóre [164]. Potom sme použili *L1* a *L2* regularizáciu pre lineárne SVC (SVM pre klasifikáciu – support vector classifier), SGD SVM (SVM, ktoré využíva stochastic gradient descent pri učení) a *L2* regularizáciu pre lineárnu verziu kernelizovaného SVM. Pre SGD SVM sme použili aj elasticnet. Z dôvodu malej rýchlosti pre celý dataset vypadli: L2 regularizácia pre SVC (40 minút pre 92 000 atribútov), CatBoost (43 minút pre 92 000). RGF pri teste dosiahol pridelené pamäťové prostriedky. Blokové HSIC lasso ukončilo test úspešne, ale je to len aproximácia, a čím viac atribútov sme použili, tým väčšiu aproximáciu sme museli povoliť, aby test ukončilo. Pre všetky atribúty (483613) dosiahlo pridelené pamäťové prostriedky aj pri najväčšej miere aproximácie. V prílohe D máme uvedené dosiahnuté časy pri výbere 1000 atribútov z celého prvého datasetu

(486136 atribútov) pre rýchle metódy a časy pri výbere 1000 atribútov z 5536 pre všetky metódy (okrem najpomalšej metódy CFS).

Na prvom datasete sme porovnali metódy na presnosť. Z 5567 atribútov v druhej skupine každý algoritmus, ktorý sme chceli použiť na celý dataset vybral 10% (556), na nich sme robili klasifikácie a porovnali presnosť klasifikácií na stromových algoritmoch (XGBoost, LGBM, LGBM GOSS, RGF) a potom na SVM (linear, RBF, sigmoid). Z každej skupiny algoritmov sme brali najlepší a najhorší výsledok pre stromové a SVM klasifikátory. Výsledky sú v nasledujúcej tabuľke (Tab. 6). V tejto tabuľke „stromy min" znamená najnižší výsledok pre stromové klasifikátory a „stromy max" zase najlepší výsledok pre tieto klasifikátory. Obdobne to platí aj pre SVM klasifikátory. Všetky metódy majú presnosť v aspoň jednom výsledku nad 95%. Túto hodnotu sme určili za prah – metódy ktoré ju nedosiahnu, nepoužijeme. Keďže ju dosiahli všetky metódy, tento spôsob výberu metód nevylúčil žiadnu. Jediné metódy, ktoré sme vylúčili pre celý dataset, sú tie, ktoré boli pomalé alebo pamäťovo náročné.

Tab. 6 Výsledky selekcií na 556 atribútoch.

	Stromy min	Stromy max	SVM min	SVM max
ANOVA	97.1%	98%	76.7%	93.3%
Chi-square	96.4%	97.4%	58.2%	92.4%
Fisher	97.2%	98.4%	76.7%	93.3%
Gini	96.2%	99%	84%	97.3%
Laplacian	89.1%	96.3%	82.2%	93.5%
Mutual info	95.3%	96.2%	60.3%	93.9%
ReliefF	95.7%	96.8%	51.3%	93%
SPEC	95.1%	95.9%	51.9%	92.2%
LGBM zisk	98.4%	98.9%	95.4%	98.5%
LGBM rozdelenie	98.2%	98.6%	95%	98%
RFC	98%	98.8%	94.1%	97.7%
SGD L1	95.6%	98.5%	95.3%	98.3%
SVC L1	98.1%	98.6%	97.5%	99.28%

XGBoost zisk	98.3%	98.9%	95.4%	98.2%
XGBoost rozdelenie	98.3%	98.9%	95.4%	98.2%
pôvodne	98.1%	98.7%	95.7%	98.1%

Na tretej skupine atribútov (766) sme naopak vyskúšali pomalšie metódy. Vybrali 76 vzoriek (10% z počtu atribútov v tretej skupine) a na rovnakých klasifikáciách sme porovnali výsledky. Pre porovnanie sme pridali aj väčšinu rýchlych metód. Za prah sme určili 90%. Výsledky sú uvedené v tabuľke (Tab. 7). CFS sme vylúčili, keďže bol veľmi pomalý (už 20 atribútov sa vyberalo 42 minút) a má najhoršiu zložitosť – kvadratickú. Z tabuľky vidno, že pomalé metódy majú podobné výsledky ako rýchle a že embedded metódy majú najlepšie výsledky.

Tab. 7 Výsledky selekcií na 76 atribútoch.

	Stromy min	Stromy max	SVM min	SVM max
CatBoost	96.7%	97.6%	86.8%	93.5%
CIFE	88.4%	92.6%	69.3%	83.6%
CMIM	92.4%	95.1%	73.1%	84%
DISR	95.9%	97%	77.4%	92.3%
FCBF (vybral 17)	90.7%	95.7%	78.6%	84.7%
HSIC Lasso (B=17)	96.6%	97.9%	92.1%	95.4%
JMI	95.3%	96%	63%	89.6%
LGBM zisk	97.8%	98.4%	86.9%	95.3%
LGBM rozdelenie	97.4%	98.2%	85.9%	93.2%
MIFS	76.6%	84.8%	48.4%	56.2%
MRMR	95.9%	97.8%	82.2%	93.2%
RFC	97.2%	97.6%	84.5%	95.2%
RGF	97.7%	98.4%	88.9%	95.2%
XGBoost zisk	97.7%	98%	87.3%	94.7%
XGBoost rozdelenie	97.6%	98.2%	87.4%	94.6%

SVC l1	95.7%	97.3%	90.9%	95.2%
SVC l2	95.8%	97.2%	90.5%	94.4%
ANOVA	95.7%	97.7%	72.7%	89.2%
Chi-square	95.6%	96.7%	71%	89%
Fisher	95.8%	97.5%	72.7%	89.2%
Gini	93.3%	97.6%	74.3%	88.2%
Laplacian	76.9%	92.6%	48.5%	65.9%
Mutual info	94.6%	96.3%	64%	89.6%
ReliefF	95.9%	96.6%	72.8%	88.9%
SPEC	92.4%	94.5%	58.2%	84.1%
pôvodne	97.7%	98.2%	88.7%	95%

Jedine MIFS metóda klesla pod 90% pri stromoch a zároveň mala najhoršie skóre pri SVM. To je zaujímavé zistenie, keďže má rovnaký model ako metóda MRMR, ktorá dosahuje omnoho lepšie výsledky. MRMR metóda vykazuje proti MIFS len jeden rozdiel - dynamickú penaltu za redundanciu, ktorá sa časom znižuje. To môže znamenať, že MIFS metóda mala z dôvodu svojej prísnosti nízke skóre pri tak malom počte atribútov. Pri väčšom množstve má potenciál dosiahnuť omnoho lepšie výsledky. Z tohto dôvodu ju necháme a v celkových výsledkoch zistíme, či sa skóre zlepšilo pri viacerých atribútoch.

Niekoľko metód v knižnici scikit-feature [170] sme pozmenili. CFS pôvodne ukončilo selekciu, keď sa päťkrát po sebe nezlepšilo skóre. My sme ukončenie podmienili počtom selektovaných atribútov, ktorý sa zadáva ako nový parameter. Pri výpočte entropie sa používa aproximačná verzia. Z tohto dôvodu sa nám stávalo, že atribút ju mal nulovú, aj keď predtým bola vykonaná selekcia varianciou. Touto entropiou sa delí skóre a nulou nemožno deliť. Atribúty s nulovou aproximovanou entropiou sme nebrali do úvahy (odchytili výnimku a nepridali ich). Pri Gini skóre sme zmenili inicializované skóre z 0.5 – maximum pre dve triedy na 1 – globálne maximum. Pre Laplacian skóre sme za predvolenú určili verziu s učiteľom.

Samotnú selekciu robíme nasledovne: zo všetkých atribútov vyberáme 1000 a 500 najlepších atribútov s metódami, ktoré zvládnu spracovať celý dataset. 1000 a 500

atribútov potom selektujeme aj z druhej a tretej skupiny, pokiaľ sú dosť veľké. Okrem toho na všetkých atribútoch spúšťame FCBF, ktoré by malo nájsť najmenšie množstvo atribútov, ktoré je potrebné pre klasifikáciu. Pre prvý dataset nám vyšlo 79 atribútov, pre druhý dataset 48 atribútov a pre tretí dataset 62 atribútov. Tento počet atribútov potom selektujeme na všetkých troch skupinách.

Pri druhom a treťom datasete sme robili len rýchle selekcie. Najprv sme zistili na troch skupinách atribútov, ktoré označenia (labels) sú pri klasifikácii lepšie (zhlukovanie alebo konsenzus). Na týchto označeniach sme urobili selekciu a následnú klasifikáciu. Pomalé metódy (tie, ktoré časovo alebo pamäťovo nezvládli celý dataset) sme vynechali, keďže pre veľké datasety sú nepoužiteľné. Z toho dôvodu je nepravdepodobné, že by sa niekde pri veľkých datasetoch využívali.

3.4 Klasifikácia a výsledky

Pred klasifikáciou sme robili optimalizáciu hyperparametrov na prvom datasete, konkrétne jeho druhej skupine. Pri stromoch sa za optimálnu maximálnu hĺbku určili 7, minimálny počet vzoriek v liste je 10, počet listov max. 80 a pre boosting sme zvolili 100 kôl a learning rate je 0.2. Pre CatBoost nemôžme zadávať obmedzenia, keďže pracujeme na CPU. Tieto metódy vyžadujú Nvidia grafické karty a my sme mali k dispozícii AMD grafickú kartu (RX 580). Dôsledok bol dlhý čas behu. RGF má predvolene 50 stromov, minimum na jeden list sme dali tiež 10 a počet listov je 1000. Mal by byť 4000, ak chceme 80 na jeden strom, ale potom je príliš pomalý a zlepšenie vo výsledkoch je minimálne. Pre SVM metódy sme väčšinou nechali predvolené hodnoty parametrov, ale zmenili sme počet cyklov na tisíc. Regularizáciu sme nezvyšovali, o to sa má postarať selekcia.

Klasifikácia prebieha nasledovne: ešte pred selekciou klasifikujeme všetky tri skupiny, aby sme mohli porovnať skóre pred a po selekcii. Keď sa vykonajú všetky selekcie, tak pre výsledok z každej metódy selekcie urobíme klasifikáciu a zaznamenáme výsledok. Pre SVM metódy robíme najprv štandardizáciu, ak selekcia nepoužívala SVM (ináč sa vykonala štandardizácia pred selekciou).

Po selekcii a klasifikácii spracúvame výsledky. Ukladáme si priemerné skóre po krosvalidácii, pre boosting algoritmy aj poradie iterácie učenia s najlepším výsledkom, aby sme vedeli posúdiť, kedy došlo ku overfittingu. Tento údaj sme využívali pri optimalizácií hyperparametrov. Pri spracovaní výsledkov pre každú selekciu zisťujeme,

aké množstvo atribútov pochádza z jednotlivých tried a koľko z nich patrilo do druhej a tretej skupiny.

3.4.1 Výber metód klasifikácie

Pri klasifikácii je výber metód dôležitejší, keďže výsledok každej metódy selekcie je použitý na všetkých metódach klasifikácie. Jediná metóda, ktorú sme vylúčili kvôli času, bol CatBoost. Už pri tretej skupine atribútov bežal 21 minút. Potom sme vylučovali metódy podľa dosiahnutých výsledkov pri klasifikácii. Pri druhej skupine (5567 atribútov) sme za prah určili 98% a pri tretej skupine (766 atribútov) 95%.

Pri druhej skupine atribútov pod prah 98% klesli: lineárny XGBoost, LGBM forest (les boostovaných stromov), RandomForestClassifier (RFC), linearSVC (knižnica liblinear), polynomiálne SVC a sigmoidálne SVC. Pri prvej skupine nedokázali dať 95% presnosť nasledovné metódy: lineárny XGBoost, RandomForestClassifier, linearSVC (liblinear), RBF SVC, polynomiálne SVC a sigmoidálne SVC. Lineárne SVC (knižnica libsvm) malo najlepší výsledok (99.28%).

Okrem LGBM lesa sú najslabšie algoritmy konzistentné (v oboch datasetoch boli pod hranicou), preto ich vynecháme. Okrem toho vynecháme aj XGBoost s použitím histogramu, ktorý teoreticky mal mať rýchlejšie trénovanie kvôli tvorbe histogramu atribútov, ale u nás ho mal minimálne dvojnásobne dlhšie. Tvorba histogramu bola približne rovnako časovo náročná ako samotné trénovanie. Tiež vynecháme XGBoost les, kde sa paralelne boostuje viac stromov, lebo v oboch prípadoch mal les rovnaký najlepší výsledok ako jeden strom. V závislosti od počtu stromov stúpol len čas tréningu. Vynechali sme aj LGBM RF. Je to len RF bez boostovania a nedosahuje výsledky čistého LGBM.

Ostáva nám XGBoost, LGBM, LGBM GOSS, RGF, SGD SVC a linearSVC (libSVM). RBF a sigmoid kernely necháme pre prípad, že by sa vyskytla množina atribútov, ktorú by vedeli rozdeliť lepšie ako lineárny SVM. Polynomiálny kernel mal z kernelov najhoršie výsledky, preto sme tento kernel vynechali. Keďže kernely sú výpočtovo komplexnejšie ako lineárne SVM, nepoužijeme ich na celý dataset.

Keď sme robili klasifikáciu na celom datasete, tak zo stromov ju zvládlo len RFC. Z toho dôvodu sme RFC nevylúčili, keďže v opačnom prípade by sme nemali žiaden stromový klasifikátor na celý dataset. LGBM, LGBM GOSS aj LGBM RF skončili

s chybou, keďže vyčerpali pamäťové prostriedky. XGBoost aj XGBoost RF neukončili učenie ani po niekoľkých hodinách. Počas behu mali využitú celú operačnú pamäť, takže muselo dochádzať k neustálemu stránkovaniu (ukladaniu častí pamäte z a na disk). Keďže kernely na celý dataset nepoužívame, tak nám ostali zo SVM metód len linearSVC (libSVM) a SGD. Len dva algoritmy sú málo vzhľadom na to, že SVM malo doteraz najlepší výsledok. Z tohto dôvodu sme nakoniec nechali aj linearSVC (liblinear) ale s L1 regularizáciou, keďže pôvodný s L2 regularizáciou bol veľmi pomalý (pri rovnakých parametroch trval beh 7 minút pre L1 a 1:46 pre L2 regularizované SVM).

4 Analýza výsledkov

V rámci tejto kapitoly porovnávame výsledky rôznych skupín algoritmov selekcie a overujeme naše predpoklady, stanovené v predchádzajúcich kapitolách Pre každý dataset porovnáme výsledky klasifikácii pred selekciou na všetkých atribútoch, aj na druhej a tretej skupine atribútov (jednoduché a veľmi jednoduché skupiny atribútov). Okrem toho porovnávame cez klasifikácie vzoriek malvéru označovanie (labeling) vo forme zhlukovania a konsenzu. Našou výskumnou otázkou je, či výsledky druhej a tretej skupiny boli blízke (menej ako percento nadol) výsledkom zo všetkých atribútov. Porovnanie nie je úplne presné, keďže nie všetky algoritmy zvládli klasifikovať vzorky malvéru na všetkých atribútoch (výpočtové alebo pamäťové obmedzenia). Následne porovnávame výsledky jednotlivých skupín aj po selekcii. V tomto prípade sú už výsledky presnejšie, keďže viac algoritmov zvládlo selektovať zo všetkých atribútoch. V prvom datasete používame aj pomalé metódy selekcie pre druhú a tretiu skupinu atribútov. V jeho zhlukovanej verzii a ostatných datasetoch ich už z časových dôvodov nepoužívame, a teda všetky tri skupiny majú rovnaké podmienky.

Tiež nás zaujíma, či počet atribútov, ktorý vybral FCBF algoritmus, bol postačujúci. Inými slovami, či najlepšie výsledky selekcií s týmto počtom atribútov mali maximálne o jedno percento nižší výsledok ako najlepší výsledok v kategórii. Pri FCBF algoritme sme ako prah delta určili nulu, teda žiadne obmedzenie, keďže aj bez ďalšieho obmedzenia nám ostávali po selekcii rádovo desiatky atribútov. Sledujeme aj to, aký počet atribútov bol minimálne potrebný pre najlepšie výsledky v jednotlivých kategóriách. V každej kategórii sledujeme, ktoré metódy selekcie vylepšili pôvodné skóre alebo ho aspoň zachovali s menším množstvom atribútov.

Aby sme kvôli jednému úspešnému výsledku neoznačili selekciu za úspešnú, rozhodli sme sa sústrediť na konzistentnosť výsledkov pre rôzne počty atribútov a rôzne metódy klasifikácie (stromové a SVM). Dôraz dávame na metódy, ktoré mali konzistentne najlepšie výsledky v jednotlivých kategóriách. Pri najlepších metódach selekcie si zaznamenávame aj to, do ktorej skupiny algoritmov patrili (rýchle, pomalé, embedded) a ktorá skupina mala najväčší počet reprezentantov pri najlepších metódach. Špeciálne nás zaujíma, aké výsledky budú embedded metódy dosahovať pri klasifikácii iným typom klasifikátora.

V každej kategórii zaznamenávame aj skupiny atribútov, ktoré vybrali najlepšie selekcie pre obe skupiny klasifikačných metód (stromové a SVM). Pre obe skupiny

klasifikácií potom zisťujeme, ktoré skupiny atribútov sa vyskytovali v najúspešnejších selekciách. Okrem toho sledujeme aj atribúty, vybrané najlepšími selekciami pre veľkosť selekcie určenú metódou FCBF. Tieto atribúty by mali mať najväčšiu rozdeľovaciu silu.

Po analýze všetkých výsledkov na každom datasete sa pozeráme na globálne trendy cez všetky tri datasety. V každom datasete robíme označovanie (labeling) cez zhlukovanie aj cez konsenzus. Zisťujeme, ktorý spôsob označovania má lepšie výsledky klasifikácie. Na tomto mieste treba brať do úvahy dva fakty. Prvým je to, že sme nepoužili úplný konsenzus, keďže sme požadovali len to, aby sa daná trieda vyskytovala vo viacerých AV v ľubovoľnej vzorke, nie v každej. Do datasetu sa tak dostali aj vzorky, ktorých trieda nebola v požadovanom počte AV rovnaká. Robili sme to kvôli tomu, aby sme mali viac tried, keďže niektoré triedy boli ináč príliš malé. Druhý fakt je to, že označovanie cez zhlukovanie robíme až na datasetoch, na ktorých boli vylúčené vzorky, ktoré neprešli konsenzom, teda sa nevytvorí generická trieda. Pri zhlukovaní boli vylúčené aj niektoré vzorky ako outliere.

Súčasne nás zaujímajú aj prípadné rozdiely medzi najlepšími metódami selekcie medzi datasetmi, hlavne pre najmenší počet atribútov. Dôležitejšie je však to, či sa zmenili najčastejšie vyberané skupiny atribútov v týchto selekciách. V prvom datasete sme sa vyhýbali zabaleným, šifrovaným a obfuskovaným vzorkám. V treťom sme vyberali práve šifrované, zabalené a obfuskované vzorky a v druhom je kombinácia oboch typov vzoriek. Zaujíma nás, ktoré skupiny atribútov dokážu odhaliť triedu aj napriek obfuskácii, šifrovaniu alebo zabaleniu a či statické atribúty dokážu mať dobré výsledky aj pri týchto obmedzeniach.

4.1 Prvý dataset

Tento prvý dataset by nemal obsahovať šifrované, obfuskované a zabalené vzorky. Pri konsenze sme mali oproti zhlukovaniu lepšie skóre len pre stromové algoritmy pre všetky atribúty (96.29% proti 94.95%). Žiaľ, v tomto prípade stromové algoritmy sú reprezentované len algoritmom RFC, ktorý náhodne vyberá atribúty pre stromy. Vo všetkých ostatných prípadoch (25 klasifikácií) malo lepšie skóre zhlukovanie. Zhlukovanie však vykazovalo určitú výhodu – odstránili sa outliere. Preto sme odstránili outliere aj z označení vzoriek malvéru pre konsenzus a znovu sme ich porovnali. Výsledky pre konsenzus sa zlepšili. To znamená, že vzorky, ktoré boli zhlukovaním

označené za outliere, pravdepodobne boli nesprávne označené antivírusovými programami, ako sme pôvodne predpokladali. Aj napriek zlepšeniu výsledkov konsenzu pomer lepších výsledkov ostal rovnaký. Okrem jedného popísaného prípadu malo zhlukovanie lepšie výsledky aj po odstránení outlierov z datasetu, označeného konsenzom.

Pred selekciou mala druhá skupina atribútov (zlúčenie skupín atribútov do 1000 členov) najvyššie výsledky pre oba typy klasifikácie – 98.96% pre stromy, 98.14% pre SVM pri použití označovania konsenzom a 99.45% pre stromy, 98.46% pre SVM pri zhlukovaní. Tretia skupina (skupiny atribútov do 100 členov) mala výsledky lepšie ako všetky atribúty len pre stromy. Pre SVM mala výsledok horší takmer o dve percentá. V oboch prípadoch to môže byť spôsobené tým, že niektoré metódy klasifikácie sme nemohli použiť na všetky atribúty.

Pri konsenze mala druhá skupina atribútov po selekciách veľmi podobné výsledky ako selekcie všetkých atribútov. Rozdiel medzi nimi nikdy neklesol pod tretinu percenta. Tretia skupina klesla pod percento vo všetkých prípadoch. Najhoršie to bolo v prípade SVM, pri ktorom klesla maximálne o 3.1%. Pri zhlukovaní sa rozdiely zmenšili. Pre stromy zaostávala druhá skupina len o stotinu percenta v dvoch prípadoch a v treťom bola o desatinu percenta lepšia. Pri SVM klesla o menej ako štvrtinu percenta. Tretia skupina na stromoch neklesla o viac ako o percento, ale na SVM aj o viac ako dve percentá. Celkovo má druhá skupina veľmi dobré výsledky pred aj po selekcii, ale tretia skupina už nestačí.

V tomto prvom datasete vybral FCBF algoritmus 79 atribútov. Vo všetkých prípadoch pre stromy neklesol ich výsledok pod najlepší výsledok viac ako o percento, ani keď sa vyberalo tisíc atribútov. Dokonca najhorší pokles pre stromy bol len o 0.22 percenta oproti selekcii tisíc atribútov. Pri SVM to bolo horšie. Pokles sa pohyboval okolo jedného percenta, v najhoršom prípade o 1.12%. Síce SVM o trochu prekročilo hranicu jedného percenta, ale 79 atribútov má veľmi dobré výsledky.

Pre stromy bolo pre všetky prípady až na jeden potrebných 500 atribútov na najlepšie skóre. V jednom prípade malo 79 atribútov ešte lepšie skóre ako 500. Pre SVM tiež v jednom prípade stačilo 79 atribútov. Pre vyššie počty sa už výsledok nezlepšil, ale pri konsenze a selekcii všetkých atribútov bolo pre najlepší výsledok potrebných tisíc atribútov. V tej istej situácii pri zhlukovaní stačilo len 79 atribútov, čiže to môže byť spôsobené množstvom vzoriek, ktoré boli na zlej strane nadroviny z dôvodu zlého

označenia. Dokonca pri zhlukovaní z druhej skupiny malo 500 atribútov lepší výsledok ako 1000. Ak berieme do úvahy len označovanie zhlukovaním, ktorý mal lepšie výsledky, tak nám pre stromy aj pre SVM stačí maximálne 500 atribútov. Použitie viacerých atribútov už skóre nezvýšilo. Pri ďalších výsledkoch berieme do úvahy už len označovanie prostredníctvom zhlukovania, keďže sa ukazuje byť v každom ohľade lepšie.

Ukázalo sa, že každá metóda selekcie aspoň raz vylepšila najlepšie skóre pre aspoň jednu skupinu metód klasifikácie. Žiadna ale nebola úplne konzistentná. Vždy sa vyskytol aspoň jeden prípad, keď daná metóda nevylepšila najlepšie skóre pred selekciou aspoň pri jednej metóde klasifikácie. Z tohto dôvodu sa pozrieme aspoň na tie, ktorým sa to podarilo najčastejšie.

Len pre stromové klasifikácie sú konzistentné obe XGBoost selekcie. Vo všetkých prípadoch (ôsmich) zlepšili pôvodný výsledok. Na druhom mieste je RFC a LGBM rozdelenie. Obe len raz nezlepšili pôvodný výsledok. Dvakrát nezlepšilo výsledok LGBM zisk, trikrát SVC L1 a SGD elasticnet. Zistili sme, že embedded stromové metódy sú najlepšie, po nich nasledujú embedded SVM metódy.

Pri SVM klasifikáciách nebol úplne konzistentný žiaden algoritmus. SVC L1 raz nevylepšilo výsledok. Na druhom mieste sú LGBM zisk, RFC a SGD L2. LGBM zisk, čo je totálny zisk, má lepšie výsledky ako XGBoost priemerný zisk. To môže znamenať, že naše predpoklady (priemerný zisk preferuje atribúty, ktoré sú použité pri rozdeleniach na vrchole stromu a preto budú dávať lepší výsledok), kvôli ktorým sme preferovali priemerný zisk, nie sú správne. Totálny zisk môže mať lepšie výsledky napríklad v situácii, keď je rozdiel medzi ziskom v horných a dolných leveloch stromu príliš veľký. Tým by sa rapídne znížil priemerný zisk atribútu, ktorý bol použitý na vrchole a často aj na najnižšom leveli. Vyššie skóre by mali atribúty, ktoré boli použité pod vrcholom, ale nikdy nie najnižšie. Ako sa dalo čakať, L1 regularizácia má najlepšie výsledky. Zaujímavé je, že embedded stromové metódy sú na úrovni ostatných embedded SVM metód. Napriek tomu, že embedded metód je spolu 10 a rýchlych metód je 8, embedded jasne prevládajú bez ohľadu na použitý klasifikátor. Pomalé metódy sme použili len pre označovanie konsenzom, ale okrem jedného prípadu, keď sa vyberalo 500 atribútov z 766 (teda väčšina), ani raz nezlepšili pôvodný výsledok. Výnimkou je FCBF algoritmus, ktorý z celého datasetu vybral len 79 atribútov a po ich použití bol výsledok lepší ako pôvodný.

Keď sme brali pre každý experiment len najlepší výsledok selekcie, potvrdila sa dominancia embedded metód aj pre iný typ klasifikátora. Najlepší výsledok môže súčasne nadobudnúť viac metód, takže počet najlepších výsledkov je väčší ako počet pozorovaní.

Pre stromy malo najviac najlepších výsledkov LGBM rozdelenie (šesť z ôsmich). Druhé bolo XGBoost rozdelenie, malo tri najlepšie výsledky. Tretí bol LGBM zisk s dvomi výsledkami. Je zaujímavé, že v oboch boosting algoritmoch rozdelenie získalo vyššie výsledky ako zisk. Je teda lepšie počítať, v koľkých prípadoch bol atribút použitý na rozdelenie, ako počítať zisk z rozdelení atribútu. Okrem stromových metód v dvoch prípadoch mali najlepšie výsledky aj SVM metódy, raz čisté SVM a raz SGD L1. Jeden výsledok mal aj XGBoost zisk. Globálne najlepší výsledok pre stromy bol 99.67% a dosiahlo ho dva krát LGBM rozdelenie.

V najlepších selekciách pre SVM klasifikácie jasne dominoval SVC L1 so štyrmi najlepšími výsledkami. Ďalšie tri SVM metódy: SGD L2, SVM, SGD elasticnet získali jeden najlepší výsledok. Jeden výsledok získalo aj XGBoost rozdelenie. Najlepší výsledok bol rovnaký ako pre stromové klasifikácie – 99.67% a dosiahli ho dve metódy – SVC L1 a SGD L2. Znovu sa ukázalo, že bez ohľadu na klasifikátor sú embedded metódy lepšie ako ostatné metódy selekcie.

Aj z druhej skupiny atribútov možno získať veľmi dobré výsledky. Pri stromoch možno získať výsledok horší len o stotinu percenta – 99.66% a to už len pri výbere 79 atribútov. Pri SVM zase možno dostať výsledok 99.55% pri 500 atribútoch z druhej skupiny.

Globálne najlepší výsledok pre stromové klasifikácie bol dosiahnutý dvakrát. V oboch prípadoch ho dosiahlo LGBM rozdelenie pre výber zo všetkých atribútov. Raz sa vybralo 500 atribútov, druhý raz 1000. Atribúty, ktoré sú v selekcii z 500, pokladáme za dôležitejšie, keďže aj pri menšom počte dokázali dať najlepší výsledok. Z toho dôvodu za najlepšiu selekciu berieme tú z 500 atribútov a popíšeme bližšie len atribúty tejto selekcie. V tejto selekcii vybral algoritmus z 500 atribútov 111 atribútov z druhej skupiny, z toho 30 bolo v tretej.

Pre SVM klasifikácie boli tiež dva výskyty globálne najlepšieho výsledku. V prvom prípade vybral SVC L1 500 najlepších atribútov zo všetkých. V druhom prípade SGD L2 vybralo 1000 atribútov zo všetkých. Znovu za najlepšiu selekciu považujeme tú z 500 atribútov. Pri tejto selekcii vybral SVC L1 len 61 atribútov z druhej skupiny, z toho len 25 z tretej.

Skupiny vybraných atribútov v najlepších výsledkov pre stromové aj SVM klasifikácie sa nachádzajú v tabuľke (Tab. 8). Pre každú skupinu je uvedené, koľko jej atribútov sa vyskytlo v selekcii, ktorá dosiahla najlepší výsledkov pre stromové a potom aj SVM klasifikátory. Pokiaľ sa skupina atribútov ocitla len v jednom z najlepších výsledkov, bude počet vybraných atribútov uvedený len v príslušnom stĺpci.

Tab. 8 Atribúty, použité pri najlepších klasifikáciách v prvom datasete.

Názov skupiny atribútov	Počet pre stromy	Počet pre SVM
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	111	24
frekvencia 2-gramov inštrukcií	93	41
existencia 2-gramov inštrukcií	-	44
frekvencia 2-gramov bajtov operačných kódov	66	11
existencia 2-gramov hexadecimálnej reprezentácie vzorky	38	178
existencia 2-gramov bajtov operačných kódov	31	42
histogram entropie bajtov	26	2
frekvencia 2-gramov registrov	22	1
existencia 2-gramov registrov	-	19
frekvencia 1-gramov bajtov operačných kódov	15	-
frekvencia 1-gramov inštrukcií	14	-
normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	13	17
existencia 1-gramov textových reťazcov	9	6
informácie o sekciách	9	-
frekvencia 1-gramov textových reťazcov	8	50
normalizovaná frekvencia 2-gramov bajtov operačných kódov	6	10
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	5	-
informácie o importovaných funkciách	5	15
informácie o importovaných knižniciach	4	5

informácie o disasemblovanom súbore	3	-
informácie o textových reťazcoch	3	-
existencia 1-gramov hexadecimálnej reprezentácie vzorky	2	7
frekvencia 1-gramov registrov	2	1
existencia 2-gramov textových reťazcov	2	-
informácie o inštrukciách a registroch	2	-
metadáta	2	-
informácie o overlay dátach	2	-
pomery veľkostí extrahovaných súborov	2	1
normalizovaná frekvencia 1-gramov bajtov operačných kódov	1	2
normalizovaná frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	1	4
frekvencia 2-gramov textových reťazcov	1	2
entropia spustiteľného súboru	1	1
informácie o zdrojoch	1	2
existencia 1-gramov inštrukcií	-	7
existencia 1-gramov bajtov operačných kódov	-	4
frekvencia 2-gramov písmen v textových reťazcoch	-	3

V najlepšej selekcii pre SVM sa oproti selekcii pre stromy zmenila hlavne jedna vec – pribudli n-gramy z textových reťazcov. Disasemblovaný súbor a hexadecimálna reprezentácia vzorky ale ostávajú medzi hlavnými zdrojmi.

Najmenšie množstvo atribútov, ktoré sme vyberali, bolo 79. Pre toto minimum atribútov sme všetky globálne najlepšie výsledky dosiahli pri výbere 79 atribútov z druhej skupiny. Pre stromy sme dosiahli 99.66% až pri troch metódach – obe LGBM a XGBoost rozdelenie. Pre SVM bol najlepší výsledok len 98.9% a dosiahlo ho SVC L1. SVM vo všeobecnosti majú horšie výsledky pre príliš malé množstvo atribútov.

LGBM zisk vybral len 13 atribútov z tretej skupiny, LGBM rozdelenie 29. XGBoost rozdelenie ich vybralo 16, SVC L1 zase 15. Vybrané atribúty sú popísané v tabuľke (Tab. 9).

Tab. 9 Najlepšie selekcie z 79 atribútov – prvý dataset.

Názov skupiny atribútov	LGBM zisk	LGBM rozdelenie	XGBoost	SVC L1
frekvencia 1-gramov inštrukcií	15	14	21	4
existencia 1-gramov textových reťazcov	11	4	17	17
informácie o importovaných funkciách	9	2	3	27
histogram entropie bajtov	8	19	9	1
frekvencia 1-gramov bajtov operačných kódov	5	-	2	-
normalizovaná frekvencia 1-gramov bajtov operačných kódov	5	-	-	-
informácie o disasemblovanom súbore	5	-	4	-
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	3	2	6	-
frekvencia 1-gramov registrov	3	-	2	-
existencia 2-gramov textových reťazcov	3	2	1	6
informácie o importovaných knižniciach	3	-	3	4
normalizovaná frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	2	1	-	6
metadáta	2	5	2	1
informácie o sekciách	2	9	3	1
existencia 1-gramov hexadecimálnej reprezentácie vzorky	1	1	-	4
informácie o inštrukciách a registroch	1	1	1	-
informácie o zdrojoch	1	2	-	2
informácie o textových reťazcoch	-	4	3	1

pomery veľkostí extrahovaných súborov	-	1	-	-
informácie o overlay dátach	-	-	2	-
existencia 1-gramov inštrukcií	-	-	-	3
frekvencia 2-gramov písmen v textových reťazcoch	-	-	-	1
informácie o exportoch	-	-	-	1

Z vybraných atribútov pre stromy vidíme, že aj keď nemáme takmer žiadne 2-gramy (lebo vyberáme len z druhej skupiny), 1-gramy sú stále postačujúce. Histogram entropie bajtov bol raz na prvom mieste, ale pri globálne najlepšom výsledku bolo vybraných ešte viac atribútov z tejto skupiny.

Pri atribútoch vybraných SVC L1 sa znovu ukazuje, že 1-gramy postačujú pre dobrý výsledok. Zaujímavé je prvenstvo importovaných funkcií. Je ich viac ako pri globálne najlepšom výsledku. Napriek tomu, že importované funkcie nie sú vo všeobecnosti často vyberané, tu vidíme, že sú dôležité. Ich výhoda je, že už pred prvotnou selekciou ich bolo málo (rádovo tisíce), a teda nezaberajú veľa miesta. Dajú sa ale ľahko obfuskovať.

4.2 Druhý dataset

Tento dataset pozostáva zo všetkých vzoriek (čistých, obfuskovaných, šifrovaných a zabalených). Vo všetkých prípadoch malo v tomto datasete označenie zhlukovaním lepšie výsledky ako označenie cez konsenzus. Z tohto dôvodu sme pri selekcii použili len dataset, označený zhlukovaním.

Pred selekciou mala najlepšie skóre na stromoch druhá skupina – 99.15%, ale na SVM bol najlepší výsledok pri všetkých atribútoch – 98.14%. Tretia skupina bola lepšia ako všetky atribúty na stromoch, ale na SVM mala najhorší výsledok – o 3% horší ako všetky atribúty. Po selekcii mala druhá skupina pre stromy horšie výsledky ako všetky atribúty, ale len minimálne – rozdiel bol vždy menší ako 0.5%. Pri SVM boli už rozdiely v neprospech druhej skupiny väčšie, až do 1.39%. Na rozdiel od prvého datasetu tu už aj druhá skupina klesla pod jedno percento. Tretia skupina má pri stromoch rozdiely

smerom nadol do jedného percenta, ale pri SVM to je oveľa horšie – najväčší rozdiel je viac ako 4%. Pokiaľ sa použijú stromy, majú dobré výsledky obe skupiny, ale pri SVM ani jedna.

FCBF algoritmus vybral 48 atribútov. Pri stromových klasifikátoroch boli pre všetky tri skupiny atribútov vo výsledku pre výber 48 atribútov a najlepším dosiahnutým výsledkom minimálne rozdiely, najväčší len 0.16%. Pre SVM malo 48 atribútov o viac ako o percento horší výsledok oproti najlepšiemu pre prvú aj druhú skupinu. Pre tretiu skupinu bol ale výber 48 atribútov lepší ako výber 500 atribútov. V druhej skupine žiadna selekcia 48 atribútov neprekonala ani pôvodný výsledok pred selekciou. Kým stromy udržujú pre tento počet atribútov takmer nezmenené skóre, pre SVM skóre už kolíše.

Pre stromy bolo na najlepší výsledok potrebných 500 atribútov vo všetkých skupinách, dokonca v druhej skupine malo 1000 atribútov horší výsledok. Pri SVM to už nie je také jasné. V jednom prípade bolo potrebných 1000, neskôr 500 atribútov a nakoniec 48 atribútov.

Konzistentné bolo pre stromové klasifikátory len XGBoost rozdelenie, ktoré vo všetkých ôsmich prípadoch zlepšilo výsledok. V jednom prípade sa to nepodarilo XGBoost zisku a LGBM rozdeleniu. LGBM zisk bol na treťom mieste so šiestimi zlepšeniami výsledku. Ďalší bol RFC, ktorý zlepšil výsledok 5-krát. Všetky SVM metódy okrem samotnej SVM knižnice (libsvm), vzájomná informácia (MI) a Chi-square zlepšili výsledok 3-krát. Pre SVM klasifikátory viedli SGD elasticnet a L2, SVC L1 a SVM s piatimi vylepšeniami. Na druhom mieste, len s dvomi vylepšeniami, sú všetky stromové selektory a SGD L1. Pri stromových klasifikátoroch sú najlepšie stromové selektory, ale niektoré rýchle metódy sú na úrovni SVM selektorov, ktoré nedosahujú veľmi dobré výsledky. Pri SVM klasifikátoroch vedú SVM metódy, po nich stromové, ale rýchle metódy nedosiahli zlepšenie pôvodného výsledku ani raz.

Najlepší výsledok dosiahli pre stromové klasifikátory najčastejšie LGBM rozdelenie a XGBoost rozdelenie – 5 krát. Na druhom mieste sú LGBM zisk a XGBoost zisk s tromi najlepšími výsledkami. Žiaden iný selektor už nedosiahol najlepší výsledok. Globálne najlepší výsledok pre stromy bol 99.69% a dosiahli ho obe XGBoost algoritmy s 500 atribútmi a obe LGBM algoritmy s 1000 atribútmi. Pre SVM malo SVC L1 až 5 najlepších výsledkov. SGD elasticnet, SGD L2 a SVM mali jeden. Pri najlepších výsledkov už vynikajú len selektory vlastnej rodiny klasifikátorov. Pre SVM bol globálne najlepší výsledok 99.92% a dosiahlo ho SVC L1 s 1000 atribútmi. Globálne najlepšie

výsledky len z druhej skupiny sú už trochu horšie: 99.38% pre stromy a 98.99% pre SVM, obe pri 500 atribútoch.

Globálne najlepší výsledok pre stromy bol dosiahnutý štyrikrát, ale ako v predošlých datasetoch za najlepší výsledok považujeme ten, ktorý potreboval menej atribútov. Obe XGBoost metódy našli rovnaké atribúty v prípadoch, keď hľadali 500 najlepších atribútov zo všetkých. Z nich bolo 112 z druhej skupiny a 82 z tretej. Globálne najlepší výsledok – 99.92% pre SVC dosiahlo SVC L1 s výberom 1000 atribútov zo všetkých. Z týchto atribútov bolo len 73 v druhej skupine a 21 v tretej. Toto je aj najlepší výsledok, aký sa nám podarilo dosiahnuť. Skupiny vybraných atribútov v najlepších výsledkov pre stromové aj SVM klasifikácie sa nachádzajú v tabuľke (Tab. 10).

Tab. 10 Atribúty, použité pri najlepších klasifikáciách v druhom datasete.

Názov skupiny atribútov	Počet pre stromy	Počet pre SVM
existencia 1-gramov textových reťazcov	150	32
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	111	89
frekvencia 2-gramov inštrukcií	53	172
existencia 2-gramov inštrukcií	-	133
existencia 1-gramov hexadecimálnej reprezentácie vzorky	48	-
frekvencia 2-gramov bajtov operačných kódov	38	75
normalizovaná frekvencia 2-gramov bajtov operačných kódov	-	32
normalizovaná frekvencia 1-gramov bajtov operačných kódov	-	3
informácie o sekciách	17	6
histogram entropie bajtov	14	5
existencia 2-gramov bajtov operačných kódov	13	139
existencia 1-gramov inštrukcií	-	14
frekvencia 1-gramov inštrukcií	6	2
existencia 2-gramov hexadecimálnej reprezentácie vzorky	6	106
existencia 2-gramov registrov	-	19
frekvencia 2-gramov registrov	6	4

normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	-	67
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	5	1
normalizovaná frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	-	8
frekvencia 1-gramov textových reťazcov	5	39
metadáta	5	1
informácie o textových reťazcoch	5	1
existencia 2-gramov textových reťazcov	4	7
informácie o importovaných knižniciach	4	3
informácie o overlay dátach	3	2
frekvencia 2-gramov textových reťazcov	2	13
informácie o importovaných funkciách	2	16
frekvencia 1-gramov bajtov operačných kódov	1	-
frekvencia 1-gramov registrov	1	-
informácie o disasemblovanom súbore	1	1
informácie o zdrojoch	-	4
frekvencia 2-gramov písmen v textových reťazcoch	-	2
existencia 1-gramov písmen v textových reťazcoch	-	1
frekvencia 1-gramov písmen v textových reťazcoch	-	1
existencia 1-gramov bajtov operačných kódov	-	1
existencia 1-gramov registrov	-	1

Najčastejšie atribúty pre stromové selekcie sú textové reťazce, obsiahnuté vo vzorkách malvéru. Ďalšie v poradí sú n-gramy: hexadecimálnej verzie súboru, inštrukcií, binárne 1-gramy hex. súboru a operačného kódu. Zaujímavé je, že medzi najčastejšími atribútmi sú aj binárne 1-gramy, ktoré nesú z n-gramov najmenej informácií.

Na rozdiel od stromov pri najlepšom výsledku pre SVM klasifikátory už prevažujú frekvenčné 2-gramy. Oproti stromom klesla dôležitosť reťazcov, ale hexadecimálna verzia súboru a inštrukcie stále patria medzi najdôležitejšie. Normalizované n-gramy

v najlepšom výsledku pre stromy vôbec neboli. V selekcii pre SVM sa vyskytujú viackrát.

Najmenšie množstvo atribútov, ktoré sme vybrali, bolo 48. Najlepšie výsledky pre tento počet sme dosiahli pri výbere zo všetkých atribútov. Pre stromy malo najlepší výsledok XGBoost rozdelenie – 99.53%. Z vybraných atribútov bolo 17 v druhej a 13 v tretej skupine. Ako sme spomínali pri prvom datasete, SVM klasifikátory majú pri veľmi malom počte atribútov zvyčajne nižšie výsledky. V tomto datasete sa to potvrdilo a najlepší výsledok pre SVM bol 98.53% a dosiahol SVC L1 selektor. Len 10 z atribútov, ktoré vybral SVC L1 selektor, je z druhej skupiny (do 1000 atribútov) a len 4 atribúty sú z tretej skupiny (do 100 atribútov). Vybrané atribúty sú popísané v tabuľke (Tab. 11).

Tab. 11 Najlepšie selekcie z 48 atribútov – druhý dataset.

Názov skupiny atribútov	Počet pre stromy	Počet pre SVM
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	13	2
frekvencia 2-gramov inštrukcií	10	4
existencia 2-gramov inštrukcií	-	5
informácie o sekciách	6	3
existencia 1-gramov textových reťazcov	4	7
metadáta	3	
existencia 2-gramov textových reťazcov	2	4
informácie o overlay dátach	2	
frekvencia 1-gramov inštrukcií	1	
frekvencia 1-gramov textových reťazcov	1	7
frekvencia 2-gramov textových reťazcov	1	1
histogram entropie bajtov	1	1
informácie o disasemblovanom súbore	1	
informácie o importovaných funkciách	1	1
informácie o importovaných knižniciach	1	
informácie o textových reťazcoch	1	

normalizovaná frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	-	3
normalizovaná frekvencia 2-gramov bajtov operačných kódov	-	3
existencia 2-gramov registrov	-	2
existencia 1-gramov hexadecimálnej reprezentácie vzorky	-	1
normalizovaná frekvencia 1-gramov bajtov operačných kódov	-	1
existencia 2-gramov hexadecimálnej reprezentácie vzorky	-	1
existencia 2-gramov bajtov operačných kódov	-	1
normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	-	1

Z vybraných atribútov pre stromy vyplýva, že aj pri tak malom počte sú 2-gramy stále dôležité. Medzi najčastejšie skupiny sa už dostávajú aj také, ktoré majú málo členov, ako sú sekcie, metadáta a overlay. SVC L1 selektor zase za najdôležitejšie označil reťazce, po nich inštrukcie, ale ani jedných nie je viac ako v globálne najlepšom výsledku pre SVM. Oproti stromom sú aj pri tak malom počte atribútov stále najdôležitejšie 1-gramy a vo väčšej miere sa vyskytuje ich existenčná variácia. Malé skupiny atribútov sa tu vyskytujú menej. Znovu sa tu oproti stromom vyskytujú normalizované n-gramy.

4.3 Tretí dataset

Tento dataset by mal obsahovať len zabalené, obfuskované a šifrované vzorky malvéru. Aj v tomto datasete mal až na tri prípady lepšie výsledky dataset označený zhlukovaním. Keďže počet prípadov, kde bol zhlukovaný dataset lepší, vysoko prevažoval počet tých, kde bol horší, rozhodli sme sa aj v tomto prípade použiť zhlukovanie.

Pred selekciou boli výsledky podobné predošlým datasetom. Najlepšie skóre mala na stromových klasifikátoroch druhá skupina – 98.05%, na SVM prvá – 97.33%, aj keď rozdiel oproti druhej bol len 0.25%. Tretia skupina bola zase na stromoch lepšia ako všetky atribúty, ale pri SVM bola o viac ako 3% horšia. Po selekcii bola druhá skupina

pri stromoch blízko straty jedného percenta, ale stále pod ním – 0.97%. Pri SVM ale klesla až o 1.21%. Tretia skupina klesla o viac ako percento už aj pri stromoch, pri SVM dokonca o viac ako 5%, čo je najväčší pokles, aký sa v nejakom datasete vyskytol. V tomto datasete dokonca aj druhá skupina pre stromy takmer dosiahla pokles o percento. Vzhľadom na povahu datasetu to je prirodzené.

FCBF algoritmus vybral 62 atribútov. Pre stromové klasifikátory sa s nimi dosiahol najlepší výsledok pre selekcie len z druhej a tretej skupiny atribútov. Oproti najlepšiemu výsledku, dosiahnutému selekciou zo všetkých atribútov, klesol výsledok pri selekcii 62 atribútov len o 0.13%. Dokonca aj pri SVM klasifikátoroch mal výber 62 atribútov pre selekcie z druhej a tretej skupiny najlepší výsledok. Pri selekcii 62 atribútov ale klesol výsledok oproti najlepšej selekcii zo všetkých atribútov o viac ako 1.5%. Pre stromové klasifikátory dosahuje tak malý počet atribútov stále výborné výsledky (rovnako ako pri ostatných datasetoch pri výbere FCBF), ale pri SVM sú už výsledky zmiešané.

Pre stromy bolo pre druhú a tretiu skupinu potrebných len 62 atribútov na najlepší výsledok, ale pre všetky atribúty bolo potrebných až 1000. Pre SVM tiež pre druhú a tretiu skupinu stačilo 62 a pre všetky atribúty 500 vybraných atribútov.

Vylepšiť výsledok pre stromové klasifikátory sa najviac darilo LGBM rozdeleniu, ktorý ho nevylepšil len raz. Dvakrát sa nepodarilo vylepšiť výsledok LGBM zisku a trikrát obom XGBoost algoritmom a SVC L1. Pri SVM klasifikátoroch mali 7 zlepšení LGBM zisk a SVC L1. O jedno zlepšenie menej malo SGD L2. Päť zlepšení malo LGBM rozdelenie, SGD elasticnet, a SVM. V tomto datasete sa SVM selektor priblížil stromovým pri stromových klasifikáciách a stromový selektor bol dokonca na úrovni najlepšieho SVM selektora pre SVM klasifikácie.

Najlepší výsledok pre stromové klasifikátory najčastejšie dosiahli obe LGBM algoritmy – len trikrát. Dvakrát ho dosiahlo XGBoost rozdelenie a raz SGD elasticnet. Globálne najlepší výsledok bol 99.39% pre LGBM zisk a výber 1000 atribútov. Pre SVM klasifikátory mal až 5-krát najlepší výsledok SVC L1. Raz ho dosiahol SGD L1 a L2 a Gini index. V tomto datasete mala prvýkrát najlepší výsledok v experimente metóda, ktorá nebola embedded. SVC L1 dosiahlo globálne najlepší výsledok – 99.87% s 500 atribútmi. V druhej skupine bol globálny výsledok pre stromy 98.9% a pre SVM 98.66%. Sú to najhoršie výsledky zo všetkých datasetov, ale pre oba bolo potrebných len 62 atribútov.

Globálne najlepší výsledok pre stromy dosiahol LGBM zisk pri výbere 1000 atribútov zo všetkých. Z vybraných atribútov bolo 154 v druhej skupine a 40 v tretej. SVM klasifikácie mali globálne najlepší výsledok pre SVC L1, ktoré vyberalo 500 atribútov zo všetkých. Z výberu bolo 99 atribútov v druhej skupine a 12 v tretej. Vybrané skupiny atribútov sú popísané v tabuľke (Tab. 12).

Tab. 12 Atribúty, použité pri najlepších klasifikáciách v treťom datasete.

Názov skupiny atribútov	Počet pre stromy	Počet pre SVM
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	233	42
frekvencia 2-gramov inštrukcií	199	117
frekvencia 2-gramov bajtov operačných kódov	192	24
normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	72	8
existencia 2-gramov bajtov operačných kódov	62	56
normalizovaná frekvencia 2-gramov bajtov operačných kódov	46	16
histogram entropie bajtov	35	
existencia 1-gramov textových reťazcov	22	18
existencia 2-gramov hexadecimálnej reprezentácie vzorky	19	7
existencia 1-gramov inštrukcií	-	11
frekvencia 1-gramov inštrukcií	15	5
frekvencia 1-gramov bajtov operačných kódov	14	-
informácie o sekciách	14	6
existencia 2-gramov registrov	-	11
frekvencia 2-gramov registrov	12	3
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	9	-
frekvencia 1-gramov textových reťazcov	9	44
existencia 2-gramov textových reťazcov	8	9

metadáta	7	-
informácie o importovaných funkciách	6	27
frekvencia 1-gramov písmen v textových reťazcoch	5	-
informácie o textových reťazcoch	5	2
informácie o importovaných knižniciach	3	2
informácie o overlay dátach	3	2
normalizovaná frekvencia 1-gramov bajtov operačných kódov	2	-
existencia 2-gramov inštrukcií	2	73
informácie o disasemblovanom súbore	2	1
informácie o zdrojoch	2	-
frekvencia 2-gramov textových reťazcov	1	16
entropia spustiteľného súboru	1	-

V najlepšej selekcii pre stromové klasifikátory sú najčastejšie prevažne frekvenčné 2-gramy. Znovu hexadecimálna verzia súboru a inštrukcie patria medzi najdôležitejšie. Zaujímavé je veľké množstvo normalizovaných 2-gramov. V predošlom datasete ich stromové selektory nevyberali a vyskytovali sa len v SVM selekciách. V najlepšej selekcii pre SVM klasifikátory sú rovnako ako v predošlých datasetoch časté 2-gramy inštrukcií a hexadecimálna verzia súboru. Neobvyklejšie sú importované funkcie a textové reťazce.

Najmenej vybraných bolo 62 atribútov. Stromové selektory dosiahli najlepší výsledok pre výber zo všetkých atribútov – 99.26%, SVM z druhej skupiny. SVM mali tak ako v predošlých datasetoch slabší výsledok – 98.66%. Stromové selektory dosiahli najlepší výsledok v dvoch prípadoch – LGBM rozdelenie a XGBoost rozdelenie. LGBM rozdelenie vybralo 31 atribútov z druhej a 8 z tretej skupiny. XGBoost rozdelenie vybralo len 18 atribútov z druhej a 5 z tretej skupiny. SVC L1 malo všetky atribúty z druhej skupiny, z nich len 7 z tretej. Selektované atribúty sú popísané v tabuľke (Tab. 13).

Tab. 13 Najlepšie selekcie z 62 atribútov – tretí dataset.

Názov skupiny atribútov	LGBM rozdelenie	XGBoost rozdelenie	SVC L1
frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	14	22	-
frekvencia 2-gramov textových reťazcov	-	-	12
histogram entropie bajtov	8	3	1
existencia 1-gramov textových reťazcov	7	7	11
frekvencia 2-gramov inštrukcií	7	4	-
existencia 1-gramov inštrukcií	-	-	8
existencia 2-gramov bajtov operačných kódov	-	3	-
frekvencia 2-gramov bajtov operačných kódov	5	13	-
frekvencia 1-gramov textových reťazcov	3	2	-
existencia 2-gramov textových reťazcov	3	2	2
metadáta	3	3	1
frekvencia 1-gramov inštrukcií	2	-	7
informácie o importovaných funkciách	2	-	11
informácie o sekciách	2	-	4
normalizovaná frekvencia 2-gramov bajtov operačných kódov	1	-	-
normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky	1	-	-
informácie o disasemblovanom súbore	1	-	1
informácie o overlay dátach	-	2	1
frekvencia 1-gramov hexadecimálnej reprezentácie vzorky	-	1	-
normalizovaná frekvencia 1-gramov bajtov operačných kódov	-	-	2
informácie o importovaných knižniciach	-	-	1

V atribútoch, selektovaných cez LGBM rozdelenie, bol histogram entropie druhou najčastejšou skupinou atribútov. Pri selekcii XGBoost rozdelením sú výsledky na prvých miestach podobné ako v predošlých datasetoch, ale chýbajú tu sekcie a importy. SVC L1 selektor vybral zase nezvyčajne veľa importovaných funkcií.

4.4 Zhrnutie výsledkov

Používali sme dva typy označovania: zhlukovaním a konsenzom, aj keď v druhom prípade sme striktne nevyžadovali konsenzus, aby sme zachovali väčší počet tried. Z dôvodu, že nie vždy sa počet tried, vytvorených oboma metódami označovania zhodoval, je ťažké porovnať efektivitu oboch metód. V našich troch datasetoch ale jasne viedla metóda zhlukovaním – z 26 klasifikácií pred selekciami v prvom datasete bol len raz dosiahnutý nižší výsledok pri označení zhlukovaním, v ostatných 25 prípadoch sa s použitím zhlukovania dosiahla vyššia presnosť klasifikácie . Tento pomer výsledkov sa zachoval aj po tom, čo sme vylúčili outliere z datasetu, označeného konsenzom. V druhom datasete bola pri všetkých 26 klasifikáciách dosiahnutá vyššia presnosť pri použití označenia zhlukovaním. V treťom datasete bola v 23 prípadoch z 26 dosiahnutá vyššia presnosť klasifikácie pri použití označenia zhlukovaním. V kapitole 3.1.1 sme spomenuli rôzne problémy, ktoré súvisia s použitím konsenzu a spôsoby, akými sa im zhlukovanie môže vyhnúť. Z týchto dôvodov si myslíme, že označovanie zhlukovaním pre datasety malvéru lepšie určí vzorkám ich triedy.

Najviac najlepších výsledkov pre SVM klasifikátory mala vo všetkých troch datasetoch SVC L1 metóda selekcie. Na druhom mieste sa vyskytlo viacero metód, ale len SGD L2 bola na druhom mieste vo všetkých datasetoch. Pre selekcie, ktoré sme kontrolovali na stromových klasifikátoroch, bolo na prvom mieste tiež viacero metód, z nich len LGBM rozdelenie bolo na prvom mieste vo všetkých datasetoch. Druhá najlepšia metóda je XGBoost rozdelenie, ktoré malo v jednom z datasetov najviac najlepších výsledkov (druhý dataset, spolu s LGBM rozdelením) a v dvoch bolo na druhom mieste. Metódy, používajúce ohodnotenie rozdelením (split), dosahujú najlepší výsledok častejšie ako tie, ktoré ohodnocujú ziskom (gain). Vo všetkých datasetoch sa ukázalo, že embedded selekčné metódy majú najlepšie výsledky. Tieto metódy mali často najlepšie výsledky aj vtedy, keď bol použitý klasifikátor iného typu. Príkladom je LGBM zisk, ktorý mal spolu so SVC L1 najviac najlepších výsledkov pre SVM klasifikácie v treťom datasete. Filtrové metódy získali zo všetkých troch datasetov len jeden najlepší

výsledok – Gini index v treťom datasete. Podľa týchto výsledkov filtrové metódy nie sú viac generalizovateľné ako embedded metódy.

Globálne najlepší výsledok selekcie pre SVM klasifikácie mala metóda SVC L1 vo všetkých troch datasetoch. V prvom datasete výsledok zdieľala s metódou SGD L2, ktorá ale potrebovala viac atribútov (1000 oproti 500). Pre stromové klasifikácie už nebol najlepší selektor taký jasný. V prvom datasete dosiahlo najlepší výsledok LGBM rozdelenie. V druhom ho dosiahli obe XGBoost metódy aj obe LGBM metódy, ale LGBM metódy potrebovali 1000 atribútov, kým XGBoost len 500. V treťom datasete bol najlepší LGBM zisk. Všetky tieto metódy používajú gradient boosting algoritmus, ktorý teda môžeme pokladať za najlepší. Najlepšie výsledky boli dosiahnuté v druhom datasete: 99.69% pre stromové klasifikátory a 99.92% pre SVM. Tretí dataset, v ktorom sme očakávali najhoršie výsledky, ich pre stromové klasifikácie aj mal, ale rozdiel bol minimálny, keďže výsledok bol 99.39% a prvý dataset mal 99.67%. Pre SVM klasifikácie mal tretí dataset dokonca lepší výsledok ako prvý: 99.87% oproti 99.67%, takže prítomnosť obfuskovaných, šifrovaných a zabalených vzoriek sa neodrazila na presnosti klasifikácií. Tretí dataset mal najhoršie výsledky aj pred použitím selekcie.

Dôležitý je aj počet atribútov, ktoré bolo potrebné použiť na získanie najlepšieho výsledku. Keďže my sme obmedzili počet atribútov selekcie na 1000, tak v prípade, že na dosiahnutie najlepšieho výsledku sa použilo všetkých 1000 atribútov, je možné, že ďalšie zvyšovanie počtu atribútov by viedlo k ďalšiemu zlepšeniu. Pri našich testoch v prvom datasete pre stromové aj pre SVM klasifikátory boli dva prípady najlepšieho výsledku a pre oba typy jeden prípad potreboval 500 a druhý 1000 atribútov. V druhom datasete boli pre stromové klasifikátory až štyri výsledky, z toho dva potrebovali 500 a dva 1000 atribútov. Pre SVM bolo potrebných 1000 atribútov. V treťom datasete zase SVM potrebovalo 500 atribútov a stromy 1000. V každom datasete teda bolo možné aspoň jeden z dvojice najlepších výsledkov (pre stromy a SVM) dosiahnuť bez použitia 1000 atribútov. Vo všetkých prípadoch však boli tieto výsledky lepšie, ako pri použití všetkých atribútov bez selekcie. Selekcia teda určite má zmysel.

Pozreli sme sa aj na veľkosť poklesu presnosti klasifikácie pri použití veľmi malého počtu atribútov. Na výber počtu sme použili FCBF, ktorý má vybrať minimálnu množinu atribútov, ktoré dobre popisujú dataset. Selekciu vykonáva tak, že postupne vyhadzuje redundantné atribúty. V jednotlivých datasetoch nám FCBF selektovalo 79, 48 a 62 atribútov, čo sú rádovo desaťtisíciny pôvodného počtu atribútov. Napriek takému

radikálnemu obmedzeniu počtu atribútov výsledky klesali minimálne. V prvom datasete boli najlepšie výsledky pre stromy a SVM s týmto počtom atribútov: 99.66% a 98.9%; v druhom 99.53% a 98.53% a nakoniec v treťom 99.26% a 98.66%. Vidíme, že pri SVM klasifikáciách už klesli výsledky vo všetkých datasetoch pod 99%, čo predstavuje v druhom aj treťom datasete pokles o viac ako percento, kým stromové klasifikátory znášajú malý počet atribútov lepšie. Najväčší pokles pre stromové klasifikátory bol len 0.16% v druhom datasete, kde najlepší výsledok potreboval 500 atribútov a FCBF vybralo len 48 atribútov. Celkovo SVM oveľa horšie znáša veľmi malý počet atribútov, ale pri správnom množstve má lepšie výsledky ako stromové algoritmy.

Ďalšia výskumná otázka, na ktorú sme sa sústredili, sa týka toho, či je potrebné vôbec extrahovať mnohodimenzionálne skupiny atribútov, a či nepostačia aj málodimenzionálne (druhá skupina do 1000 atribútov). V prvom datasete bol pokles presnosti najlepšieho výsledku pri použití druhej skupiny atribútov minimálny (99.66 vs 99.67% a 99.55% vs 99.67%). V druhom datasete už druhá skupina spôsobila pokles pod 99% (98.99% vs 99.92%) pri SVM. V treťom datasete už oba najlepšie výsledky pre druhú skupinu klesli pod 99% (98.9% a 98.66%). Najväčší pokles výsledku bol v druhom datasete pre SVM a výber 48 atribútov - 1.39%. Ostatné poklesy sú do jedného percenta, čo je veľmi dobrý výsledok, keďže druhá skupina zvyčajne tvorila rádovo len stotinu všetkých atribútov. V druhej skupine boli po prvotnej selekcii a pred druhotnou selekciou tisíce atribútov, všetkých boli státisíce. Tretia skupina mala horšie výsledky, aj na stromoch už bol najväčší pokles nad jedno percento. Pre SVM bol najväčší pokles v treťom datasete – viac ako 5%.

Náš hlavný cieľ bolo zistiť, ktoré skupiny atribútov sú najdôležitejšie. Z tohto dôvodu sme urobili prienik atribútov pre najlepšie výsledky na všetkých troch datasetoch. Keď bolo viacero najlepších výsledkov v jednom datasete, tak sme požadovali, aby v prieniku boli atribúty z výsledku, v ktorom bolo použitých najmenej atribútov. Ak sa počet atribútov rovnal, tak atribút mohol pochádzať z hociktorého výsledku (or). Prienik sme urobili osobitne pre stromové a SVM klasifikátory. Následne sme urobili aj globálny prienik pre všetky klasifikátory. V tomto prípade sme hľadali atribúty, ktoré sa vyskytujú v najlepšom výsledku pre obe skupiny klasifikátorov súčasne v každom datasete. Rovnaké prieniky sme robili aj pre najlepšie výsledky so selekciou atribútov, ktorých počet vybralo FCBF. Skupiny atribútov, ktoré ostanú v týchto prienikoch, by mali patriť

medzi najdôležitejšie. Je to z dôvodu, že vo všetkých datasetoch sa dostali do skupiny len niekoľkých desiatok atribútov, ktoré potom mali najlepší výsledok.

Skupiny atribútov, použité v najlepších výsledkoch pre obe skupiny klasifikátorov boli tieto:

- existencia 1-gramov textových reťazcov,
- frekvencia 1-gramov inštrukcií,
- frekvencia 1-gramov textových reťazcov,
- existencia 2-gramov hexadecimálnej reprezentácie vzorky,
- existencia 2-gramov textových reťazcov,
- existencia 2-gramov bajtov operačných kódov,
- frekvencia 2-gramov bajtov operačných kódov,
- frekvencia 2-gramov hexadecimálnej reprezentácie vzorky,
- frekvencia 2-gramov inštrukcií,
- frekvencia 2-gramov registrov,
- frekvencia 2-gramov textových reťazcov,
- informácie o disasemblovanom súbore,
- informácie o importovaných funkciách,
- informácie o importovaných knižniciach,
- informácie o sekciách.

Napriek tvrdeniu, že 1-gramy sú neefektívne [5], vidíme, že pri dosiahnutí najlepších výsledkov sa používajú, dokonca aj ich binárna verzia.

Pre stromové klasifikátory obsahoval prienik najlepších výsledkov okrem atribútov pre obe skupiny klasifikátorov aj tieto skupiny atribútov:

- frekvencia 1-gramov bajtov operačných kódov,
- frekvencia 1-gramov hexadecimálnej reprezentácie vzorky,
- histogram entropie bajtov,
- metadáta,
- informácie o overlay dátach
- informácie o textových reťazcoch

Pre n-gramy sa zmenilo len to, že v dvoch prípadoch použili okrem 2-gramov aj 1-gramy, ale z rovnakého zdroja. Pribudol ale histogram entropie bajtov, metadáta z hlavičky, informácie o overlay (dáta mimo sekcií). Okrem n-gramov z textových reťazcov sa využili aj nepriame informácie o nich (dĺžka riadkov, veľkosť, entropia...).

Pre SVM klasifikátory zase najlepšie výsledky použili okrem atribútov, najlepších v oboch skupinách klasifikátorov, aj tieto skupiny atribútov:

- existencia 1-gramov inštrukcií,
- normalizovaná frekvencia 2-gramov bajtov operačných kódov,
- normalizovaná frekvencia 2-gramov hexadecimálnej reprezentácie vzorky,
- existencia 2-gramov inštrukcií,
- existencia 2-gramov registrov.

Tu sa pridali len n-gramy, ale zdroje ostali rovnaké. SVM využívajú aj normalizované n-gramy a viac používajú 2-gramy.

Ak sa použilo len minimum atribútov (počet vybraný FCBF), tak prienik už mohol obsahovať len atribúty z druhej skupiny (do tisíc atribútov v skupine). Je to spôsobené tým, že z druhej skupiny boli v druhom datasete oba najlepšie výsledky (pre SVM aj stromové klasifikátory) a v treťom najlepší výsledok pre SVM. Prienik pre oba typy klasifikátorov bol rovnaký ako prienik len pre SVM klasifikátory a obsahoval tieto skupiny atribútov:

- existencia 1-gramov textových reťazcov,
- existencia 2-gramov textových reťazcov,
- histogram entropie bajtov,
- informácie o importovaných funkciách,
- informácie o sekciách.

Aj keď prvotná selekcia vylúčila takmer všetky 2-gramy z textových reťazcov, tie, ktoré ostali, patria medzi najdôležitejšie atribúty. Histogram entropie bajtov je snaha o kompaktné zachytenie rozloženia frekvencie bajtov a ich entropie v súbore, a ako vidíme, tento spôsob je úspešný. Stromové klasifikátory mali v prieniku navyše:

- frekvencia 1-gramov inštrukcií,
- informácie o disasemblovanom súbore,

- metadáta,
- informácie o overlay dátach.

Posledná výskumná otázka, na ktorú sme sa zamerali, sú skupiny atribútov, ktoré majú dobré výsledky len na treťom datasete (ktorý obsahuje len obfuskované, zabalené a šifrované vzorky). Atribúty sme získali z rozdielu atribútov, použitých v najlepších výsledkoch pre oba typy klasifikátorov v treťom datasete a v ostatných dvoch. Zistili sme, že neexistuje skupina atribútov, ktoré je použitá v nejakom najlepšom výsledku len v treťom datasete a v ostatných nie. Z toho vyplýva, že skupiny atribútov, ktoré majú dobré výsledky pre zabalené, šifrované a obfuskované vzorky, budú mať dobré výsledky aj pre čisté vzorky.

Záver

V práci sme riešili problematiku klasifikácie malvéru s použitím selekcie atribútov. Zo vzoriek malvéru sme vytvorili dataset, ktorý sme rozdelili na tri časti – čisté vzorky, vzorky, ktoré sú pravdepodobne zabalené, šifrované alebo obfuskované, a nakoniec sme ešte spojili obe časti do zmiešaného datasetu. V druhej kapitole sme spracovali návrh postupu a porovnanie jednotlivých metód selekcie a klasifikácie.

Dataset sme následne označovali – priradzovali sme vzorkám triedu. Vyskúšali sme viacero prístupov určovania podobnosti malvéru. Cez službu VirusTotal sme získali názvy vzoriek z vybraných antivírusových programov. Tieto názvy sme potom použili pri označovaní konsenzom a označovaní zhlukovaním cez metódu HDBSCAN a metriku Levenshtein distance. Pri označovaní konsenzom sme z názvov odstránili generické slová, zlúčili aliasy, rozdelili sme názvy a až potom sme použili konsenzus. Robili sme pokusy aj s kombinovaným prístupom – získanie možných tried konsenzom a vytváranie skupín cez Levenshtein distance. Ukázalo sa, že toto riešenie je menej robustné ako možnosť, pri ktorej sa do úvahy berú celé názvy. Problémom boli krátke názvy tried, ktoré sa navzájom zlučovali skôr, ako sa vytvorili triedy z dlhých názvov. Metóda označovania zhlukovaním dosahovala pri klasifikácii konzistentne lepšie výsledky. Po označení sme vybrali z datasetov menšiu časť tak, aby všetky triedy mali približne rovnaký počet vzoriek. Chceli sme zabrániť tomu, aby jedna trieda vplývala na výsledky vo väčšej miere ako ostatné.

Zo vzoriek sme potom extrahovali množstvo statických atribútov. Počas extrakcie sme na jednotlivých skupinách atribútov vykonávali aj prvotnú selekciu. Najväčšie skupiny atribútov tvorili n-gramy. Pôvodne sme extrahovali n-gramy až po n=3, teda aj 3-gramy. Týmto sme získali po prvotnej selekcii približne milión atribútov, čo bolo príliš veľa na ďalšie spracovanie, preto sme 3-gramy vylúčili. Počet atribútov, ktoré prešli prvotnou selekciou, sa potom v jednotlivých datasetoch pohyboval okolo 500 000. Počet vzoriek v jednotlivých datasetoch sa pohyboval od 822 po 1559. Po extrakcii sme atribúty spracovali – najprv sme ich diskretizovali a potom štandardizovali.

Ďalším krokom bolo porovnanie metód klasifikácie a selekcie a výber metód, ktoré použijeme. Kritériom výberu bola časová zložitosť vzhľadom na počet atribútov. Potom sme metódy testovali na menších datasetoch. Testom sme vylúčili metódy, ktoré potrebovali príliš veľa pamäte alebo mali konzistentne slabé výsledky.

Následne sme vykonali klasifikáciu nad celými datasetmi. Popis implementácie celého postupu sa nachádza v tretej kapitole. Pri klasifikovaní sme použili stratifikovanú 10-fold krosvalidáciu, keďže sme nemali dostatok vzoriek na vytvorenie testovacích datasetov a chceli sme sa vyhnúť overfittingu. Výsledky klasifikácie potom uvádzame ako priemer presnosti (accuracy) na 10-fold krosvalidácii. Pred použitím selekcie sme určili počet selektovaných atribútov – najprv sme vyberali tisíc atribútov, čo je približne priemerná veľkosť datasetu, potom 500 a posledný počet určil algoritmus FCBF a pre všetky datasety bol menší ako 100. Potom sme použili jednotlivé metódy selekcie a klasifikovali sme len na selektovaných atribútoch. Po získaní výsledkov na selektovaných atribútoch sme vyvodili jednotlivé závery. Analýzu výsledkov a vyhodnotenie sme spracovali vo štvrtej kapitole.

Výskumným cieľom bolo aj zistiť, či použitie len statických atribútov postačuje na presnú klasifikáciu malvéru. Výsledky ukázali, že statické atribúty postačujú. Už pred selekciou boli dosiahnuté pomerne vysoké hodnoty presnosti klasifikácií – aj viac ako 98%. Po selekcii bola dosiahnutá na každom datasete presnosť nad 99%.

Naším cieľom bolo ukázať, že selekcia atribútov má pri klasifikácii malvéru kladný dopad na výsledky, aj keď sa selektuje len veľmi malé množstvo atribútov. Selektovali sme maximálne tisíc atribútov z datasetov v ktorých ich bolo približne 500 000. Napriek tomu sme vo všetkých datasetoch pre všetky počty vybraných atribútov dosiahli lepší výsledok ako pred selekciou, dokonca aj vtedy, keď sa selektovali rádovo desiatky atribútov. Pre počet atribútov, ktorý určila metóda FCBF a použití stromových klasifikátorov, bolo možné dosiahnuť presnosť nad 99% vo všetkých datasetoch. SVM klasifikátory pri tak malom počte už nedosahovali až také dobré výsledky, ale stále boli vo všetkých datasetoch schopné dosiahnuť presnosť nad 98%.

Najlepšie výsledky boli dosiahnuté podľa očakávania použitím atribútov vybraných embedded metódami selekcie pri klasifikácii vlastným typom klasifikátora. S týmito atribútmi sa ale dosahovali veľmi dobré výsledky aj pri klasifikácii iným typom klasifikátora. Použitím atribútov, vybraných filtrovými metódami, boli dosiahnuté najnižšie výsledky. Podľa týchto výsledkov sú embedded metódy selekcie generalizovateľné.

Jedným z našich cieľov bolo zistiť, aké výsledky budú dosiahnuté použitím selekcie len zo skupín atribútov, ktoré majú málo prvkov. Zaujímalo nás, či nestačí extrahovať len málodimenzionálne skupiny atribútov. Preto sme pre každý dataset vytvorili ešte dva

ďalšie datasety. Jeden obsahoval len atribúty zo skupín, ktoré mali menej ako tisíc prvkov, v druhom museli mať menej ako sto prvkov. Selektovaním zo skupín do tisíc atribútov sa dosiahli veľmi dobré výsledky, najlepšie výsledky boli vždy nad hranicou 98%. Použitie skupín do sto atribútov sa už ukázalo nedostatočné, najlepšie výsledky klesali aj pod 95%.

Hlavným problémom použitia statických atribútov je to, že útočníci ich zakrývajú šifrovaním, obfuskovaním alebo zabalením malvéru. Z toho dôvodu bolo jedným z našich cieľov zistiť, či existujú skupiny atribútov, ktoré boli použité pri dosiahnutí najlepších výsledkov len pre dataset, ktorý obsahuje šifrované, obfuskované alebo zabalené vzorky. Tieto skupiny atribútov by boli špeciálne užitočné pri takýchto vzorkách. Zistili sme, že neexistuje skupina atribútov, ktorá je použitá v nejakom najlepšom výsledku len v tomto datasete, a v ostatných nie. Okrem toho bolo výskumným cieľom zistiť, či dokážu dosahovať statické atribúty dobré výsledky aj napriek obfuskácii, šifrovaniu a zabaleniu, a ako veľmi tieto skutočnosti ovplyvnia výsledky. Výsledky boli veľmi vysoké – najlepší výsledok pre stromové aj SVM klasifikátory mal hodnotu nad 99%. Z toho vyplýva, že statické atribúty je možné použiť pre presnú klasifikáciu aj vtedy, keď boli použité metódy, ktoré majú tieto atribúty zakryť a výsledky klasifikácie nebudú touto skutočnosťou ovplyvnené.

V budúcnosti by bolo dobré overiť naše zistenia na väčšom datasete a zahrnúť do selekcie aj dynamické atribúty. Bolo by potrebné zamerať sa na problematiku označovania malvéru zhlukovaním a porovnať viac metód zhlukovania a prístupov k zhlukovaniu malvéru. Je dôležité zamerať sa na zhlukovanie len pomocou atribútov, získaných z malvéru, a nie externých zdrojov dát, ako je ochrana koncových zariadení. Model, ktorý by na označovanie nepotreboval dáta od služieb ako je VirusTotal, by bol schopný samostatne a lokálne spracovať, označiť a klasifikovať malvér. Dôležitá vlastnosť by bolo aj rozlišovanie nových tried. Pri porovnaní výsledkov označenia zhlukovaním a konsenzom bude potrebné vytvoriť brať do úvahy optimálny počet členov konsenzu, optimálny počet zhlukov, možnosti filtrovania generických vzoriek a outlierov. Až potom bude možné vytvoriť objektívnu metriku na porovnanie týchto prístupov k označovaniu.

Zoznam použitej literatúry

1. McAfee LabsThreats Report in August 2019. 2019 [online] Dostupné na: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf
2. YAN, Jinpei; QI, Yong; RAO, Qifan. Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks, 2018, 2018.
3. ISLAM, Rafiqul, et al. Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications, 2013, 36.2: 646-656.
4. SAXE, Joshua; SANDERS, Hillary. Malware Data Science: Attack Detection and Attribution. 2018.
5. RAFF, Edward, et al. An investigation of byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques, 2018, 14.1: 1-20.
6. YAN, Guanhua; BROWN, Nathan; KONG, Deguang. Exploring discriminatory features for automated malware classification. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Berlin, Heidelberg, 2013. p. 41-61.
7. DENG, Houtao; RUNGER, George. Feature selection via regularized trees. In: The 2012 International Joint Conference on Neural Networks (IJCNN). IEEE, 2012. p. 1-8.
8. WANG, L.; LIU, J.; CHEN, X. Microsoft Malware Classification Challenge (BIG 2015) First Place Team: Say No To Overfitting (2015). 2015.
9. HWANGA, Seong Oun; NGUYENB, Trong Kha; LY, Vu Duc. Feature selection for malware classification. Technical report, January, 2018.
10. PE Format. 2019 [Online] Dostupné na: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format
11. MENAHEM, Eitan, et al. Improving malware detection by applying multi-inducer ensemble. Computational Statistics & Data Analysis, 2009, 53.4: 1483-1494.
12. SANTOS, Igor, et al. Opem: A static-dynamic approach for machine-learning-based malware detection. In: International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions. Springer, Berlin, Heidelberg, 2013. p. 271-280.

13. JAIN, Sachin; MEENA, Yogesh Kumar. Byte level n–gram analysis for malware detection. In: International Conference on Information Processing. Springer, Berlin, Heidelberg, 2011. p. 51-59.
14. LIN, Chih-Ta, et al. Feature Selection and Extraction for Malware Classification. J. Inf. Sci. Eng., 2015, 31.3: 965-992.
15. YUXIN, Ding; SIYI, Zhu. Malware detection based on deep learning algorithm. Neural Computing and Applications, 2017, 1-12.
16. SHABTAI, Asaf, et al. Detecting unknown malicious code by applying classification techniques on opcode patterns. Security Informatics, 2012, 1.1: 1.
17. SHABTAI, Asaf, et al. "Andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, 2012, 38.1: 161-190.
18. AGGARWAL, Charu C. (ed.). Data classification: algorithms and applications. CRC press, 2014.
19. GU, Quanquan; LI, Zhenhui; HAN, Jiawei. Generalized fisher skóre for feature selection. arXiv preprint arXiv:1202.3725, 2012.
20. SANTOS, Igor, et al. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences, 2013, 231: 64-82.
21. KARAMPATZIAKIS, Nikos, et al. Using file relationships in malware classification. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Berlin, Heidelberg, 2012. p. 1-20.
22. KANG, BooJoong, et al. N-opcode analysis for android malware classification and categorization. In: 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security). IEEE, 2016. p. 1-7.
23. YU, Lei; LIU, Huan. Feature selection for high-dimensional data: A fast correlation-based filter solution. In: Proceedings of the 20th international conference on machine learning (ICML-03). 2003. p. 856-863.
24. LI, Jundong, et al. Feature selection: A data perspective. ACM Computing Surveys (CSUR), 2018, 50.6: 94.
25. KONG, Deguang, et al. Multi-label relieff and f-statistic feature selections for im
26. HALL, Mark Andrew. Correlation-based feature selection for machine learning. 1999.
27. AHMADI, Mansour, et al. Novel feature extraction, selection and fusion for effective malware family classification. In: Proceedings of the sixth ACM conference on data and application security and privacy. ACM, 2016. p. 183-194.

28. Microsoft Malware Classification Challenge: 6th place solution. 2015 [online] Dostupné na: https://github.com/sash-ko/kaggle-malware-classification/blob/master/SolutionDescription.pdf
29. Microsoft Malware Classification Challenge third place solution. 2015 [online] Dostupné na: https://github.com/geffy/kaggle-malware/blob/master/description.pdf
30. CHEN, Tianqi; GUESTRIN, Carlos. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016. p. 785-794.
31. FERNÁNDEZ-DELGADO, Manuel, et al. Do we need hundreds of classifiers to solve real world classification problems?. The Journal of Machine Learning Research, 2014, 15.1: 3133-3181.
32. GIBERT, Daniel, et al. Using convolutional neural networks for classification of malware represented as images. Journal of Computer Virology and Hacking Techniques, 2018, 1-14.
33. JUNG, Byungho; KIM, Taeguen; IM, Eul Gyu. Malware classification using byte sequence information. In: Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems. ACM, 2018. p. 143-148.
34. GIBERT, Daniel, et al. Classification of malware by using structural entropy on convolutional neural networks. In: Thirty-Second AAAI Conference on Artificial Intelligence. 2018.
35. CUNNINGHAM, Padraig; DELANY, Sarah Jane. k-Nearest neighbour classifiers. Multiple Classifier Systems, 2007, 34.8: 1-17.
36. RISH, Irina, et al. An empirical study of the naive Bayes classifier. In: IJCAI 2001 workshop on empirical methods in artificial intelligence. 2001. p. 41-46.
37. KOLOSNJAJI, Bojan, et al. Adaptive semantics-aware malware classification. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2016. p. 419-439.
38. MCINNES, Leland; HEALY, John. Accelerated hierarchical density based clustering. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, 2017. p. 33-42.
39. KOLOSNJAJI, Bojan, et al. Deep learning for classification of malware system call sequences. In: Australasian Joint Conference on Artificial Intelligence. Springer, Cham, 2016. p. 137-149.

40. LI, Yuping, et al. Experimental study of fuzzy hashing in malware clustering analysis. In: 8th Workshop on Cyber Security Experimentation and Test ({CSET} 15). 2015.
41. CANZANESE, Raymond; KAM, Moshe; MANCORIDIS, Spiros. Toward an automatic, online behavioral malware classification system. In: 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems. IEEE, 2013. p. 111-120.
42. NATARAJ, Lakshmanan, et al. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence. ACM, 2011. p. 21-30.
43. ZHAO, Hengli, et al. Malicious executables classification based on behavioral factor analysis. In: 2010 International Conference on e-Education, e-Business, e-Management and e-Learning. IEEE, 2010. p. 502-506.
44. KOLOSNJAJI, Bojan, et al. Empowering convolutional networks for malware classification and analysis. In: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017. p. 3838-3845.
45. ANNACHHATRE, Chinmayee; AUSTIN, Thomas H.; STAMP, Mark. Hidden Markov models for malware classification. Journal of Computer Virology and Hacking Techniques, 2015, 11.2: 59-73.
46. SAHU, Kanti; SHRIVASTAVA, S. K. Kernel K-means clustering for phishing website and malware categorization. International Journal of Computer Applications, 2015, 111.9.
47. WOJNOWICZ, Michael, et al. Projecting" better than randomly": How to reduce the dimensionality of very large datasets in a way that outperforms random projections. In: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE, 2016. p. 184-193.
48. MARICONTI, Enrico, et al. Mamadroid: Detecting android malware by building markov chains of behavioral models. arXiv preprint arXiv:1612.04433, 2016.
49. DAHL, George E., et al. Large-scale malware classification using random projections and neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013. p. 3422-3426.
50. Holmes Processing Automated Malware Relationships, preprocessing. 2017 [online] Dostupné na: https://github.com/HolmesProcessing/gsoc_relationship/tree/master/ml/pre-processing

51. MCINNES, Leland; HEALY, John; ASTELS, Steve. hdbscan: Hierarchical density based clustering. The Journal of Open Source Software, 2017, 2.11: 205.
52. KONG, Justin; WEBB, David J.; HAGIWARA, Manabu. Formalization of Insertion/Deletion Codes and the Levenshtein Metric in Lean. In: 2018 International Symposium on Information Theory and Its Applications (ISITA). IEEE, 2018. p. 11-15.
53. DAMERAU, Fred J. A technique for computer detection and correction of spelling errors. Communications of the ACM, 1964, 7.3: 171-176.
54. SAXE, Joshua; BERLIN, Konstantin. Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2015. p. 11-20.
55. ISLAM, Rafiqul, et al. Classification of malware based on string and function feature selection. In: 2010 Second Cybercrime and Trustworthy Computing Workshop. IEEE, 2010. p. 9-17.
56. MASUD, Mohammad M.; KHAN, Latifur; THURAISINGHAM, Bhavani. A scalable multi-level feature extraction technique to detect malicious executables. Information Systems Frontiers, 2008, 10.1: 33-45.
57. LYDA, Robert; HAMROCK, James. Using entropy analysis to find encrypted and packed malware. IEEE Security & Privacy, 2007, 5.2: 40-45.
58. ANDERSON, Hyrum S.; ROTH, Phil. Ember: an open dataset for training static PE malware machine learning models. arXiv preprint arXiv:1804.04637, 2018.
59. TIAN, Ronghua, et al. An automated classification system based on the strings of trojan and virus families. In: 2009 4th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2009. p. 23-30.
60. JANG, Jiyong; BRUMLEY, David; VENKATARAMAN, Shobha. Bitshred: feature hashing malware for scalable triage and semantic analysis. In: Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011. p. 309-320.
61. NARAYANAN, Barath Narayanan; DJANEYE-BOUNDJOU, Ouboti; KEBEDE, Temesguen M. Performance analysis of machine learning and pattern recognition algorithms for malware classification. In: 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS). IEEE, 2016. p. 338-342.
62. CHAPELLE, Olivier. Training a support vector machine in the primal. Neural computation, 2007, 19.5: 1155-1178.

63. CHEN, Chih-Ming; LEE, Hahn-Ming; CHANG, Yu-Jung. Two novel feature selection approaches for web page classification. Expert systems with Applications, 2009, 36.1: 260-272.
64. JOVIĆ, Alan; BRKIĆ, Karla; BOGUNOVIĆ, Nikola. A review of feature selection methods with applications. In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2015. p. 1200-1205.
65. CHANDRASHEKAR, Girish; SAHIN, Ferat. A survey on feature selection methods. Computers & Electrical Engineering, 2014, 40.1: 16-28.
66. TANG, Jiliang; ALELYANI, Salem; LIU, Huan. Feature selection for classification: A review. Data classification: algorithms and applications, 2014, 37.
67. YAMADA, Makoto, et al. Ultra high-dimensional nonlinear feature selection for big biological data. IEEE Transactions on Knowledge and Data Engineering, 2018, 30.7: 1352-1365.
68. YAMADA, Makoto, et al. High-dimensional feature selection by feature-wise kernelized lasso. Neural computation, 2014, 26.1: 185-207.
69. TUV, Eugene; TORKKOLA, Kari. Feature filtering with ensembles using artificial contrasts. In: Proceedings of the SIAM 2005 Int. Workshop on Feature Selection for Data Mining. 2005. p. 69-71.
70. ExifTool. 2020 [online] Dostupné na: https://exiftool.org/
71. LightGBM. 2019 [online] Dostupné na: https://github.com/Microsoft/LightGBM
72. Catboost. 2019 [online] Dostupné na: https://github.com/catboost/catboost
73. Johnson, Rie, and Tong Zhang. "Learning nonlinear functions using regularized greedy forest." IEEE transactions on pattern analysis and machine intelligence 36.5 (2014): 942-954.
74. Hexdump. 2013 [online] Dostupné na: http://man7.org/linux/man-pages/man1/hexdump.1.html
75. Dumpbin. 2019 [online] Dostupné na: https://docs.microsoft.com/en-us/cpp/build/reference/dumpbin-command-line?view=vs-2019
76. Mastiff,. 2015 [online] Dostupné na: https://github.com/KoreLogicSecurity/mastiff
77. IDA. 2015 [online] Dostupné na: https://www.hex-rays.com/products/ida/
78. Objdump. 2019 [online] Dostupné na: http://man7.org/linux/man-pages/man1/objdump.1.html
79. Radare. 2019 [online] Dostupné na: https://rada.re/r/

80. Pescanner. 2015 [online] Dostupné na: https://www.aldeid.com/wiki/Pescanner
81. Pev. 2019 [online] Dostupné na: https://github.com/merces/pev
82. Peframe. 2019 [online] Dostupné na: https://github.com/guelfoweb/peframe
83. Pefile. 2019 [online] Dostupné na: https://github.com/erocarrera/pefile
84. LIEF. 2019 [online] Dostupné na: https://github.com/lief-project/LIEF
85. REMnux. 2016 [online] Dostupné na: https://remnux.org/docs/distro/tools/
86. PEiD. 2013 [online] Dostupné na: https://www.aldeid.com/wiki/PEiD
87. Packerid. 2014 [online] Dostupné na: https://www.aldeid.com/wiki/Packerid
88. ZHAO, Zheng, et al. Advancing feature selection research. ASU feature selection repository, 2010, 1-28.
89. SINGH, Sanasam Ranbir, et al. Feature Selection for Text Classification Based on Gini Coefficient of Inequality. Fsdm, 2010, 10: 76-85.
90. VirusTotal. 2019 [online] Dostupné na: https://www.virustotal.com/gui/home/upload
91. Process Monitor. 2017 [online] Dostupné na: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon
92. BREIMAN, Leo. Random forests. Machine learning, 2001, 45.1: 5-32.
93. DUDA, Richard O.; HART, Peter E.; STORK, David G. Pattern classification. John Wiley & Sons, 2012.
94. HORN, Roger A.; JOHNSON, Charles R. Matrix analysis. Cambridge university press, 2012.
95. KE, Guolin, et al. Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems. 2017. p. 3146-3154.
96. Classification metrics. 2019 [online] Dostupné na: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics
97. RIECK, Konrad, et al. Learning and classification of malware behavior. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Berlin, Heidelberg, 2008. p. 108-125.
98. TSUGE, Satoru, et al. Dimensionality reduction using non-negative matrix factorization for information retrieval. In: 2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat. No. 01CH37236). IEEE, 2001. p. 960-965.

99. ROBERTS, Stephen; CHOUDREY, Rizwan. Bayesian independent component analysis with prior constraints: An application in biosignal analysis. In: International Workshop on Deterministic and Statistical Methods in Machine Learning. Springer, Berlin, Heidelberg, 2004. p. 159-179.

100. POWERS, David Martin. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. 2011.

101. SU, Jiawei, et al. Lightweight classification of IoT malware based on image recognition. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2018. p. 664-669.

102. GIBERT, Daniel; MATEU, Carles; PLANES, Jordi. An End-to-End Deep Learning Architecture for Classification of Malware's Binary Content. In: International Conference on Artificial Neural Networks. Springer, Cham, 2018. p. 383-391.

103. Strings. 2020 [online] Dostupné na: https://github.com/EricZimmerman/bstrings

104. YAKURA, Hiromu, et al. Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy. ACM, 2018. p. 127-134.

105. KOLOSNJAJI, Bojan, et al. Empowering convolutional networks for malware classification and analysis. In: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017. p. 3838-3845.

106. RAHUL, R. K., et al. Deep learning for network flow analysis and malware classification. In: International Symposium on Security in Computing and Communication. Springer, Singapore, 2017. p. 226-235.

107. CHAKRABORTY, Tanmoy; PIERAZZI, Fabio; SUBRAHMANIAN, V. S. EC2: ensemble clustering and classification for predicting android malware families. IEEE Transactions on Dependable and Secure Computing, 2017.

108. YUE, Songqing. Imbalanced malware images classification: a CNN based approach. arXiv preprint arXiv:1708.08042, 2017.

109. KABANGA, Espoir K.; KIM, Chang Hoon. Malware images classification using convolutional neural network. Journal of Computer and Communications, 2017, 6.01: 153.

110. ØSTBYE, Morten Oscar. Multinomial malware classification based on call graphs. 2017. Master's Thesis. NTNU.

111. HASSEN, Mehadi; CHAN, Philip K. Scalable function call graph-based malware classification. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. ACM, 2017. p. 239-248.

112. Sigcheck. 2017 [online] Dostupné na: https://docs.microsoft.com/en-us/sysinternals/downloads/sigcheck

113. Pype32. 2015 [online] Dostupné na: https://github.com/crackinglandia/pype32

114. Strings. 2009 [online] Dostupné na: https://linux.die.net/man/1/strings

115. DREW, Jake; HAHSLER, Michael; MOORE, Tyler. Polymorphic malware detection using sequence classification methods and ensembles. EURASIP Journal on Information Security, 2017, 2017.1: 2.

116. GARCIA, Felan Carlo C.; MUGA, I. I.; FELIX, P. Random forest for malware classification. arXiv preprint arXiv:1609.07770, 2016.

117. HAN, Kyoung Soo, et al. Malware analysis using visualized images and entropy graphs. International Journal of Information Security, 2015, 14.1: 1-14.

118. KANG, BooJoong, et al. PageRank in malware categorization. In: Proceedings of the 2015 Conference on research in adaptive and convergent systems. ACM, 2015. p. 291-295.

119. YANG, Chao, et al. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In: European symposium on research in computer security. Springer, Cham, 2014. p. 163-182.

120. RAFIQUE, M. Zubair, et al. Evolutionary algorithms for classification of malware families through different network behaviors. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, 2014. p. 1167-1174.

121. HAN, KyoungSoo; KANG, BooJoong; IM, Eul Gyu. Malware analysis using visualized image matrices. The Scientific World Journal, 2014, 2014.

122. CHO, In Kyeom, et al. Malware Similarity Analysis using API Sequence Alignments. J. Internet Serv. Inf. Secur., 2014, 4.4: 103-114.

123. ZHANG, Mu, et al. Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. ACM, 2014. p. 1105-1116.

124. GONZALEZ, Lilia E.; VAZQUEZ, Roberto A. Malware classification using Euclidean distance and artificial neural networks. In: 2013 12th Mexican International Conference on Artificial Intelligence. IEEE, 2013. p. 103-108.

125. NARI, Saeed; GHORBANI, Ali A. Automated malware classification based on network behavior. In: 2013 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2013. p. 642-647.

126. RAVI, Saradha; BALAKRISHNAN, N.; VENKATESH, Bharath. Behavior-based Malware analysis using profile hidden Markov models. In: 2013 International Conference on Security and Cryptography (SECRYPT). IEEE, 2013. p. 1-12.

127. Multiscanner. 2019 [online] Dostupné na: https://github.com/mitre/multiscanner

128. LIANGBOONPRAKONG, Chatchai; SORNIL, Ohm. Classification of malware families based on n-grams sequential pattern features. In: 2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA). IEEE, 2013. p. 777-782.

129. KONG, Deguang; YAN, Guanhua. Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013. p. 1357-1365.

130. LEDOUX, Charles; WALENSTEIN, Andrew; LAKHOTIA, Arun. Improved malware classification through sensor fusion using disjoint union. In: International Conference on Information Systems, Technology and Management. Springer, Berlin, Heidelberg, 2012. p. 360-371.

131. ISLAM, Rafiqul; ALTAS, Irfan. A comparative study of malware family classification. In: International Conference on Information and Communications Security. Springer, Berlin, Heidelberg, 2012. p. 488-496.

132. RIECK, Konrad, et al. Automatic analysis of malware behavior using machine learning. Journal of Computer Security, 2011, 19.4: 639-668.

133. MOONSAMY, Veelasha; TIAN, Ronghua; BATTEN, Lynn. Feature reduction to speed up malware classification. In: Nordic Conference on Secure IT Systems. Springer, Berlin, Heidelberg, 2011. p. 176-188.

134. PEKTAŞ, Abdurrahman; ERIŞ, Mehmet; ACARMAN, Tankut. Proposal of n-gram based algorithm for malware classification. In: The Fifth International Conference on Emerging Security Information, Systems and Technologies. 2011. p. 7-13.

135. TIAN, Ronghua, et al. Differentiating malware from cleanware using behavioural analysis. In: 2010 5th international conference on malicious and unwanted software. IEEE, 2010. p. 23-30.

136. PARK, Younghee, et al. Fast malware classification by automated behavioral graph matching. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. ACM, 2010. p. 45.

137. HU, Xin; CHIUEH, Tzi-cker; SHIN, Kang G. Large-scale malware indexing using function-call graphs. In: Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009. p. 611-620.

138. SORNIL, Ohm; LIANGBOONPRAKONG, Chatchai. Malware Classification Using N-grams Sequential Pattern Features. Journal of Information Processing and Management, 2013, 4.5: 59-67.

139. PEKTAŞ, Abdurrahman; ERIŞ, Mehmet; ACARMAN, Tankut. Proposal of n-gram based algorithm for malware classification. In: The Fifth International Conference on Emerging Security Information, Systems and Technologies. 2011. p. 7-13.

140. XU, Dongkuan; TIAN, Yingjie. A comprehensive survey of clustering algorithms. Annals of Data Science, 2015, 2.2: 165-193.

141. CHAN, Tony F.; GOLUB, Gene H.; LEVEQUE, Randall J. Algorithms for computing the sample variance: Analysis and recommendations. The American Statistician, 1983, 37.3: 242-247.

142. HALL, Mark A. Correlation-based feature selection of discrete and numeric class machine learning. 2000.

143. LOWRY, Richard. Concepts and applications of inferential statistics. 2014.

144. Chi2. 2019 [online] Dostupné na: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2

145. SUNEETHA, N.; HARI, V. M. K.; KUMAR, V. Sunil. Modified gini index classification: a case study of heart disease dataset. International Journal on Computer Science and Engineering, 2010, 2.06: 1959-1965.

146. HE, Xiaofei; CAI, Deng; NIYOGI, Partha. Laplacian skóre for feature selection. In: Advances in neural information processing systems. 2006. p. 507-514.

147. NIE, Feiping, et al. Trace ratio criterion for feature selection. In: AAAI. 2008. p. 671-676.

148. ZHAO, Zheng; LIU, Huan. Spectral feature selection for supervised and unsupervised learning. In: Proceedings of the 24th international conference on Machine learning. ACM, 2007. p. 1151-1157.

149. ROBNIK-ŠIKONJA, Marko; KONONENKO, Igor. Theoretical and empirical analysis of ReliefF and RReliefF. Machine learning, 2003, 53.1-2: 23-69.

150. KRASKOV, Alexander; STÖGBAUER, Harald; GRASSBERGER, Peter. Estimating mutual information. Physical review E, 2004, 69.6: 066138.

151. DING, Chris; PENG, Hanchuan. Minimum redundancy feature selection from microarray gene expression data. Journal of bioinformatics and computational biology, 2005, 3.02: 185-205.

152. PENG, Hanchuan; LONG, Fuhui; DING, Chris. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2005, 8: 1226-1238.

153. MEYER, Patrick E.; BONTEMPI, Gianluca. On the use of variable complementarity for feature selection in cancer classification. In: Workshops on applications of evolutionary computation. Springer, Berlin, Heidelberg, 2006. p. 91-102.

154. SAMMUT, Claude; WEBB, Geoffrey I. (ed.). Encyclopedia of machine learning. Springer Science & Business Media, 2011.

155. UniExtract. 2019 [online] Dostupné na: https://github.com/Bioruebe/UniExtract2

156. Chir.B description. 2019 [online] Dostupné na: https://www.virusradar.com/en/Win32_Chir.B/description

157. r2pipe. 2019 [online] Dostupné na: https://github.com/radareorg/radare2-r2pipe

158. Sigcheck. 2019 [online] Dostupné na: https://docs.microsoft.com/en-us/sysinternals/downloads/sigcheck

159. Numpy, genfromtxt. 2019 [online] Dostupné na: https://docs.scipy.org/doc/numpy/reference/generated/numpy.genfromtxt.html

160. Pandas read_csv. 2019 [online] Dostupné na: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

161. Numpy data types. 2019 [online] Dostupné na: https://docs.scipy.org/doc/numpy-1.10.0/user/basics.types.html

162. sklearn StandardScaler. 2019 [online] Dostupné na: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

163. Permutation importance vs MDI 2019 [online] Dostupné na: https://scikit-learn.org/dev/auto_examples/inspection/plot_permutation_importance.html

164. CatBoost feature importance. 2019 [online] Dostupné na: https://catboost.ai/docs/concepts/fstr.html#fstr

165. Pyew. 2019 [online] Dostupné na: https://github.com/joxeankoret/pyew

166. Generic parser. 2018 [online] Dostupné na: https://github.com/uppusaikiran/generic-parser

167. HYDE, Randall. Write Great Code, Vol. 2: Thinking Low-Level, Writing High-Level. No Starch Press, 2004.

168. NumPy. 2020 [online] Dostupné na: https://numpy.org/

169. Pandas. 2020 [online] Dostupné na: https://pandas.pydata.org/

170. scikit-feature. 2018 [online] Dostupné na: https://github.com/jundongl/scikit-feature

171. MONNAPPA, K. A. Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware. Packt Publishing Ltd, 171048.

172. BRANCO, Rodrigo Rubira; BARBOSA, Gabriel Negreira; NETO, Pedro Drimel. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. Black Hat, 17104171, 4.

173. SIKORSKI, Michael; HONIG, Andrew. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press, 17104171.

174. Portable Executable File Format. 2018 [online] Dostupné na : https://blog.kowalczyk.info/articles/pefileformat.html

Prílohy

Príloha A: CD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe, program na selekciu atribútov

Príloha B: Zoznam odborných a vedeckých prác zaoberajúcich sa selekciou atribútov

Príloha C: Nástroje na analýzu programov

Príloha D: Podrobné výsledky z analýz