Klasifikácia malvéru využitím selekcie atribútov – analýza a návrhy riešenia

Bc. Peter Chomič

1Im, 2018 - 2019

Abstrakt. Práca sa venuje multinomiálnou klasifikáciou škodlivého kódu (malvéru). V rámci výskumnej práce si vytvárame vlastný dataset, ktorého cieľom je zachytenie, čo najväčšieho množstva charakteristík vzoriek. Cieľom tejto práce je pomocou algoritmov na selekciu atribútov vylúčiť tie atribúty, ktoré nie sú dôležité pri klasifikácii. Keďže atribútov je enormne veľké množstvo a mnohé metódy sú výpočtovo náročné tak selekciu delíme na hrubú a jemnú. Pri hrubej selekcii používame výpočtovo nenáročné metódy ktoré vylúčia atribúty, ktoré majú minimálny alebo žiadny vplyv na klasifikáciu. Po nej nasleduje jemná selekcia ktorá vylúči irelevantné a redundantné atribúty pomocou výpočtovo náročnejších metód.

Kľúčové slová: klasifikácia, multinomiálna klasifikácia, malvér, selekcia atribútov

1 Úvod

Nový škodlivý kód (malvér) rapídne pribúda. Podľa štúdie [1] urobenej v decembri 2018 McAfee zachytilo v treťom kvartáli 2018 63 miliónov vzoriek nového malvéru, čo je 53% nárast oproti minulému kvartálu. Každý kvartál za posledné dve roky (2017 a 2018) pribudnú rádovo desiatky miliónov vzoriek nového malvéru. Celkové množstvo zachyteného malvéru stúpalo každým kvartálom rokov 2017 a 2018 pričom v treťom kvartáli 2018 to bolo 837 miliónov vzoriek. Antivírusové programy používajú tzv. signatúry získané z malvéru extrakciou vhodných atribútov, ktoré charakterizujú jeho rodinu. Tento prístup má však 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].

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, ako boli získané [3], a to na statické a dynamické.

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 systému a procesom (spusteným programom). Vďaka tomu sa možno vyhnúť problémom spojeným so statickými atribútmi. Spúšťanie vzoriek malvéru je ale časovo náročné a získané údaje majú veľký objem. 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“ [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]. Overfitting nastáva keď sa model naučil tak, že dáva veľmi dobré výsledky na tréningovej vzorke ale utrpela jeho generalizácia – na ostatných vzorkách dáva veľmi zlé výsledky [65].

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:

- **Filtrovacie metódy** – podľa danej metriky ohodnotia všetky atribúty a vyberú tie, ktoré sa nachádzajú nad špecifikovaným prahom. Sú nezávislé od klasifikátora. Metódy sa líšia použitou metrikou. Nevýhoda je, že do úvahy sa berie ohodnotenie len pre jednotlivé atribúty a nie 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 na [24]:
    - založené na štatistickej teórii (napríklad chi-square)
    - založené na teórii informácií (odvodené od entropie)
    - založené na podobnosti (napríklad fisher score)
- **Wrapper metódy** – zo začiatočnej množiny atribútov budujú najlepšiu podmnožinu pomocou klasifikátora. Ohodnotenie podmnožiny je presnosť klasifikátora s jej použitím. Sú omnoho pomalšie, lebo pre každé ohodnotenie je potrebné vykonať klasifikáciu. Pre daný klasifikátor 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 mnohé filtrovacie metódy pri určení atribútu ktorý delí podstrom [7]. Takto sa aj stromy samotné dajú využiť pre selekciu atribútov. Z tohto dôvodu sa algoritmy na tvorbu stromov pomocou filtrovacích 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.

Pred samotnou klasifikáciou je pri tvorbe vlastného datasetu potrebné každej vzorke určiť triedu. Je možné to urobiť manuálne, pomocou antivírusových programov alebo klastrovaním vzoriek a tvorbou vlastných tried. Spôsobov klastrovania je niekoľko. Xu a Tian [140] porovnávajú 71 algoritmov. Okrem výberu algoritmu je dôležité vybrať aj metriku pre vzdialenosť, prípadne podobnosť. V rámci toho istého článku sa uvádza 8 najčastejších metrík. Takto môžu vzniknúť stovky kombinácií klastrovania. 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). Nevedie rozdeliť nekonvexné klastre. Jadrové (Kernel) verzie sa pomocou jadrového (kernel) triku. Ide o transformáciu do viacrozmerného priestoru, kde sa snažia nekonvexné klastre previesť na konvexné. Príkladom sú K-means, K-medoids, K-medians.
- **hierarchické:** vedia nájsť klastre vo všetkých typoch údajov, ale sú výpočtovo náročné. Príkladom sú BIRCH, CURE, ROCK, chameleon.
- **distribution based**: škálovateľné, ale časovo náročné. Príkladom sú DBCLASD, DMM.
- **density based:** efektívne zvládnu všetky tvary klastrov. Na druhej strane potrebujú veľa pamäte. Výsledky môžu byť nepresné, ak nie je rovnomerná hustota vzoriek. Príkladom sú DBSCAN, OPTICS, mean-shift.
- **grid based:** priestor sa rozdelí do mriežky, ktorá sa dá klastrovať paralelne. Sú rýchle, ale nie veľmi presné. Príkladom sú CLIQUE, STING.
- **grafové:** vzorky predstavujú vrcholy grafu a vzťahy medzi nimi predstavujú hrany. Príkladom sú CLICK a MST.
- **swarm intelligence based:** Simulujú meniace sa procesy v biologickej populácii (kolónia mravcov, včiel). Príkladom sú kategórie algoritmov PSO, ACO, SFLA, ABC.
- **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 klastra. Algoritmus je greedy.

Až po aplikácií vyššie popísaných krokov, možno použiť samotné klastrovanie. Týchto metód je tiež veľké množstvo. Fernández-Delgado, et al. v rámci svojho výskumu [31] porovnali 179 klasifikátorov zo 17 rodín na 121 datasetoch. Niektoré triedy použitých klasifikátorov sú [31]:

- support vector machine (SVM) – rozdelí priestor na podroviny.

- neurónové siete.
- bayesovské klasifikátory,
- rozhodovacie stromy (decision trees - DT) – postupne delia dataset podľa hraníc vo vybraných atribútoch. Viac stromov tvorí Random forest (RF).

1.2 Prehľad súčasného stavu

V rámci danej 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).

1.2.1 Selekcia atribútov pre klasifikáciu malvéru

1.2.1.1 Filtrovacie metódy

Pokiaľ je počet atribútov veľmi veľký je lepšie rozdeliť selekciu na niekoľko krokov a na začiatku použiť metódy, ktoré sú výpočtovo menej náročné [5]. Pokiaľ je počet atribútov tak veľký, že je nemožné akokoľvek ich spracovať, je potrebné hneď na začiatku niektoré atribúty vylúčiť.

Príkladom nespracovateľného množstva atribútov je článok [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 1 percenta súborov bola redukcia viac ako 99.9 % z 1.6 milióna položiek. Tento postup sa používa často. Príkladom použitia môže byť napríklad práce [10,11]. 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 200krát 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, kde atribúty ohodnotíme podľa počtu výskytov v jednej vzorke sa označuje ako **term frequency (TF) –** frekvencia výrazov. U nás 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, aby zabezpečil, že zachytí aj atribúty pre triedy, ktoré nemajú veľkú frekvenciu svojich najfrekventovanejších n-gramov a do globálneho rebríčka by sa nedostali. Tian, et al. v článku [59] si rozdelili každú rodinu na dve množiny – tréningovú a testovaciu. Z tréningovej množiny extrahovali reťazce. Následne brali do globálneho setu len tie reťazce, ktoré sa vyskytovali aspoň v 10 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 používaná metóda ohodnotenia je **TF-IDF** (term frequency – inverse document frequency). 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 atribút sa vyskytuje len v malom počte tried (prípadne vzoriek), takže bude pravdepodobne lepšie rozdeľovať dataset.

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

C predstavuje konštantu, bez ktorej veľké množstvo n-gramov dosahovalo maximálne skóre. Premenné „*mj*“, „*bj*“ 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 – generalizobané Fisher skóre [19]. Fisher skóre dáva vysoké hodnotenie atribútom ktoré dosahujú podobné hodnoty vo vzorkách ktoré patria do rovnakej triedy a zároveň rôzne

hodnoty pre vzorky z rôznych tried [66]. Skóre i-tého atribútu S_i sa počíta cez vzorec (7) kde premenné u_{ij} a p_{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.

$$S_i = \frac{\sum n_j (u_{ij} - u_i)^2}{\sum n_j * p_{ij}^2} \tag{7}$$

Častejšie používané ohodnotenie bolo mutual information gain (MIG) – **zisk vzájomnej informácie**, ktorá vyjadruje, aká veľka závislosť je medzi rozdelením jednotlivých atribútov a tried v datasete [66]. Je viacero vzorcov na počítanie MIG. Santos. Et al. [12] použili vzorec (8) popisuje MIG pre vstup X,Y kde X je množina frekvencií výskytu jednotlivých atribútov v datasete a Y je množina tried. Tang, et al. [66] použili vzorec (9) 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 entropie po pozorovaní rozdelenie tried (podmienená entropia).

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log\left(\frac{P(x,y)}{P(x)P(y)}\right) \tag{8}$$

$$IG(f_i, C) = H(f_i) - H(f_i|C) \tag{9}$$

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 filtrovacie metódy, a to **ReliefF**, ktorý počíta pomer vzdialenosti atribútu ku „*k*“ atribútom z vlastnej triedy (vzdialenosť vo vnútri triedy) a inej triedy (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 [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* a u_{js} sa počíta pomocou vzorca (11) kde n_{j*} je počet vzoriek v ktorých atribút nadobúda hodnotu *j* v datasete a n_{*s} predstavuje počet vzoriek v triede *r*.

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

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

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 sšítajú 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.

1.2.1.2 Embedded metódy

Regularizácia bráni tvorbe príliš komplikovaného modelu (klasifikátora) tým, že pri trénovaní pridáva chybu ktorá sa zväčšuje s komplexitou modelu. Regularizácia sa používa preto, lebo príliš komplexný model často značí, že dochádza k overfittingu. L1-regularizácia pridáva chybu ako absolútnu hodnotu rozdielu hodnôt (least absolute deviations - LAD) a L2-regularizácia pridá chybu ako druhú mocninu rozdielu hodnôt (least squares error - LSE). L1-regularizácia sa označuje aj ako LASSO (least absolute shrinkage and selection operator) a L2-regularizácia sa označuje ako Ridge. Aj keď pôvodny účel regularizácie je zabránenie overfittingu dá sa využiť pre selekciu atribútov. LASSO a Ridge menia koeficienty pri korelovaných atribútoch na hodnoty blízke nule, LASSO 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).

1.2.1.3 Wrapper metódy

Metódy sa líšia použitou optimalizáciou na tvorbu najlepšej podmnožiny atribútov, keďže všetky podmnožiny je často nemožné vyskúšať. V článku [27] Ahmadi, et al. používajú greedy metódu forward stepwise feature selection – dopredná krokovaní selekcia atribútov. Metóda sa označuje ako dopredná, lebo 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 z nej odoberá 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). 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.

1.2.1.4 Redukcia dimenzionality

Trofimov v rámci [29] vyskúšal Principal Component Analysis – PCA, Non-negative matrix factorization -NMF a Independent Component Analysis - ICA. NMF dávalo najlepšie výsledky. Wojnowicz, et al. a Dahl, et al. v článkoch [47,49] používajú RPCA – random 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. Naproti tomu, v článku [14] Lin, et al. použili PCA a KPCA – Kernel PCA.

PCA ortogonálne transformuje atribúty do iného, menšieho priestoru pomocu dekompozície matice atribútov v datasete [98] – vzniknú tak nové atribúty ktoré sú od seba nezávislé a sú kombináciou pôvodných – popisujú všetky pôvodné, aj keď ich je menej. Navyše sú zotriedené podľa variancie. Tieto nové atribúty sa nazývajú principal components. 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 [99].

Pojem feature hashing sa používa v situáciách, ak sa pomocou nejakej funkcie prevedú atribúty do menšieho priestoru tým, že slúžia ako vstupy pre danú funkciu a výstupy funkcie sú nové atribúty. Feature hashing bol viackrát použitý, napríklad 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 bytov 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.

1.2.2 Klasifikácia malvéru

Z pohľadu klasifikácie malvéru sme porovnali 55 článkov za posledných 10 rokov [6,8,27,32-34,37,38,42,43,47,55,101-139]. 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 accuracy – 99.98% [8] a 99.77% [27]. XGBoost používa aj regularizáciu (embedded metóda) [62].

Podľa Fernández-Delgado, et al. [31] (porovnanie 179 klasifikátorov) bol najlepší 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. Tiež

druhý bol K-nearest neighbours classification [35] – 16 výskytov, až potom CNN – 12 a nakoniec Naive Bayes [36] – 8, ktorý mal aj globálne najhoršie výsledky.

1.2.3 Metriky klasifikácie malvéru

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 klasifikátoroch. 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 a 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 multinomiálnej 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. 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. 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 presnosť 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 a vyjadruje sa rozdielom TPR 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 je pre klasifikátor zreteľná. Pre jej vyjadrenie je potrebné definovať si precíznosť falošných negatív (FNP) ako podiel FN a všetkých negatívnych vzoriek (FN+TN). Zreteľnosť potom je rozdiel precíznosti a FNP [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].

1.2.4 Označovanie (labeling)

Pri klasifikácii atribútov je potrebné mať vzorky označené triedou. Pokiaľ nie je dataset ručne označený je potrebné pridať označenia automaticky. Roztriediť dataset možno dvomi spôsobmi. Prvé riešenie je analyzovať ho cez antivírusový program a prideliť mu triedu na základe toho, do akej triedy ho zaradil program. Druhá možnosť je klastrovanie datasetu , kde vzniknuté klastre budú predstavovať jednotlivé triedy.

Yan, et al. v článku [6] použili službu VirusTotal [90], ktorá v sebe zahŕňa analýzu pomocou viac ako 40 antivírusových programov (AV). Označenia jednotlivých programov rozdelili do slov a odstránili príliš generické slová. Potom vynechajú 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 kontrolujú výstupy programov (aspoň 4 správne označenia z 5). Za alias sa považuje keď rôzne antivírusové programy označia tú istú triedu iným (často podobným) názvom.

Kolosnjaji, et al. v rámci článkov [37,39,44] robili binárny vektor zo všetkých označení AV. Tieto vektory klasifikujú pomocou DBSCAN [38] cez kosínusovú vzdialenosť (podiel skalárneho súčinu vektorov a súčinu ich veľkostí).

Li, et al. v článku [40] urobili množinu slov z označení AV, odstránili málo informatívne slová (agent, malware). Odstránili vzorky, ktorých označenia naznačovali packing 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.

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 klastrovanie.

Pomocou atribútov získaných zo vzoriek možno robiť aj vlastné rodiny nezávislé na označeniach AV. Annachhatre, et al. v článku [45] získali sekvencie operačného kódu (operation code), ktoré použili ako vstup pre Hidden Markov model. Z takýchto atribútov je možné dosiahnúť pomocou K-means klastrovania 82% presnosť.

Sahu, et al. v článku [46] používajú kernel k-means na frekvenciu inštrukcií. Kernel k-means využíva kernel trick známy pre SVM – premapuje atribúty do viac-rozmerného priestoru, kde už môžu byť separovateľné, keďže k-means zle klastruje lineárne neseparovateľné údaje. Takto je možné získať presnosť až 78%.

2 Klasifikácia malvéru použitím selekcie atribútov

Klasifikáciu malvériu použitím selekcie atribútov je nutné rozdeliť do niekoľkých etáp:

- klastrovanie,
- extrakcia atribútov,
- hrubá selekcia atribútov (výpočtovo najjednoduchšie metódy),
- jemná selekcia (filtrovacie / embedded metódy),
- finálna selekcia (wrapper / embedded metódy) a
- klasifikácia.

Ako prvé vytvoríme čo najväčšiu sadu skupín atribútov, nad ktorými vykonáme postupnú selekciu a určíme, ktoré atribúty sú najdôležitejšie. Cieľom je nájsť najmenšiu podmnožinu atribútov, ktorá ešte dokáže dobre rozdeliť dataset.

Cieľom práce je aj to, či je možné nájsť takúto podmnožinu len pre málodimenziálne 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. V tomto je aj prínos našej práce. Podľa nám dostupnej literatúry, existuje jediná práca [27], ktorá sa sústreďuje na porovnanie skupín atribútov. Ahmadi, et al. v danej práci nevykonali prvotnú selekciu atribútov, Vykonali len selekciu 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, takže niektoré skupiny atribútov neboli schopní získať.

Samozrejme, malvér sa časom mení a o pár rokov by nami vybrané atribúty stratili rozdeľovaciu schopnosť kvôli novému malvéru, ale vybrané skupiny atribútov by pravdepodobne ostali rovnaké. V prípade, že by začala klesať presnosť predikcie, stačilo by vytvoriť nové rodiny klastrovaním, pre vybrané skupiny urobiť zase selekciu jednotlivých atribútov a riešenie by fungovalo naďalej.

2.1 Klastrovanie

Klastrovať sme sa rozhodli pomocou značenia antivírusových programov cez službu VirusTotal. Keďže viaceré AV pri zabalených a obfuskovaných vzorkách používajú generické názvy ako „packed“, „obfuscator“, tak sme sa rozhodli podobne ako v práci [40] takéto vzorky odstrániť. Dôvodom bolo najmä, aby nám nevznikli osobitné triedy pre zabalený a obfuskovaný malvér, ktoré by nemali zmysel. Tieto vzorky odstránime 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 bytov nenulových. Z toho dôvodu reálna entropia bude ešte nižšia. Preto sme sa rozhodli vylúčiť vzorky s entropiou väčšou alebo rovnou ako 6. 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).

Keďže nevieme určiť málo informatívne slová na odstránenie ani dôležité slová na extrakciu, nemôžeme tak určiť triedu spôsobom podobným tým, ktoré sme popísali v sekcii labeling. Navyše nechceme, aby naše riešenie bolo závislé od ručne tvoreného zoznamu, ktorý sa časom mení a obmedzoval by neskoršie využitie nášho riešenia. Nami navrhované riešenie spočíva v spojenia označenia rôznych antivírusových programov do jedného dlhého slova, a klastrovať tieto slová. Malvér ktorý veľa AV označí podobne budú patriť do jednej triedy. Výhoda je, že slová nemusia byť rovnaké ako v spomenutých riešeniach, stačí ak sú podobné. Za metriku si vyberáme Levenshtein distance [52], prípadne Damerau-Levenshtein distance [53], keďže sú symetrické a spĺňajú trojuholníkovú nerovnosť.

Iná možnosť by bola za triedu považovať najčastejšie sa vyskytujúce sa slovo. Ak ale 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 ale často boli veľmi podobné (niekedy len rozdiel v jednom písmene). Toto ukazuje na to, že klastrovať podľa podobnosti slov je lepšie riešenie, ako len brať do úvahy najčastejšie slovo.

Ako algoritmus pre klastrovanie sme si vybrali HDBSCAN [51]. Ide o pomerne nový (2013) algoritmus, ktorý spája výhody hierarchického a density-based prístupu a je rýchlejší ako podobný algoritmus OPTICS.

HDBSCAN na rozdiel od DBSCAN nepoužíva parameter pre minimálnu vzdialenosť bodov, aby boli v klastri, ale namiesto toho len minimálnu veľkosť klastra. Takto odstraňujú nevýhodu density-based algoritmov, ktoré majú problém, ak klastre majú rôznu hustotu. Teraz už vedia správne klastrovať aj rôzne husté klastre [51].

Po samotnom klastrovaní vytvoríme z triedenej sady dataset, ktorý bude mať približne 1000 vzoriek a každá trieda v ňom bude rovnomerne zastúpená podľa možností. Máme mnohonásobne väčšie množstvo vzoriek, ale kvôli obmedzenej výpočtovej kapacite (nepoužívame klastre počítačov) sa musíme vo veľkosti datasetu obmedziť.

2.2 Extrakcia atribútov

Snažili sme sa zahrnúť čo najviac 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 portable executable – PE formáte, tak sme mohli zahrnúť aj atribúty špecifické pre tento formát. 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 3-gramy (teda berieme len bigramy a trigramy) kvôli pamäťovým obmedzeniam, aj keď často sa používali aj 4-gramy a niektoré práce zašli až po 10-gramy (napríklad [22,29]). Efektívne sú všetky n-gramy pre n>=2 a 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 bytov to neplatí [22]. Aj samotné majú rozlišovaciu schopnosť medzi malvérom a benignwarom (neškodný softvér). Odteraz myslíme pod extrakciou n-gramu získanie oboch verzií.

Atribúty možno rozdeliť do niekoľkých skupín podľa pôvodu:
Statické údaje
Dynamické údaje
Zdrojmi statických údajov sú najčastejšie:
hexadecimálne binárne súbory,
disasemblované súbory a
samotná vzorka.

2.2.1 Statické údaje

Z hexadecimálneho binárneho súboru možno získať n-gramy, pričom 1-gram predstavuje jeden bajt, čo sú pri hexadecimálnej reprezentácii dve znaky [13]. Ďalej je možné zaznamenať frekvencie výskytov každého bajtu. V normalizovanej verzii sú frekvencie vydelené veľkosťou súboru. Zaznamenáva sa aj samotná veľkosť.

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 open source programov. Prikladmi sú: pescanner [80], pev [81] a peframe [82]. Viacero takýchto programov sa vyskytuje v Linux distribúcii REMnux [85]. Existujú aj knižnice pre python – pefile [83], ktorú používa viacero programov na tískanie údajov zo vzoriek. a LIEF [84], ktorú extenzívne Anderson, et al. využili v článku [58].

V disasemblovanej vzorke n-gramy predstavujú postupnosť assembly inštrukcií (opcode). Keďže inštrukcie môžu mať parametre ktoré sa môžu odkazovať na miesta v súbore a tak budú pre každý súbor unikátne tak parametre sa nezahrňajú. Existuje možnosť ako uchovať aspoň informáciu o type parametru – samotný parameter sa nahradí jeho typom – register, pamäť alebo konštanta. Takto sa počet možných atribútov zvýši len 27 násobne (3^3). Registrov nie je až tak veľa a preto môže ako atribút slúžiť frekvencia použitia registrov [27]. Ďalšie atribúty ktoré pomôžu zachytiť viac informácie o parametroch sú frekvencia dĺžky riadkov a priemerná dĺžka riadkov. Ak chceme zistiť koľko miesta si vzorka reálne rezervuje môžeme použiť 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 si miesto.

V disasemblovanej vzorke sa vyskytujú inštrukcie aj v hexadecimálnom tvare aj s parametrami. Na týchto inštrukciách sa tiež dajú robiť n-gramy aj frekvencie výskytu. Použitím n-gramov v assembly aj v hexadecimálnom tvare sa podarí zachytiť aj všeobecné aj podrobné informácie. V hexadecimálnom tvare sú aj zdroje v resource sekcii (rdata). Z týchto možno získať rovnaké atribúty ako z hexadecimálnych inštrukcií. Ak sa na získanie disasemblovanej vzorky použije IDA, tak sa dá ako atribút použiť frekvencia dĺžok funkcií, keďže IDA uchováva aj frekvencie použité vo vzorke [3]. Poslednými atribútmi sú veľkosť súboru a pomer jeho veľkosti s veľkosťou pôvodného (hexadecimálna reprezentácia) súboru.

Zo samotnej vzorky (spustiteľného súboru) je možné získať veľké množstvo atribútov. Programy (aj malvér) využívajú pri práci funkcie z knižníc operačného systému. Zo vzorky možno zistiť aké funkcie importuje aj exportuje. Z tohto údaju možno vytvoriť atribúty pre počty volaných funkcií pre jednotlivé DLL a pre výskyt jednotlivých funkcií vo vzorke pre importované aj exportované funkcie. Ďalšie dve atribúty vychádzajú zo samotnej štruktúry súboru – entropia bajtov a histogram entropie bajtov [54]. Konkrétne Saxe a Berlin v [54] mali posuvné okno po 1024 bytes s posunom po 256 bytes (skokový 1024-gram) na ktorom počítali entropiu a frekvenciu jednotlivých bytov. Potom urobili histogram rozdelený na oboch osiach na 16 hodnôt a os x zachytávala entropiu a os y počet jednotlivých bytov, 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 bytov a už 512 bolo podľa nich veľa lebo malé okná nízkej entropie ovplyvňovali celú entropiu. Ahmadi, et al. [27] ale použili veľkosť okna 10000 bytov pre hexadecimálnu reprezentáciu. Preto skúsime viac veľkostí okien. Histogram aj entropiu bajtov je možné rátať aj na hexadecimálnej reprezentácii.

Ďalej 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 PE súboru:

- metadáta z hlavičiek sekcií a hlavičky súboru (napr. timestamp, autor)
- veľkosť súboru
- virtuálna veľkosť súboru
- entropia celého súboru

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 každej sekcie) berieme do úvahy len známe sekcie. Aby sme zachovali aspoň nejaké informácie o neznámych sekciách pridáme atribúty ktoré zachytávajú neznáme sekcie ako celok (napríklad ich počet). Atribúty viažúce sa k sekciám sú nasledovné:

- počet sekcií
- počet známych a počet neznámych sekcií
- entropia známych sekcií
- 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 priečinkov (data directory) resource sekcie. Ak zistíme, že existuje malý počet známych priečinkov (ako pri sekciách) tak zaznamenáme aj ich veľkosti.

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. Tiam, et al. a Islam, et al. v [59,3] berú reťazce z IDA ktorá defaultne extrahuje tiež len reťazce s dĺžkou aspoň 5. 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. Pri reťazcoch je dobré zamerať sa okrem všeobecných atribútov aj na atribúty zachytávajúce počet špecifických reťazcov ktoré bližšie popisujú činnosť vzorky. Medzi reťazce ktorým chceme venovať zvýšenú pozornosť patria tie, ktoré obsahujú umiestnenie („C:\"), webové stránky ("http"), registre ("HKEY_") a reťazec „MZ". Reťazec „MZ" sa vyskytuje na začiatku každého PE súboru, a ak je ich vo vzorke viac, môže to indikovať že sú v nej zabalené ďalšie spustiteľné programy [58]. Po získaní reťazcov z nich vyextrahujeme nasledujúce všeobecné atribúty:

- n-gramy cez znaky a slová
- frekvencie dĺžky reťazcov
- priemerná dĺžka reťazcov
- frekvencia jednotlivých slov
- frekvencia použitých znakov [58]
- entropia celého zoznamu reťazcov
- histogram entropie znakov [58]

2.2.2 Dynamické údaje

Pre získanie dynamických údajov používame Cuckoo Sandbox [91]. Jeho report obsahuje množstvo informácií, ktoré sa vyskytovali pri dynamických údajoch (API calls, DLLs) s výhodou toho, že ich získava zo spustenej vzorky. Týmto sa vyhne rôznym obfuskačným praktikám. Okrem dynamických údajov Cuckoo zaznamenáva aj množstvo statických údajov (sekcie, ich veľkosť, entropia, zdroje (resources), importy, reťazce). Dokáže aj kontaktovať VirusTotal a pridať jeho hlásenie do výstupu.

Cuckoo zachytáva sieťovú komunikáciu ktorú poskytne v pcap súbore. Komunikáciu monitoruje cez IDS (intrusion detection system) Snort [92] alebo Suricata [94] a vo výstupe zaznamenáva jeho výstrahy. Zachytáva tiež dump pamäte počas vykonávania súboru. Okrem toho vie urobiť dump pamäte pre jednotlivé procesy.

Používa aj signatúry, konkrétne Yara pravidlá. Yara [93] je nástroj ktorý umožňuje definovať množstvo rôznych pravidiel pre identifikáciu malvéru. Vzorka sa potom skontroluje a ak spĺňa nejaké pravidlo tak je vo výstupe označená. Pravidlo sa môže týkať napríklad toho, či vzorka zisťuje že je debugovaná alebo sa snaží zistiť či je spustená vo virtuálnom stroji. Medzi služby ktoré Cuckoo využíva patrí aj IRMA [95] ktorá slúži na analýzu malvéru.

V logu sa zachytávajú vytvorené procesy, súbory s ktorými sa manipulovalo, a chybové hlášky. Vykonáva sa aj behaviorálna analýza ktorá okrem procesov a súborov monitoruje aj volania Win32 API s použitými registrami, používané služby (services) a synchronizáciu (čas, mutexy).

2.3 Hrubá selekcia

Cieľom tejto selekcie je zredukovať počet atribútov pod hranicu ktorú predstavuje veľkosť datasetu, v našom prípade 1000.

Ešte pred samotnou selekciou, už pri extrakcii atribútov možno robiť redukciu dimenzionality. Ak sa ukáže že niektoré atribúty naberajú príliš veľa hodnôt (napr. function length frequency), 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.

Prvým krokom bude 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 budeme brať 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 vylúčime atribúty, ktoré sa v globálnom datasete vyskytujú v počte vzoriek menšom ako je jedna desatina veľkosti najmenšej triedy.

Stále nám ostane pomerne vysoký počet atribútov, ktoré sú roztrúsene vo všetkých triedach, prípadne väčšine tried v malom množstve. Z tohto dôvodu nemajú veľkú rozhodovaciu silu. Preto urobíme osobitne pre každú triedu DF a odstránime tie atribúty, ktoré sú v malom množstve vzoriek vo všetkých triedach. Súčet ich výskytov musí byť pod istou hranicou a zároveň v žiadnej triede nie je výskyt nad inou určenou hranicou.

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 pri klasifikácii byť využitá.

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 v takmer úplnej väčšine datasetu tak ho nemôže dobre rozdeliť. Tento fakt využíva selekcia podľa nízkej variancie (low variance selection). 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 variancie určíme empiricky.

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.

2.4 Jemná selekcia

Pri danej etape by sme mali mať rádovo menej atribútov ako na začiatku. Na tomto mieste budú brané v úvahu aj hodnoty atribútov. Rozhodli sme sa vôbec nepoužívať wrapper metódy. Dôvodom je, že očakávame časovo náročné trénovanie. Pri výskume [8] trvalo trénovanie jeden deň, a aj po použití optimalizačných techník by sme museli vykonať minimálne desiatky trénovaní. Tiež nepovažujeme presnosť klasifikátora za dobré hodnotenie množiny atribútov. Hodnota atribútu je viazaná na klasifikátor

a môže sa dosť zmeniť pri zmene parametrov alebo funkcií klasifikátora (napríklad kernelu pri SVM), a preto nemá až takú veľkú výpovednú hodnotu pre ostatné klasifikátory. Rovnaký problém majú aj embedded regularizačné metódy [24]. Tieto metódy ale majú iné výhody, ktorým sa budeme venovať bližšie v ďalšom texte.

V rámci etapy si najprv rozdelíme atribúty na dve veľké skupiny podľa toho, či pochádzajú z mnohodimenziálnych skupín atribútov alebo nie. Od tejto chvíle budeme pracovať na dvoch datasetoch atribútov – v prvom budú všetky a v druhom budú len málodimenziálne. Toto robíme kvôli tomu, lebo v nasledujúcich krokoch budeme odstraňovať redundantné atribúty a mohli by sme odstrániť málodimenziálne atribúty, ktoré sú globálne redundantné oproti mnohodimenzionálnym ale vo vlastnej skupine nie sú. Všetky nasledujúce kroky vykonáme pre obe skupiny, ktoré na konci porovnáme.

V rámci danej etapy súčasne porovnáme niekoľko algoritmov na selekciu atribútov. Potrebujeme algoritmus, ktorý vylúči redundantné aj irelevantné atribúty. Medzi tieto algoritmy môžeme zaradiť [70,24]:

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

CFS berie do úvahy koreláciu medzi atribútom a triedou pre nájdenie irelevantných atribútov a medzi dvojicami atribútov pre nájdenie redundantných. Ako koreláciu používa symetrickú neistotu [24], ktorá nezvýhodňuje atribúty s vyšším množstvom možných hodnôt ako informačný zisk. Na začiatku si vypočíta korelácie pre každý jeden atribút, a od najlepšieho atribútu postupne buduje najlepšiu množinu. CFS je popísané vo vzorci (13). *S* je k-prvkovká podmnožina atribútov, r_{cf} je priemerná korelácia medzi triedami a atribútmi, a r_{ff} je priemerná korelácia medzi atribútmi navzájom. DISR je popísaný vo vzorci (15).

$$CFS(S) = \frac{k * r_{cf}}{\sqrt{k - k(k-1)r_{ff}}} \tag{13}$$

$$SU(X_S, Y) = 2\frac{I(X_S, Y)}{H(X_S) + H(Y)} \tag{14}$$

$$DISR(k) = \sum_{j \in S} \frac{I(jk, Y)}{H(jkY)} \tag{15}$$

Na podobnom princípe (korelácia medzi atribútmi a triedami aj korelácia medzi atribútmi navzájom) pracuje FCBF. Najprv si algoritmus vyberie množinu atribútov

S ktorá je vysoko korelovaná s rozdelením tried a táto korelácia je nad stanoveným prahom. Ako koreláciu používa symentrickú neistotu medzi atribútmi X_s a triedami *Y* ktorú počíta pomocou vzorca (14) kde *H* je entropia a *I* je informačný zisk. Atribút *k* je potom nazvaný dominantný ak *SU(k,Y)* je väčšie ako prah a zároveň pre žiaden iný, už vybraný atribút *j* neplatí že $SU(x,j) \geq SU(x,Y)$. Tie atribúty pre ktorá táto nerovnosť platí sú označené ako redundantné vzhľadom na *x*. Redundantné atribúty sa potom rozdelia na tie, pre ktoré $SU(j,Y) > SU(k,Y)$ a na tie, pre ktoré platí opačná nerovnosť. Z oboch skupín sa potom heuristicky odstránia niektoré atribúty. Tieto dve algoritmy (CFS a FCBF) ako jediné vyberajú podmnožinu najlepších atribútov na rozdiel od hodnotenia jednotlivých atribútov ako ostatné [24]. CFS a FCBF použijeme ako naše filtrovacie algoritmy. Ostatné algoritmy tiež pracujú na princípe korelácie medzi atribútmi a triedami aj atribútmi navzájom. MIFS pre nový atribút *k* ktorý chceme pridať medzi vybrané atribúty *S* je popísane vo vzorci (15). MRMR len špecifikuje β z CIFS ako $\frac{1}{|S|}$. CIFE má β rovné jednotke, a navyše pripočíta podmienený informačný zisk medzi atribútmi za predpokladu rozdelenia tried *Y*. JMI počíta len podmienený informačný zisk. CMIM, popísané vo vzorci (16) – počíta sa minimum podmieneného informačného zisku medzi novým atribútom a rozdelením tried za predpokladu už vybraných atribútov. Potom sa berie atribút ktorá má toto minimum najvyššie, teda má silnú koreláciu s triedami a zároveň nie je redundantný vzhľadom na už vybrané atribúty.

$$MIFS(k) = I(k,Y) - \beta \sum_{j \in S} I(k,j) \tag{15}$$

$$CMIM(k) = \min_{j \in S}(I(k,Y|j)) \tag{16}$$

Tieto metódy nájdu koreláciu medzi dvojicami (pairwise correlation), ale vzťahy môžu existovať aj medzi viacerými atribútmi a nemusia ani byť lineárne. Takto môže byť jeden atribút kombináciou viacerých a klasifikátoru môže postačovať len ten jeden. Na nájdenie týchto vzťahov musíme použiť iné metódy. Na rozdiel od filtrovacích metód ensemble DT metódy vedia zachytiť aj interakcie medzi viacerými atribútmi pri svojom použití filtrovacích metód [69]. Z tohto dôvodu ich budeme používať.. Ich problémom je, že neodstraňujú redundantné atribúty. Z tohto dôvodu je dobré použiť ich až po filtrovacích metódach. 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 (napríklad [67,68]). Z toho dôvodu ho zahrnieme medzi metódy, ktoré budeme testovať.

Aj stromy môžu priamo používať regularizáciu. Ako napríklad môžeme uviesť XGBoost, ktorý je ensemble stromov [30]. Takisto ju používa aj Regularized Greedy Forest (RGF) [73]. Aby sme vyskúšali embedded regularizáciu aj pre iný klasifikátor použijeme aj regularizovaný SVM (Linear SVC, SGD classifier) pre výber atribútov.

2.5 Klasifikácia

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 použiť ich ako naše klasifikátory. Pri DT chceme použiť aj ensemble založený na gradient boostingu – XGBoost, keďže mal najlepšie skóre z porovnávaných klasifikátorov. Okrem XGBoost skúsime aj dve novšie gradient boosting algoritmy, ktoré tiež dosahujú veľmi dobré výsledky: LGBM [71] a Catboost [72].

SVM rozdeľuje rovinu na podroviny, takže my použijeme jeho verziu pre viactriednuklasifikáciu – SVC (support vector classifier) a jeho výpočtovo menej náročnú lineárnu verziu (Linear SVC, SGD classifier). Pri SVC možno použiť viacero jadier (kernelov) – lineárny, polynomiálny, RBF (radial basis function), sigmoidálny. Nelineárne vedia rozdeliť aj lineárne neseparovateľné údaje.

Pri trénovaní pre všetky klasifikátory použijeme 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. 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. Ako metriku použijeme ROC-AUC – obsah plochy pod ROC krivkou. Najprv výsledok trénovania ktorý rozdelíme na sériu binárnych klasifikácií na ktorých urobíme ROC krivku. Potom na všetkých krivkách vypočítame AUC a z nich urobíme priemer.

3 Záver

V rámci práce sa zameráme na analýzu rôznych prístupov k extrakcii atribútov, redukcii ich dimenzionality, selekcii atribútov a samotnej klasifikácii. Ďalej budeme študovať rôzne možnosti, ako označiť dataset malvéru triedami a v súvislosti s tým aj použitie rôznych prístupov ku klastrovaniu. S ohľadom na tieto údaje si vyberieme vhodnú metódu klastrovania. V ďalšom kroku sa zameráme na extrakciu atribútov pričom sa budeme snažiť pokryť čo najväčšie množstvo atribútov použitých v literatúre. Následne si určíme spôsob selekcie atribútov, ktorý sa bude skladať z niekoľkých etáp naviazaných na seba, pričom neskoršie etapy budú výpočtovo náročnejšie ale vykonajú sa len na atribútoch vybraných predošlou etapou. V poslednej etape vyskúšame niekoľko prístupov selekcie atribútov na rôznych klasifikátoroch, ktoré porovnáme. Vyberieme si najpoužívanejšie klasifikátory, ktoré dosahovali najlepšie výsledky. Tieto klasifikátory tiež porovnáme na všetkých skupinách atribútov získaných v poslednej etape. Nakoniec budeme analyzovať zmenu presnosti pre tieto skupiny atribútov, ak z nich odstránime atribúty pochádzajúce z mnohodimenziálnych zdrojov.

Poďakovanie. Týmto sa chcem poďakovať JUDr. RNDr. Pavlovi Sokolovi, PhD. a Mgr. Ladislavovi Bačovi za cenné rady, pomoc a odborné vedenie pri príprave článku.

Literatúra

1. McAfee LabsThreats Report in December 2018. 2018 [online] Dostupné na: https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-december-2018.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. YAN, Jinpei; QI, Yong; RAO, Qifan. Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks, 2018, 2018.
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 score 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. ESTER, Martin, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd. 1996. p. 226-231.
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. 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.
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. DURGA, A. G.; PRIYA, A. G. Feature Subset Selection Algorithm for High Dimensional Data using Fast Clustering Method. International Journal of Computing and Technology, 2014, 1.2: 240-242.
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. 2009 [online] Dostupné na: https://linux.die.net/man/1/objdump
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. Cuckoo Sandbox. 2019 [online] Dostupné na: https://cuckoosandbox.org/
92. Snort. 2019 [online] Dostupné na: https://www.snort.org/
93. Yara. 2015 [online] Dostupné na: https://virustotal.github.io/yara/

94. Suricata. 2019 [online] Dostupné na: https://suricata-ids.org/
95. IRMA. 2018 [online] Dostupné na: https://irma.quarkslab.com/
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. YAN, Jinpei; QI, Yong; RAO, Qifan. Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks, 2018, 2018.
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.
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. GROSSE, Kathrin, et al. Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435, 2016.
113. 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.
114. 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.

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. 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.
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. 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.
139. 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.
140. XU, Dongkuan; TIAN, Yingjie. A comprehensive survey of clustering algorithms. Annals of Data Science, 2015, 2.2: 165-193.