|  |  |
| --- | --- |
| Univerzita Pavla Jozefa šafárika v košiciach  Prírodovedecká fakulta Názov fakultyNázov vysokej školy | |
| Klasifikácia malvéru využitím selekcie atribútov | |
|  | |
|  | |
| 2019 | Bc. Peter Chomič |

|  |  |
| --- | --- |
| Univerzita Pavla Jozefa šafárika v košiciach  Prírodovedecká fakulta | |
| Klasifikácia malvéru využitím selekcie atribútov | |
| **Bakalárska práca, Diplomová práca, Dizertačná práca, Habilitačná práca**Diplomová práca | |
| Študijný program: | ŠpecializáciaInformatika |
| Pracovisko (katedra/ústav): | Názov katedryÚstav informatiky |
| Vedúci diplomovej práce: | JUDr. RNDr. Pavol Sokol, PhD. |
| Konzultant diplomovej práce: (nepovinný) | Titul Meno Priezvisko, HodnosťMgr. Ladislav Bačo |
|  |  |
| MestoKošice 20092019 | Bc. Peter Chomič |

**Zadanie záverečnej práce**

Zadanie záverečnej práce (ďalej len „zadanie“) je dokument, ktorým vysoká škola stanoví študentovi študijné povinnosti v súvislosti s vypracovaním záverečnej práce. Zadanie spravidla obsahuje: typ záverečnej práce, názov záverečnej práce, meno, priezvisko a tituly študenta, meno, priezvisko a tituly školiteľa, v prípade externého školiteľa meno, priezvisko a tituly konzultanta, školiace pracovisko, meno, priezvisko a tituly vedúceho pracoviska, anotáciu záverečnej práce, jazyk, v ktorom sa práca vypracuje, dátum schválenia zadania.

**Poďakovanie (nepovinné)**

Na tomto mieste môže byť vyjadrenie poďakovania napr. vedúcemu práce resp. konzultantom za pripomienky a odbornú pomoc pri vypracovaní práce. Nie je zvykom ďakovať za rutinnú kontrolu, menšiu spoluprácu alebo všeobecné rady. Vyjadrenie poďakovania v prípade využitia inej práce sa uskutočňuje formou citácie na konci hlavného textu práce a odkazy na citáciu sa musia uviesť aj na zodpovedajúcich miestach v texte.

|  |
| --- |
| Abstrakt v štátnom jazyku |
| V práci sa venujeme 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**: malvér, selekcia atribútov, klasifikácia

|  |
| --- |
| Abstrakt v cudzom jazyku |
| This thesis deals with multinomial classification of malicious software (malware). We create our own dataset with goal of capturing as much traits of samples as possible. Our goal is to remove unimportant features with algorithms for feature selection. 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 none importance for classification in gross selection. After that comes fine selection to remove irrelevant and redundant features with computationally more expensive methods.  **Key words:** malware, feature selection, classification |

Obsah

[Obsah 6](#_Toc24731223)

[Úvod 8](#_Toc24731224)

[1 Prehľad súčasného stavu 13](#_Toc24731225)

[1.1 Selekcia atribútov pre klasifikáciu malvéru 13](#_Toc24731226)

[1.1.1 Filtrovacie metódy 13](#_Toc24731227)

[1.1.2 Embedded metódy 17](#_Toc24731228)

[1.1.3 Wrapper metódy 17](#_Toc24731229)

[1.1.4 Redukcia dimenzionality 18](#_Toc24731230)

[1.2 Klasifikácia malvéru 19](#_Toc24731231)

[1.2.1 Metriky klasifikácie 19](#_Toc24731232)

[1.3 Označovanie (labeling) 21](#_Toc24731233)

[2 Metodológia 23](#_Toc24731234)

[2.1 Labeling 23](#_Toc24731235)

[2.2 Extrakcia atribútov 26](#_Toc24731236)

[2.2.1 Statické atribúty 26](#_Toc24731237)

[2.2.2 Dynamické atribúty 29](#_Toc24731238)

[2.3 Hrubá selekcia 30](#_Toc24731239)

[2.4 Jemná selekcia 31](#_Toc24731240)

[2.4.1 Rýchle metódy 32](#_Toc24731241)

[2.4.2 Pomalé metódy 36](#_Toc24731242)

[2.5 Klasifikácia a embedded metódy 39](#_Toc24731243)

[3 Implementácia a postup 48](#_Toc24731244)

[3.1 Tvorba datasetu 48](#_Toc24731245)

[3.1.1 Labeling 49](#_Toc24731246)

[3.1.2 Extrakcia dát 52](#_Toc24731247)

[3.2 Extrakcia atribútov 53](#_Toc24731248)

[3.2.1 Prvý dataset 54](#_Toc24731249)

[3.2.2 Druhý a tretí dataset 56](#_Toc24731250)

[3.3 Selekcia atribútov 56](#_Toc24731251)

[3.3.1 Prvotná selekcia a predspracovanie 56](#_Toc24731252)

[3.3.2 Výber metód 59](#_Toc24731253)

[3.4 Klasifikácia a výsledky 63](#_Toc24731254)

[3.4.1 Výber metód 64](#_Toc24731255)

[4 Analýza výsledkov 66](#_Toc24731256)

[4.1 Prvý dataset 66](#_Toc24731257)

[4.2 Druhý dataset 66](#_Toc24731258)

[4.3 Tretí dataset 66](#_Toc24731259)

[4.4 Globálne pozorovania 66](#_Toc24731260)

[Záver 67](#_Toc24731261)

[Zoznam použitej literatúry 68](#_Toc24731262)

[Prílohy 81](#_Toc24731263)

[Príloha B – práce o klasifikácii malvéru 82](#_Toc24731264)

[Príloha C – nástroje na analýzu programov 86](#_Toc24731265)

[Príloha D – výsledky prvého datasetu 87](#_Toc24731266)

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 mnoho. 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). Nevedia rozdeliť nekonvexné oblasti. Jadrové (Kernel) verzie to dokážu 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.
* 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.
* grafové: vzorky predstavujú vrcholy grafu a vzťahy medzi nimi predstavujú hrany. Príkladom sú CLICK a MST
* grid based: priestor sa rozdelí do mriežky, ktorá sa dá klastrovať paralelne. Sú rýchle, ale nie veľmi presné. Príkladom sú CLIQUE, STING.
* 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 samotnú klasifikáciu. 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 nadroviny.
* neurónové siete
* bayesovské klasifikátory
* rozhodovacie stromy (decision trees - DT) – postupne delia dataset podľa hraníc vo vybraných atribútoch. Viac stromov tvorí Random forest (RF)

Našim cieľom je ukázať, že selekcia atribútov má pri klasifikácii malvéru veľký dopad na výsledky. Malvér ako zdroj datasetu na klasifikáciu je špecifický tým, že zo vzoriek je možné získať enormné množstvo atribútov (aj rádovo viac ako miliarda) a preto je nejakým spôsobom vždy nutné obmedziť reálne použité atribúty. Preto si vytvárame dataset ktorý pokrýva množstvo skupín atribútov a na ňom používame algoritmy na selekciu z viacerých skupín, pričom sa zameriavame na ich výpočtovú a pamäťovú zložitosť, ako aj výsledky ktoré nimi selektované atribúty dosiahli. Okrem toho selekciu a následnú klasifikáciu robíme aj na skupinách atribútov ktoré majú málo položiek (v prvom datasete menej ako tisíc, v druhom menej ako sto). Vďaka tomu chceme zistiť, či sú pre klasifikáciu postačujúce len tieto atribúty. Ak by boli, tak by nebolo potrebné veľké skupiny atribútov ani extrahovať, čo by ušetrilo veľké množstvo času a pamäte pri klasifikácii malvéru a umožnilo ju spúšťať aj na zariadeniach s obmedzenými zdrojmi. Okrem toho porovnávame presnosť klasifikácie pri statických atribútoch pre dataset s čistými vzorkami a dataset v ktorom prevládajú zabalené a obfuskované vzorky. Cieľom je zistiť, či má ešte zmysel používať len statické atribúty.

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. 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 mnoho algoritmov na viacerých datasetoch a viacerých prahoch selekcie pre viaceré skupiny klasifikačných algoritmov. Okrem výsledkov klasifikácií empiricky porovnávame aj časy behu algoritmov a pre najrýchlejšie aj použitú pamäť. Z výsledkov potom vyvodzujeme závery.

Prehľad súčasného stavu

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).

Selekcia atribútov pre klasifikáciu malvéru

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

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 pri extrakcii 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).

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).

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].

predstavuje konštantu, bez ktorej veľké množstvo n-gramov dosahovalo maximálne skóre. Premenné a 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 ktoré patria 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 a sú priemer a variancia hodnôt *i*-tého atribútu v *j*-tej triede, je počet vzoriek v *j*-tej triede a je celkový priemer *i*-tého atribútu.

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 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 , *C* kde 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). 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].

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 ostatných tried (vzdialenosť von z triedy) a F1-statistics [25]. F1-statistics je popísaná vo vzorci (10) kde *K* je počet tried, je priemer hodnôt daného atribútu v celom datasete, je počet vzoriek v triede *k* a a sú priemerná a štandardná odchýlka atribútu v triede k.

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 ktorý nadobúda *r* rôznych hodnôt sa skóre počíta pomocou vzorca (12) kde je počet vzoriek v ktorých atribút nadobúda hodnotu *j* v triede *s* a sa počíta pomocou vzorca (11) kde je počet vzoriek v ktorých atribút nadobúda hodnotu *j* v celom datasete a predstavuje počet vzoriek v triede *s*.

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.

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. Aj keď pôvodny 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.

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 – 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).

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 pomocou 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.

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 – 99.83% [8] a 99.77% [27]. XGBoost používa aj regularizáciu (embedded metóda) [30].

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. 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. //TODO – zmeň v prílohe články na príslušné čísla zdrojov.

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 klasifikátoroch, 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 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].

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 dosiahnuť 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 nedokáže rozdeliť lineárne neseparovateľný dataset. Takto získali presnosť až 78%.

Metodológia

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

* Labeling
* Extrakcia atribútov
* Hrubá selekcia atribútov (výpočtovo nenáročné metódy)
* Jemná selekcia (filtrovacie a embedded metódy)
* Klasifikácia

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, ktorá ešte dokáže dobre rozdeliť vzorky.

Cieľom práce je zistiť 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. 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 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, takže niektoré skupiny atribútov neboli schopní získať.

Samozrejme, malvér sa 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, urobiť zase selekciu jednotlivých atribútov a riešenie by fungovalo naďalej. Samotné metódy klasifikácie je tiež nutné časom preučiť a pri preučení by sa mohla robiť aj nová selekcia.

Labeling

Triedy sme sa rozhodli získavať pomocou značenia antivírusových programov (AV) 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 pri klastrovaní 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 6.66. Pre porovnanie sme si nechali aj druhý dataset v ktorom ostali aj tieto vzorky aby sme mohli 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 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“ preto sme sa rozhodli nerozbaľovať vzorky.

Nami navrhované riešenie označenia vzoriek spočívalo v spojení označení rôznych antivírusových programov vzorky do jedného dlhého slova, a klastrovaní týchto slov. Náš predpoklad bol, že vzorky ktoré by mali byť rovnakej triede jeden antivírus 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. Za metriku sme si vybrali Levenshtein distance [52], prípadne Damerau-Levenshtein distance [53], keďže 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 na iné. Damerau-Levenshtein distance pridáva výmenu pozícií pre dve písmena vedľa seba.

Ako algoritmus pre klastrovanie sme si vybrali HDBSCAN. 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. Tento algoritmus dokáže správne klastrovať aj rôzne husté klastre [51]. Hierarchické klastrovanie sme chceli preto, lebo sme nevedeli dopredu určiť aký veľký počet klastrov je optimálny.

Druhá možnosť je za triedu považovať najčastejšie sa vyskytujúce sa slovo pri označení rôznych AV - konsenzus. 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). Preto najprv identifikujeme príliš generické slová, tie z tried vynechávame a potom na zvyšných triedach hľadáme aliasy ktoré počítame za jednu triedu. Vzorka 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 klastrovanie a konsenzus.

Vyskúšali sme obidve prístupy k labelingu a ich presnosť sme ohodnotili pri klasifikácii. Klastrovanie malo lepšie výsledky, ale len minimálne (približne pol percenta), preto skúsime selekciu atribútov pre obe riešenia. Podrobnejšie opíšeme testy v tretej kapitole. Rozmýšľali sme aj nad kombinovaným prístupom – získanie tried cez najčastejšie slovo 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 dlhšímí názvami ktoré k sebe naozaj patrili (napr. „wannacry“ a „wannacryptor“) už boli zlúčené skupiny s krátkymi názvami (napr. „dorv“, „worm“). Je to kvôli tomu, že pri Leveshtein distance je vzdialenosť minimálne rozdiel dĺžky slov. Toto kombinované riešenie sme nakoniec nepoužili.

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 2-gramy (teda berieme len unigramy a bigramy) kvôli pamäťovým obmedzeniam, aj keď často sa používali aj 3-gramy a 4-gramy a niektoré práce zašli až po 10-gramy (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 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 jednotlivých položiek vo vzorke to neplatí. Aj samotné majú rozlišovaciu schopnosť medzi malvérom a benignwarom (neškodný softvér) [22]. Odteraz myslíme pod extrakciou n-gramu získanie oboch verzií.

Atribúty možno rozdeliť podľa pôvodu na statické a dynamické. V nasledujúcich podkapitolách sú zoznamy atribútov ktoré sme extrahovali z oboch skupín.

Statické atribúty

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 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]. 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. // TODO prílohu prepíš do slovenčiny, je ešte z čias keď som chcel písať v angličtine.

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 bajtov) [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 [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, na získanie štatistík o entropii. My sme vyskúšali viacero dĺžok, pri žiadnej prvotná selekcia nič nevylúčila, nakoniec sme mali okno s veľkosťou 1024 bajtov, s posunom 256 bajtov. Histogram sme mali rovnaký ako Saxe a Berlin [54], použili sme ich zdrojový kód.

V disasemblovanej vzorke n-gramy predstavujú postupnosť assembly inštrukcií (opcode). Keďže inštrukcie môžu mať parametre ktoré sa môžu odkazovať na miesta v súbore tak do n-gramov berieme inštrukcie bez parametrov, ináč by ich vzniklo enormné množstvo. Registrov nie je až tak veľa a preto môže ako atribút slúžiť frekvencia použitia registrov [27]. My sme robili pre registre priamo n-gramy, frekvencia je v nich zahrnutá. Ďalšie atribúty ktoré pomôžu zachytiť viac informácie o inštrukciách sú frekvencia dĺžok riadkov a ich priemerná dĺžka, počet riadkov, použitých inštrukcií (aj hexadecimálna verzia) a registrov. Pre zistenie koľko miesta si vzorka 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 si miesto. To isté sme urobili pre všetky jump inštrukcie [8].

V disasemblovanej vzorke sa vyskytujú inštrukcie aj v hexadecimálnom tvare (aj s parametrami). Na týchto inštrukciách 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 celých inštrukciách. Poslednými atribútmi sú veľkosť súboru a pomer jeho veľkosti s veľkosťou pôvodného súboru a hexadecimálnej verzie. 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.

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 v hlavičke PE súboru. My sme vybrali nasledujúce metadata: CodeSize, LinkerVersion, SubsystemVersion, InitializedDataSize, UninitializedDataSize, OSVersion, ImageVersion, TimeStamp, EntryPoint, MachineType, Size. 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 každej 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 viažuce 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. 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 venujeme 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]. Sledujeme aj počet reťazcov ktoré tvarom zodpovedajú IP adresám (IPv4). Po získaní reťazcov z nich extrahujeme nasledujúce všeobecné atribúty:

* n-gramy cez znaky a slová
* frekvencie dĺžky reťazcov v intervaloch a priemerná dĺžka reťazcov
* Počet reťazcov a veľkosť súboru v ktorom sú
* entropia celého zoznamu reťazcov

Dynamické atribúty

Pre získanie dynamických údajov používame process monitor [91] ktorý zachytáva systémove volania pre manipuláciu so súbormi, priečinkami registrami, vláknami a procesmi. Tieto volania zahŕňajú tvorbu, mazanie a zmenu súborov, priečinkov a registrov, čítanie hodnôt v registroch a metadát súborov a priečinkov. Ďalej obsahujú tvorbu a ukončenie vláken a procesov cieľovým procesom. Zachytáva tiež komunikáciu procesu cez internet (TCP, UDP). Sleduje aj načítanie knižníc a programov procesom (image load sekcia).

Výstupy process monitora sme rozdelili na skupiny (súborový systém, registre, vlákna/procesy, Internet, import/load). Na každej skupine sme urobili n-gramy na jednotlivých operáciách ktoré spomíname v predošlom odstavci. Pri každej skupine sme zaznamenali počet rôznych operácií. Pri súborovom systéme sme zaznamenali aj frekvenciu použitia jednotlivých súborov a priečinkov, to isté sme urobili aj pre registre. Pre procesy/vlákna sme pridali aj ich celkový počet vytvorený vzorkou. Pre dynamické načítavanie sme vytvorili rovnaké atribúty ako pri statickom importovaní.

//TODO – ak bude čas, skúsim ich získať ešte raz ináč a pridám ich aj do tretej kapitole k extrakcii. Ak nie, vymažem túto sekciu.

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 pri klasifikácii byť 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 v takmer úplnej väčšine datasetu tak ho 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. Pri binárnej premennej to je 99% rovnakých hodnôt, pri premenných kde sú rozdiely medzi hodnotami väčšie to je ešte 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%) a ak bolo nad desať tisíc položiek tak prah bol 0.1 (90%). Pri počítaní variancie je potrebné prechádzať dáta dva krát, ale existujú algoritmy na jeden prechod ktoré dokážu vypočítať varianciu s malou chybou [141].

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.

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 a preto sme museli buď pomalšie algoritmy vylúčiť, alebo rozdeliť jemnú selekciu na dve etapy. Rozdelenie na etapy má nevýhodu – algoritmus použitý v prvej etape môže vylúčiť atribúty ktoré by lepší algoritmus v druhej etape nevylúčil a kvôli tomu môže mať horšie skóre oproti ostatným algoritmom v druhej etape ktoré pracujú ináč. 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 ktoré toto množstvo zvládnu časovo aj pamäťovo. Ale, ako sme v úvode spomínali, máme aj osobitné datasety skupín do tisíc atribútov a skupín do sto atribútov. Prvý dataset označujeme ako jednoduchý, druhý ako veľmi jednoduchý. Na týchto dvoch datasetoch použijeme aj ostatné metódy. Takto zabezpečíme že všetky metódy porovnáme na rovnakých dátach ktoré nebudú ovplyvnené iným algoritmom. Navyše získame výsledky na dátach ktoré sú pre nás zaujímavé, lebo môžu viesť k rýchlej extrakcii.

Rýchle metódy

Tu sme vybrali hlavne algoritmy založené na štatistickej teórii ktoré majú vo všeobecnosti veľmi nízku výpočtovú zložitosť ale nevedia sa vyhnúť redundancii a preto 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 teórií 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. Potom 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 informácie. 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,
* ANOVA F-score (Analysis of Variance)
* Gini index score
* Fisher score
* Laplacian score
* Trace ratio
* SPEC (Spectral Feature Selection)
* ReliefF
* Mutual Information (Information Gain)

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

Najjednoduchšia verzia ANOVA ktorú využijeme aj my v sklearn je One way Anova for independent samples. Tá predstavuje pomer agregovaných rozdielov medzi priemermi atribútu v jednotlivých triedach a náhodnej variability ktorá existuje vnútri tried [143]. Popisuje ju vzorec (14). a sú popísané vo vzorci (15). Prvé predstavuje varianciu atribútu medzi skupinami. Je to suma kvadratických odchýlok priemerov jednotlivých skupín od priemeru datasetu nádobená ich veľkosťou. Druhé je suma kvadratických odchýlok prvkov jednotlivých skupín od priemeru skupiny. *dfbg* je stupeň slobody medzi skupinami, tu je rovný počtu skupín mínus jedna. *dfwg* je stupeň slobody vnútri skupín – je to rozdiel súčtu počtu prvkov jednotlivých skupín atribútov rozdelených podľa tried a počtu tried.

Pri F-value sa testuje nulová hypotéza že prvky v jednotlivých triedach boli vybrané náhodne zo spoločného zdroja. Obe MS hodnoty sú odhadom variancie tohto zdroja. Ak je hypotéza pravdivá tak F je rovné alebo menšie ako 1, ak nie tak je o dosť väčšie 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 populácue. 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.

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

*C* predstavuje počet tried. Pre všetky triedy sa počíta podmienená pravdepodobnosť triedy za predpokladu vzniknutej skupiny *W*, resp. . To, čo Jundong Li, et.al. nazývajú gini index score korešponduje s gini impurity gain ktorú si predstavíme pri rozhodovacích stromoch. Pre *n* tried možno postupne deliť vzorky až do *n* skupín ak má atribút dosť hodnôt. Pri tom sa ale zvýši zložitosť, lebo je potrebné vytvoriť všetky kombinácie hodnôt atribútu až po počet tried. Počet rozdelení by potom bol: kde *n* je počet rôznych hodnôt atribútu a *k* je počet tried. Výpočtovo lacnejšie riešenie je po nájdení najlepšieho rozdelenia rekurzívne hľadať najlepšie rozdelenia pre vzniknuté skupiny vzoriek, čo sa robí pri stromoch. //poznámka: viaceré články si zamieňajú gini index a gini impurity, aj keď sú to dosť rôzne veci.

Laplacian score patrí medzi algoritmy založené na podobnosti. Algoritmus je založený na myšlienke, že dva body pravdepodobne súvisia ak sú blízko seba. Nižšie skóre je lepšie. Algoritmus prebieha nasledovne [146]: Najprv sa urobí graf susednosti medzi hodnotami atribútu vo všetkých vzorkách. Ak nemáme rozdelenie na triedy tak za suseda sa považuje *k* najbližších susedov. Ak poznáme rozdelenie tried tak najbližší susedia sa berú len z rovnakej triedy. Potom sa urobí matica kde pre spojené hodnoty *xi a xj* políčko nadobudne pre vhodnú konštantu *t* hodnotu (17). Pre políčka bez susedov bude mať nulovú hodnotu.

*Lr* je skóre *r*-tého atribútu *fri* je hodnota atribútu v *i-*tej vzorke *r*-tého atribútu pričom máme *m* vzoriek. Pre *r*-tý atribút sa definujú začiatočné argumenty (18):

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

Nakoniec sa vypočíta samotné skóre (20).

Trace ratio vyberá najlepšiu podmnožinu atribútov pomocou Fisher alebo Laplacian score. Algoritmus je nasledovný [147]: Vstup je *d* atribútov v matici *X* z ktorých sa nakoniec vyberie *m*. Vytvorí sa selektovacia matica *W* o veľkosti *d\*m* ktorej *i*-tý stĺpec *wi* má jednotku len na *i*-tom mieste, inde nuly. Vytvoria sa dve grafy susednosti ako pri Laplacian (respektíve Fisher) score, ale jeden bude predstavovať lokálnu vzdialenosť vnútri tried a má väčšie hodnoty pre prvky ktoré majú podobné hodnoty alebo sú v rovnakej triede a druhý graf predstavuje globálnu vzdialenosť prvkov atribútu medzi triedami – má väčšie hodnoty pre prvky ktoré sa líšia alebo sú v rôznych triedach. Z nich sa vytvoria Laplacian matice *Lw a Lb*. Skóre *i*-tého atribútu je vo vzorci (21).

Myšlienka algoritmu je maximalizovať podobnosť prvkov atribútu vnútri tried a zároveň minimalizovať podobnosť medzi triedami [24]. Existuje aj verzia ktorá konverguje ku globálne najlepšej podmnožine atribútov a berie do úvahy skóre celej podmnožiny [147].

SPEC je rozšírenie Laplacian score. Namiesto Laplacian matice má jej normalizovanú verziu (22).

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

γ je funkcia ktorá penalizuje vysoku frekvenciu hodnôt atribútu aby redukovala šum [24]. Ak sa použije RBF pre vypočítanie vzdialenosti a nepoužije sa γ tak zložitosť SPEC je *O(AV2)* pre *A* atribútov a *V* vzoriek [148].

ReliefF má zložitosť *O (MVA)* 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(VA)*. 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. Preto 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 *Ri*. Pre túto vzorku sa nájde *k* najbližších susedov v jeho triede (*Hj*) a potom pre každú ďalšiu triedu *C* sa nájde ďalších *k* najbližších susedov *(Mj(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).

Funkcionalitu Mutual information sme vysvetlili už v prvej kapitole. Algoritmus ktorý používa sklearn pre Mutual information získa entropiu odhadom od vzdialeností *k* najbližsích susedov. Takéto algoritmy majú komplexitu až pri hladkej distribúcii [150].

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. Tiež nepovažujeme presnosť klasifikátora s tým, že sa používa ako black box za dobré hodnotenie množiny atribútov. Táto selekcia sa nedá nijako interpretovať. Ďalší problém je, že 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 metódy [24]. Tieto metódy ale priamo využívajú štruktúru klasifikátora a jeho trénovací algoritmus, neberú ho ako blackbox. Navyše majú iné výhody. Na rozdiel od filtrovacích metód ensemble DT metódy vedia zachytiť aj interakcie medzi viacerými atribútmi (nielen dvojicami) pri svojom použití filtrovacích metód [69]. Z tohto dôvodu ich budeme 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 filtrovacích algoritmoch sme sa zamerali na tie, ktoré odstraňujú irelevantné aj redundantné atribúty čo znižuje nevýhodu ich pomalosti. 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 [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 (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 a je možnosť vybrať si stratégiu hľadania: forward selection, backward elimination a best first [26]. CFS je popísané vo vzorci (25). *S* je k-prvkovká podmnožina atribútov, *rcf* je priemerná korelácia medzi triedami a atribútmi, a *rff* 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 (A4)* [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].

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 *δ*. Ako koreláciu používa symetrickú neistotu (SU). SU medzi atribútom *Xs* a rozdelením tried *Y* sa počíta pomocou vzorca (14) kde *H* je entropia a *IG* je informačný zisk. Korelácia medzi atribútom *Fi* a triedou *C* sa nazve predominantnáak je väčšie ako prah a zároveň pre žiaden iný, už vybraný atribút *Fj* neplatí že . Tie atribúty pre ktoré táto nerovnosť platí sú označené ako redundantné vzhľadom na *Fi* a uložia sa do množiny *Spi*. Samotný atribút *Fi* je predominantý 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 sa uložia do *Spi+* a tie, pre ktoré platí opačná nerovnosť sa uložia do *Spi-*. Zo všetkých troch skupín sa potom heuristicky odstránia niektoré atribúty. Štartovacia heuristika vyberie za predominantný atribút ten *Fp*, ktorý má najvyššiu koreláciu s rozdelením tried. Potom sa skontrolujú atribúty s nižšou koreláciou s triedami. Všetky, ktoré sú redundantné pre *Fp* sa odstránia a nové *Fp* bude atribút s najvyššou koreláciu s triedami ktorý sa neodstránil a 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.

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 (28). MIFS pridáva ku IG penalizáciu sa redundanciu ktorá sa prejavuje v časti vzorca ktorá sa odčítava kde penalizuje atribúty ktoré korelujú s už vybranými atribútmi. MRMR len špecifikuje z MIFS ako , 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 [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]. CIFEku 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.* 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, popísané vo vzorci (29) – 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.

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 použiť ich ako naše klasifikátory. 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))* kde *d* je maximálna hĺbka stromu, *K* je počet stromov, *m* je počet atribútov a *n* je počet vzoriek (exact greedy algoritmus). 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 Information gain berú do úvahy hlavne vzorky s vysokým gradientom, teda tie ktoré ešte potrebujú viac zmenšiť svoju chybu. Pre zrýchlenie pri riedkych (sparse) dátach sa používa Exclusive Feature Bundling – zlučujú sa atribúty ktoré nenadobúdajú naraz nenulové hodnoty, prípadne ich naraz nadobúdajú málokedy [95].

**Gradient boosting** popíšeme v nasledujúcej sekcii [30]: Pre dataset s *m* atribútmi X, cieľovým atribútom Y a *n* príkladmi stromový ensemble používa *K* aditívnych funkcií na predikciu výsledku (30).

V tomto vzorci je je priestor regresných stromov. *q* reprezentuje štruktúru každého stromu ktorá mapuje príklad na index korešpondujúceho listu (T je počet listov). Každé *fk* korešponduje s jedným stromom *q* a váhami jeho listov *w*. Na rozdiel od decision trees má každý strom skóre na každom liste a *wi* je skóre *i*-tého listu. Pre každý príklad sa pomocou rozhodovacích pravidiel *q* stromy rozhodnú klasifikovať príklad do jedného zo svojích 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).

Tu je *l* diferencovateľná konvexná chybová funkcia na počítanie rozdielu medzi predikciou a reálnou cieľovou hodnotou . 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 nasledovnou minimalizáciou (32):

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

V tomto vzorci je a sú gradienty chybovej funkcie prvého a druhého stupňa.

Keďže regularizačný parameter je pri stromoch rovný možno pre boosting vypočitať optimálnu váhu každého listu *j* kde *Ij* je množina príkladov v tomto liste (34).

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

Toto skóre sa používa na nájdenie najlepších stromov. Jednotlivé stromy sa skórujú podobným spôsobom ako decision trees pri random forest kde sa používa impurity. 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 *IL* a *IR* sú množiny príkladov v ľavej a pravej vetve po rozdelení a *I*je ich prienik, tak zníženie chyby po rozdelení je vo vzorci (36).

Skóre atribútu môže závisieť od toho, koľko krát bol použitý pri najlepšom rozdelení (split) alebo o koľko znížili chybu rozdelenia v ktorom bol použitý (gain), prípadne priemerný gain na split.

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 impurity tak v uzle sú rovnomerne prítomné vzorky zo všetkých tried. Ak je nulová, všetky vzorky pochádzajú z rovnakej triedy [154]. Existuje viac typov impurity. Entropy impurity množiny *S* pre počet tried *C* kde *Si* sú vzorky z triedy *i* je vo vzorci (37) [154] kde je pomer veľkosti a celého *S* v uzle.

Ďalšia miera nečistoty je Gini impurity, 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]:

Pri delení uzlu je cieľom, aby nové uzly mali čo najväčší pokles nečistoty (gain - zisk) 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]:

*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 synov na jednotlivé množiny *Si* a vypočíta sa pokles. Keďže prvý výraz vo vzorci – impurity 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 použitý gini impurity [154]. Ak sa použije entropy impurity tak ziskom je information gain [93]. Problém so ziskom je, že atribúty ktoré majú viac možných hodnôt majú väčšiu šancu že rozdelia uzol na čistých synov. Preto sa používa gain ratio kde sa získaná entropia normalizuje entropiou rozdelenia (40) [154]:

V náhodnom lese potom jednotlivé stromy hlasujú na priradení triedy. Náhoda môže mať viacero podôb – 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 nezávisly, 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 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 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]:

Násobok optimálneho riešenia je tiež optimálne riešenie. Preto 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: . Teraz môže byť od separujúcej posunutá o malú kladnú chybu: Toto obmedzenie sa použije pri novej optimalizácii kde sa berie do úvahy aj veľkosť chyby (42).

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

Tento duál sa stále počíta kvadratickým programovaním, ale už ma už iné, jednoduchšie obmedzenia: .

Pre *d* atribútov (dimenzií) a *n* vzoriek (bodov) má primárna optimalizácia zložitosť *O(nd2+d3)* a duálna má zložitosť *O(dn2+n3)*. Preto 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*). 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 dvoni stratégiami. Prvá je one-versus-rest keď sa postupne snaží rozdeliť jednotlivé triedy od zvyšku datasetu. V druhej, one-versus-one sa snaží oddeliť postupne dvojice tried.

Vo vzorci (42) sme spomenuli regularizáciu**. Regularizácia** sa dá použiť pre všetky typy lineárnych modelov, nie len 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]:

SVM vo vzorci (42) prirodzene využíva ℓ*2*-normu, ale používa sa aj ℓ*1*-norma.

Regularizácia sa používa aj pri selekcii atribútov kvôli tomu, že regularizátor spôsobuje že kofecienty (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. *ℓ0*-norm regularizácia priamo hľadá optimálnu množinu nenulových atribútov. Optimalizačný problém sa stáva problém integer programovania ktorý je náročné riešiť (NP úplné). Preto sa viac používa *ℓ1*-norm regularizácia ktorá je najužšia konvexná relaxácia *ℓ0*-normy [24].

Pri multi-class klasifikácii (prvok môže patriť do viacerých tried) alebo multivariate regresii sa využívajú *ℓp,q*-normy [24]. V tomto prípade je aj výstup jedného príkladu vektor a nie číslo čo sa odráža aj v norme. *ℓp,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].

Na selekciu atribútov pomocou *ℓ1*-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 Lasso predstavuje optimalizáciu vo vzorci (46) [68] kde je vektor koeficientov regresie (váhy), a sú *ℓ1*- a *ℓ2*- normy. je regularizačný parameter.

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

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

je Frobenius norms čo je *ℓ2,2*-norma, a sú centrované Gram matice , a *K(x,x‘)* a *L(y,y‘)* sú kernel funkcie. je centrovacia matica, je *n*-rozmerná identická matica a 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 ktoré majú 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).

V tejto rovnici je ( je meradlo nezávislosti založené na kerneli ktoré sa označuje Hilbert-Schmidt independence criterion (HSIC) a *tr* znamená trace (suma elementov na hlavnej diagonále). je konštanta. *HSIC* môže mať len kladné hodnoty a je nulový len keď sú dve atribúty štatisticky nezávislé. Keď má atribút veľkú závislosť na rozdelení tried ***y*** tak má tiež vysokú hodnotu čo spôsobí že bude mať tiež vysokú hodnotu aby sa celý výraz minimalizoval. V opačnom prípade (nezávislosť) bude hodnota blízka nule a bude vo všeobecnosti eliminované regularizácou. Po regularizácii ostanú len relevantné atribúty ktoré sú závislé na výstupnom atribúte. Podobne má veľkú hodnotu pri závislých atribútoch, čo znamená že jeden z nich je redundantný. Keďže tu sa výraz pripočítava tak bude veľmi malé kvôli minimalizácii celého výrazu. Vo všeobecnosti jeden z nadobudne nulovú hodnotu takže redundantné atribúty sú eliminované.

HSIC lasso má pamäťovú zložitosť *O(d\*n2)* kde *d* je počet atribútov a *n* je počet vzoriek, ale 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 kernelu a jeho násobenia potom je *O(dbn+mb2dn) = O(mb2dn)* kde *m* je počet vybraných parametrov. Pri naivnej verzii by bola zložitosť *O(mdn3)* s pamäťou *O(d\*n2)* [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. Ako metriku použijeme accuracy. Pôvodne sme chceli použiť ROC-AUC krivku, ale tá je pre binárnu klasifikáciu a my máme veľa tried. Accuracy je v nami analyzovaných článkoch o klasifikácii malvéru najčastejšia metrika, a preto budeme vedieť porovnať svoje výsledky s ostatnými prácami. Naším výsledkom je priemerná accuracy pre 10-fold krosvalidáciu.

Implementácia a postup

Celý postup sme robili nad datasetom malvéru ktorý bol v rokoch 2017 a 2018 získaný zo služby VirusTotal. V tomto datasete je 90355 vzoriek ktoré majú veľkosť 109 GB. // Laco asi vie o datasete viac.

Tvorba datasetu

V pôvodnom datasete je príliš veľa vzoriek na to, aby sme zvládli spracovať všetky. Preto sme najprv dataset vyčistili. Dataset sme rozdelili na vzorky a reporty a vymazali sme všetko, čo nemalo svoj pár. Takýchto prípadov bolo veľmi málo. Po tomto kroku sme mali 90355 vzoriek 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, 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. Na nájdenie typu sme použili príkaz find s parametrom type v linuxe. 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, nemali informácie o sekciách alebo o importoch. Predpokladali sme, že tieto súbory museli byť poškodené keď sa z nich nedali získať tieto informácie. Po tomto kroku nám ostalo 79856 vzoriek. Vybrali sme si 6 antivírusových programov ktoré neskôr slúžili na tvorbu labels. Najprv sme zistili koľko vzoriek označili ako false, a v koľkých sa nevyskytli. Počty sú nasledujúce (prvé číslo je false, druhé nevyskytujúce sa):

Kaspersky: 10145 + 366 = 10551

Microsoft: 18951 + 215 = 19166

McAfee: 4628 + 203 = 4831

Eset: 6008 + 39 = 6047

BitDefender: 5097 + 1009 = 6106

Malwarebytes: 38499 + 3128 = 41627

Microsoft a Malwarebytes sme sa rozhodli vylúčiť, lebo ako false označili príliš veľa vzoriek. Pre zvyšné štyri AV sme odstránili všetky vzorky ktoré označili ako false alebo pri nich neboli všetky štyri AV použité. Ostalo nám 63428 vzoriek. Pôvodne sme odstraňovali aj vzorky ktoré mali globálne viac false ako malware hodnotení, ale vylúčilo sa nám príliš veľa vzoriek, preto sme sa sústredili len na vybrané AV.

Potom sme rozdelili dataset na tri – v prvom sme sa snažili odstrániť zabalené a obfuskované vzorky, v druhom sme nechali všetky vzorky (63428) – zmiešaný dataset, v treťom sme si nechali len vzorky ktoré sme chceli v prvom odstrániť. V prvom datasete sme v reportoch hľadali reťazce „packe“, „obfuscat“, „encrypt“ na nájdenie zabalených, obfuskovaných a zašifrovaných vzoriek. Síce sme mohli mať aj veľa false positive ale nám išlo hlavne o to, aby sme mali čo najmenej takých 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 ako sme písali v predošlej kapitole. To nám odstránilo približne polovicu vzoriek, ostalo nám ich 23595. V treťom datasete sme zase vyhodili vzorky ktorých reporty neobsahovali reťazce „packer“, „obfuscat“, „encrypted“, „packed“, „unpack“. Ostalo nám 23452 vzoriek. Nasledujúce kroky sú rovnaké pre všetky datasety.

Labeling

Pre každý zo štyroch vybraných AV sme zistili, do akých tried zaradzuje malvér. Každé označenie vzorky sa skladá z jej hierarchického názvu ktorý žiaľ nie je nijako štandardizovaný. Vo všeobecnosti je na začiatku platforma alebo typ malvéru, potom skupina/trieda, podtrieda, predpona, meno a prípona. Všetko sa oddeľuje bodkou. Preto sme cez bodku oddelili každý výstup, zmenili veľké písmená na malé a pre každý AV osobitne sme spočítali koľko krát 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 slová ktoré sa vyskytovali v menej ako 50 vzorkách, lebo takéto triedy by boli príliš malé v porovnaní s ostatnými a tieto slova aj tak pravdepodobne boli podtriedy alebo ešte jemnejšie rozdelenie. Potom 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, lebo by sa nám zlúčili vzorky rovnakého typu ale z rôznych tried do jednej, príliš všeobecnej. Tu je ich zoznam:

"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. Potom sme hľadali prienik medzi triedami ktoré ostali. Pre dataset bez obfuskovaných 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 sme zase brali len dve zhody, ináč nám ostalo málo tried. Pre prvý dataset nám vzniklo 10 tried so 5506 vzorkami. Rozdelenie tried bolo nasledovné:

('crytex', 1135), ('msil', 917), ('upatre', 895), ('zbot', 553), ('sality', 519), ('vbkrypt', 491), ('neshta', 487), ('runouce', 326), ('linkury', 96), ('wannacry', 86)

V druhom bolo 8 tried, s 5851 vzorkami s týmto rozdelením:

('sality', 1627), ('upatre', 1118), ('zbot', 962), ('dealply', 649), ('expiro', 616), ('locky', 488), ('wannacry', 210), ('linkury', 181)

Tretí dataset mal X tried s X vzorkami. Rozdelenie tried bolo nasledovné:

//TODO dopíš rozdelenie tried pre tretí dataset

Keďže používame accuracy, 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 ignoruje, lebo z najväčšej triedu budú najväčšie zisky v accuracy, kým rozdelením najmenšej sa získa minimum. Preto sme v prvom datasete obmedzili veľkosť triedy na 100, v druhom na 200 a v treťom na X //TODO dopíš. V prvom nám teraz ostalo 982, v druhom 1581 a v treťom X //TODO dopíš vzoriek. Pri tvorbe atribútov ktorú sme robili po tomto kroku sme niektoré vzorky museli vylúčiť, neskôr popíšeme prečo. V prvom datasete sme vylúčili 13, v druhom 22 a v treťom X //TODO dopíš vzoriek. V najmenšej triede zostal rovnaký počet vzoriek, takže nerovnosť medzi triedami sa ešte znížila. Teraz majú datasety 969, 1559 a X //TODO dopíš vzoriek. Toto sú finálne veľkosti datasetov. 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 tieto triedy zlučovať cez Levenshtein distance aby sme spojili triedy ktoré v rôznych AV majú len drobné odchýlky. Výsledky boli veľmi zlé, lebo 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.

Náš predpoklad bol, že vzorky s podobnými označeniami budú mať podobnú aj časť pred triedou ako sme spomínali v predošlej kapitole. Najprv sme vyskúšali na celých názvoch (aj s predponami a príponami) identitu – hľadali sme vzorky ktoré majú úplne rovnaký celý názov v dvoch rôznych AV, ale ostali nám len dve triedy nad 50 vzoriek, takže sme totožnosť zavrhli. Potom sme klastrovali tieto celé názvy cez HDBSCAN. Robili sme to tak, že celé označenia so všetkých štyroch AV sme spojili do jedného slova a na Levenshtein distance medzi týmito slovami slúžila ako vzdialenosť. Maticu vzdialenosti sme dali na vstup pre HDBSCAN. Minimálnu veľkosť klastra sme zadali 86, čo je veľkosť najmenšej triedy v predošlom rozdelení.

V prvom datasete nám vylúčil 57 vzoriek ako outliere, ostalo ich 912. Zvyšné vzorky klastroval do 9 tried. Početnosť tried bola nasledujúca:

128, 114, 102, 101, 100, 100, 95, 86, 86.

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

//TODO dopíš, dopíš aj pre tretí dataset.

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](mailto:W32/Chir.b@MM.virus)

Microsoft: Virus:Win32/Chir.B@mm

Symantec: [W32.Chir.B@mm](mailto: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](mailto:Email-Worm.Win32.Runouce.b#W32/Chir.b@MM)

[Email-Worm.Win32.Runouce.b#W32/Chir.b@MM#Win32/Virut.NBP#Win32.Runouce.B@mm](mailto:Email-Worm.Win32.Runouce.b#W32/Chir.b@MM)

Ako vidíme, 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. Aj tak, vzorky ktoré by dostali od dvoch AV označenia „runounce“, od dvoch „b@mm“ a od jedného „virut“ teraz patria do jedného klastra.

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 pri jednej triede takmer vždy priradil rovnaký názov (aj keď to nemusel byť práve názov triedy) a vo veľmi malom počte vzoriek priradil iný názov tak po spojení označení zo všetkých AV to malé množstvo vzoriek bude mimo klastra – vznikne outlier. Toto je ďalšia výhoda klastrovania – nevadí keď rôzne AV používajú rôzne označenia, stačí že navzájom sú ich označenia konzistentné. Ak nie sú, tak pravdepodobne došlo k nesprávnemu označeniu vzorky. Toto predošlým prístupom nevieme odhaliť.

Extrakcia dát

Po tom, čo sme vyčistili dataset sme zo vzoriek 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], lebo jeho výstup mal menšiu veľkosť ako výstupy ostatných programov (umožňuje výstup bez ASCII prekladu aj offsetu).

Na získanie disasemblovaného súboru sme použili objdump [78], lebo umožňuje disasemblovať všetky sekcie, nie len tie, v ktorých sa očakáva kód. To sa hodí pri vzorkách s premenovanými sekciami a droppermi. Pri niektorých vzorkách ale hádzal 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 pár vzorkách mu zase došla pamäť, čo sa môže stať pri rekurzii keď sa robí demangling [78].

Preto sme vyskúšali radare2. Radare2 za normálnych okolností otvorí vlastný terminál v ktorom možno používať príkazy programu na súbore, ale 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) sa dá využiť r2pipe [157], vďaka čomu možno príkazy pre Radare zadávať priamo cez programovací jazyk. R2pipe podporuje množstvo jazykov. My sme použili verziu pre Python. Výstupy ale boli príliš veľké (230 MB oproti 55 MB pre objdump na 5 MB súbore) a spracovanie bolo o dosť pomalšie takže sme nakoniec ani nespustili Radare2 na celom datasete. Preto sme sa rozhodli použiť objdump a súbory pri ktorých mal počas behu chyby sme vyhodili. Z tohto dôvodu sme odstránili 13 vzoriek z prvého datasetu, 22 z druhého a X //TODO dopíš z tretieho. Tieto vzorky sme spomínali v minulej podkapitole.

Textové reťazce sme získal linux príkazom strings pričom sme brali len reťazce s minimálne 5 znakmi. Entropiu sme získali rovnakým spôsobom ako keď sme podľa nej čistili prvý dataset – cez sigcheck.

Extrakcia atribútov

Počas extrakcie sme vykonávali aj prvotnú selekciu lebo sa extrahovalo príliš veľké množstvo atribútov a nechceli sme všetky ukladať a zase spracovávať pri selekcii. Bližšie máme túto selekciu popísanú v nasledujúcej podkapitole. Tu 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 ktoré pochádzajú zo skupín ktoré majú do tisíc atribútov a nakoniec tie, ktorých skupiny majú len do sto atribútov. Na jednotlivých skupinách budeme robiť selekciu a klasifikáciu a tak zistíme, či postačujú aj malé skupiny atribútov.

Prvý dataset

Z importov sme mali 172 knižníc so 6424 funkciami. Po prvotnej selekcii sme mali 40 knižníc a 457 funkcií. Žiaden zo 103 exportov neprešiel cez DF, ostal nám len atribút ich počtu ktorý prešiel cez varianciu.

Pri metadátach sme nerobili DF lebo nemá zmysel - všetky metadáta sa vyskytujú vo všetkých vzorkách. Variancia vyhodila len MachineType, takže nám ostalo 10 metadát. Pri entropii súboru sme nerobili prvotnú selekciu - entropiu má každý súbor a hodnoty sa líšia dosť na to, aby prešla varianciou. Overlay atribúty využili len varianciu, nič nevyhodila a ostali všetky 3 atribúty. Tak isto sme postupovali pri sekciách, lebo máme malý počet známych sekcií. Po selekcii varianciou nám zo všeobecných atribútov vypadol počet sekcií s prázdnym menom, ostatné ostali. Celkovo z 56 atribútov nám ostalo 42.

Zo sekcie .edata nám vypadli všetky atribúty, čo znamená že sa nezvykne používať. U viacerých sekciách nám vypadol pomer ich veľkosti k veľkosti súboru (ich veľkosť nemala odchýlky) a entropia, u sekcii .bss aj samotná veľkosť, čo naznačuje že mala prakticky konštantnú, pravdepodobne nulovú veľkosť aj entropiu. Jediné sekcie 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, ale tá 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, a preto, lebo tak malé hodnoty pri normalizácii ešte rapídne zmenšia sme sa rozhodli, že budeme deliť veľkosť súboru s veľkosťou sekcií. Pomer medzi sekciami sa zachová, len veľkosti budú reverznuté. Teraz nám ostalo 47 atribútov. Takýto reverznutý pomer používame aj pri ostatných atribútoch pri ktorých sa delí veľkosťou súboru.

Pri zdrojoch (resources) sme mali 48 typov zdrojov, ostalo nám 18 a atribút vyjadrujúci ich celkový počet. Ďalšie boli n-gramy z reťazcov. Pri n-gramoch sme už vždy používali DF aj varianciu. V reťazcoch bolo spolu 419217 odlišných slov. Súbory mali spolu 647349 2-gramov a 741590 3-gramov slov (1-gram je celé slovo). Po prvotnej selekcii nám ostalo 964 binárnych a 9245 frekvenčných 1-gramov, 304 binárnych a 13792 frekvenčných 2-gramov a 188 binárnych a 15712 frekvenčných 3-gramov.

Znakov bolo 97, ich dvojíc bolo 9408 a trojíc bolo 641597. Po prvotnej selekcii nám ostalo 94 1-gramov pre obe verzie, 6 binárnych a 89 frekvenčných 2-gramov a 2 binárne a 13 frekvenčných 3-gramov. DF nevylúčilo žiadne 2-gramy a len polovicu 3-gramov. Keďže variancia vylúčila väčšinu 2 aj 3-gramov tak sa museli vyskytovať takmer v každej vzorke vo veľmi podobných množstvách.

Pri reťazcoch nepoužívame DF, lebo maximálna dĺžka reťazca bola 979417 znakov čo je príliš veľké množstvo (viac ako všetky 3-gramy čo som doteraz robil) a väčšina dĺžok by bola všade nulová a tých pár ostatných by pravdepodobne bolo nulových príliš často. Namiesto toho sledujeme iba 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 desiatkách po stovku, potom po stovkách do tisícky a po tisícoch do desaťtisíc. Od desaťtisíc nahor je všetko v jednom intervale. Ostatné atribúty sme popísali v minulej kapitole. Zo 42 atribútov nám variancia nechala 36. Vypadli niektoré intervaly dĺžky reťazcov, hlavne najväčšie – v tisícoch (napr. tisíc až dvetisíc...).

Z hexadecimálnej reprezentácie súboru sme mali 256 1-gramov, 65536 2-gramov a 14004842 3-gramov. Po prvotnej selekcii ostalo 65 binárnych, 256 frekvenčných a 256 pomerov veľkosti súboru a frekvenčných 1-gramov. Pre 2-gramy to bolo 65026, 65536 a 65536. Pri 3-gramoch po 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. Preto sme sa rozhodli, že tieto 3-gramy vylúčime.

Pri byte-entropy histograme 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, tak sme ostali pri 512 a 2048. Pre pomer veľkostí súborov nemá zmysel robiť DF – každý súbor má veľkosť. Variancia vylúčila z 9 atribútov dve.

Potom sme spracovávali disasemblované súbory. 1-gramov hexadecimálnej reprezentácie inštrukcií (inštrukcia je 1-gram) bolo 5388075, po selekcii 29701 binárnych a 92089 frekvenčných 1-gramov. Vzhľadom na enormné množstvo 1-gramov sme 2-gramy nerobili. Túto hexadecimálnu reprezentáciu sme potom brali ako postupnosť bytov a robili sme 1,2,3-gramy na bytoch. Mali sme 256 1-gramov, 65536 2-gramov a 13432761 3-gramov. Po prvotnej selekcii nám ostalo 82 binárnych, 256 frekvenčných a 256 pomerov veľkosti súboru a frekvenčných 1-gramov. Pre 2-gramy to bolo 65412, 65536 a 65536. Pri 3-gramoch po DF ostalo 2812295 3-gramov, čo je stále príliš veľa a preto sme 3-gramy vynechali.

Samotných inštrukcií bolo 1076, ostalo z nich 838 binárnych a 881 frekvenčných 1-gramov. 2-gramov bolo 116270, ostalo 21603 binárnych a 38257 frekvenčných. 3-gramov sme mali 2184555, z toho ostalo 73257 binárnych a 219712 frekvenčných.

Registrov bolo spolu 108, z toho ostalo 75 binárnych a 100 frekvenčných 1-gramov, 2-gramov z 3924 ostalo 2359 a 2737, a 3-gramov z 76520 ostalo 19723 a 30027.

Ďalšie atribúty sa týkali inštrukcií – počet alokačných a jump inštrukcií, pomer všetkých a týchto vybraných inštrukcií, počet inštrukcií, počet hex-reprezentácií inštrukcií a počet registrov. DF sme nerobili, pri variancii vypadli alokačné inštrukcie – ukázalo sa, že neboli v žiadnej vzorke takže to 7 atribútov ostalo 6.

Posledné statické atribúty sa týkali vlastností disasemblovaného súboru (dĺžka, počet riadkov...). Maximálna dlžka riadku je 130 znakov, takže nemuseli sme robiť intervaly. Zo 115 atribútov (aj mean,sum) ostalo 106. Dokopy z 20861316 atribútov (s tým, že sme ďalších asi 82312809 úplne vylúčili – príliš veľké skupiny) ostalo 966466 atribútov. Preto sme sa rozhodli vylúčiť všetky 3-gramy aj 1-gramy hexadecimálnych reprezentácií inštrukcií ktorých bolo na začiatku 5388075. Teraz nám ostalo 486136 atribútov. Po rozdelení na skupiny nám ostalo 5567 atribútov v druhej skupine a 766 v tretej skupine.

Druhý a tretí dataset

Podobný text ako v predošlej podkapitole, doplním neskôr

Selekcia atribútov

Prvotná selekcia a predspracovanie

Ako sme písali v predošlej podkapitole, prvotnú selekciu sme robili počas extrakcie. Skladala sa z document frequency a low variance selection. Pri document frequency 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...) a nulová hodnota teda znamená neprítomnosť. 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, a tak by sa mohlo stať, 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. Preto 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 mala najmenšia trieda X vzoriek, prah bol Y //TODO dopíš.

Keďže v niektorých skupinách bolo príliš veľa atribútov tak sme chceli počet zmenšiť už teraz v prvom kroku. Preto sme pre skupiny v ktorých je nad 100 000 atribútov za hranicu určili 20 pre prvý dataset, 25 pre druhý a X pre tretí //TODO dopíš. Okrem toho sme vylúčili v týchto skupinách aj prvky ktoré v takmer celom datasete nemali nulovú hodnotu – aspoň 950 vzoriek v prvom datasete, 1540 v druhom a X v treťom //TODO dopíš. Tak veľké skupiny majú len n-gramy a náš predpoklad bol, že n-gram ktorý sa vyskytuje takmer všade 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 defaultne, pre skupiny nad 1000 atribútov to je 0.05 a pre skupiny nad 10 000 atribútov je prah 0.1. Ak sme nerobili DF alebo selekciu varianciou pri nejakej skupine, tak to spomíname pri extrakcii aj s dôvodom. Väčšinou sme pri príliš malých skupinách nerobili DF, len varianciu. Hlavný cieľ prvotnej selekcie je redukovať počet atribútov a preto sa sústredíme na hlavne veľké skupiny. Jadro selekcie prebehne neskôr, pričom po každej metóde už aj skontrolujeme presnosť pri klasifikácii a tak 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 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. Preto sme zmenili názov atribútov na meno ich skupiny ktorú vieme kontrolovať.

Pre načítanie a prácu s atribútmi sa v Pythone používajú dve knižnice: NumPy a Pandas. NumPy polia sú rýchlejšie, Pandas dataframe ponúka zase 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 vytvorí jednorozmerné pole zoznamov jednotlivých riadkov ktoré sme nemohli použiť. Pandas taký problém nemala. Ale dve knižnice (pyHSICLasso a skfeature) nepodporujú Pandas DataFrames (lasso podporuje, ale len priamo z csv súboru ktorý musí byť v tvare ktorý náš nemal). Preto 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 z ktorých každý musí zisťovať svoju adresu, preto sú omnoho pomalšie. Okrem rýchlosti má aj výhodu vo veľkosti. Typy np.float64 a np.int64 majú 64 bitov [161], kým float v Pythone má minimálne 24 bytov, 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 z desatinnou čiarkou. Potom odsekneme časť za desatinnou čiarkou na maximálne 5 cifier. Ak ml atribút menej ako 5 cifier za desatinnou čiarkou tak jeho hodnoty vynásobíme 10max, ináč 105 a zmeníme ich na 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 to nechceli robiť, lebo zlučovaním hodnôt by sme stratili časť informácií v atribútoch.

Po diskretizácii robíme aj štandardizáciu. Musíme ju robiť preto, že pre niektoré funkcie ako je napríklaf RBF pre SVM alebo L1 a L2 regularizácie pri lineárnych modeloch nie je dobré ak rôzne atribúty majú príliš veľký rozdiel vo variancii (jeden atribút má veľmi veľké rozpätie hodnôt a druhý veľmi malé) lebo atribút s veľkou varianciou pri učení bude dominovať a tie s malou budú mať minimálny vplyv. Navyše pre tieto funkcie očakávajú vstup centrovaný na nulu [162]. Preto pred tým ako pustíme selekciu alebo klasifikáciu na SVM alebo HSIC lasso tak použijeme standard scaler [162] ktorý 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, polynom a sigmoid 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 mi ostal výsledok aj čas rovnaký ale v dokumentácii sa odporúča štandardizácia pred jeho použitím (pre možné zrýchlenie výpočtu).

Výber metód

Pred tým ako sme začali robiť samotnú selekciu sme sa rozhodli vylúčiť niektoré metódy lebo ich máme príliš veľa a mnohé pracujú na rovnakých princípoch s malými odlišnosťami. Najprv sme potrebovali z rýchlych metód vybrať tie, ktoré zvládnu všetky atribúty. Všetky rýchle metódy sme spustili na 92000 a 220000 atribútoch. Najhorší čas bol 25 minút na 220000 atribútoch a mal ho SPEC. Na týchto atribútoch bola najrýchlejšia ANOVA ktorá nebežala ani dve sekundy. Trace ratio nezbehlo, došla pamäť už pri 92000 atribútoch, čo je pochopiteľné lebo využíva dve matice atribútov naraz.

Potom sme vyskúšali aj embedded metódy: gain a split pre XGBoost a LGBM (popisovali sme ich pri gradient boosting), potom ohodnotenie atribútov pre CatBoost, RandomForestClassifier (RFC) s RegularizedForest (RGF). RFC používa mean decrease in impurity (MDI) [163], teda gain. 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 *l*1 a *l*2 regularizáciu pre lineárne SVC (SVM pre klasifikáciu), SGD SVM (SVM ktoré využíva stochastic gradient descent pri učení) a *l*2 regularizáciu pre lineárnu veziu kernelizovaného SVM. Pre SGD SVM sme použili aj elasticnet. Kvôli malej rýchlosti pre celý dataset vypadli: *l*2 regularizácia pre SVC (40 minút pre 92000 atribútov), CatBoost (43 minút pre 92000) a RGF (mal len 3 minúty, ale neskôr sme zmenili jeho parametre na vyššiu presnosť a stal sa mnohonásobne pomalším a pri teste mu došla pamäť). Blokové HSIC lasso zbehlo tiež, ale je to len aproximácia, a čím viac atribútov bolo tým väčšiu aproximáciu sme museli povoliť aby zbehlo. Pre všetky atribúty (483613) mu došla pamäť aj pri najväčšej miere aproximácie.

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 výsledky najlepšieho a najhoršieho stromu a SVM. Výsledky sú v nasledujúcej tabuľke (Tab. 1). Všetky metódy majú aspoň jeden výsledok nad 95% čo sme určili za prah a teda nechávame ich všetky. Jediné metódy ktoré sme vylúčili pre celý dataset sú tie, ktoré boli pomalé alebo pamäťovo náročné.

Tab. 1 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 gain | 98.4% | 98.9% | 95.4% | 98.5% |
| LGBM split | 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 gain | 98.3% | 98.9% | 95.4% | 98.2% |
| XGBoost split | 98.3% | 98.9% | 95.4% | 98.2% |
| pôvodne | 98.1% | 98.7% | 95.7% | 98.1% |

Na tretej skupine atribútov (766) sme zase vyskúšali pomalšie metódy. Tiež mali vybrať 10% (76) a na rovnakých klasifikáciách sme porovnali výsledky. Pre porovnanie sme pridali aj väčšinu rýchlych metód. Za prah sme určili 90%. Výsledky sú v tabuľke (Tab. 2). CFS sme vylúčili lebo 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. 2 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 gain | 97.8% | 98.4% | 86.9% | 95.3% |
| LGBM split | 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 gain | 97.7% | 98% | 87.3% | 94.7% |
| XGBoost split | 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 kleslo pod 90% pri stromoch a zároveň malo najhoršie skóre pri SVM čo je zaujímavé, lebo má rovnaký vzorec ako MRMR ktoré dosahuje omnoho lepšie výsledky. MRMR má len jeden rozdiel - dynamickú penaltu za redundanciu ktorá sa časom znižuje. To môže znamenať že MIFS malo kvôli svojej prísnosti nízke skóre pri tak malom počte atribútov, ale pri väčšom množstve má potenciál dosiahnuť omnoho lepšie výsledky, preto ho nechám a v celkových výsledkoch uvidíme či sa skóre zlepšilo pri viacerých atribútoch.

Zopár metód v knižnici sklearn sme pozmenili. CFS sa končil keď sa 5-krát po sebe nezlepšilo skóre. My sme ukončenie podmienili počtom atribútov ktorý sa zadáva ako nový parameter. Pri výpočte entropie sa používa aproximačná verzia a tak sa nám stávalo že atribút ju mal nulovú, lenže entropiou sa pri výpočte skóre delí. Tieto atribúty sme ignorovali (odchytili výnimku a nepridali ich). Pri gini score sme zmenili inicializované skóre z 0.5 – maximum pre dve triedy na 1 – globálne maximum. Pre laplacian score sme za defaultnú určili supervised verziu.

Samotnú selekciu robíme nasledovne: Zo všetkých atribútov vyberáme 1000 a 500 najlepších s metódami ktoré zvládnu tento počet atribútov. 1000 a 500 atribútov potom selektujeme aj z druhej a tretej skupiny, pokiaľ sú dosť veľké. Okrem toho na všetkých atribútoch spúšťame CFS ktoré by malo nájsť najmenšiu skupinu atribútov ktorá dobre delí dataset. Pre prvý dataset nám vyšlo 79 atribútov. //TODO doplni pre ostatné. 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 labels sú lepšie (klastrované alebo splitované) a na tých labels sme urobili selekciu a klasifikáciu. Pomalé metódy sme vynechali, lebo pre veľké datasety sú nepoužiteľné a preto je nepravdepodobné že by sa niekde pri veľkých datasetoch využívali.

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 dali 100 kôl a learning rate je 0.2. Pre CatBoost nemôžme zadávať obmedzenia keď pracujeme na CPU, len na GPU. Keďže vyžadujú Nvidia grafickú kartu a my máme AMD tak sme ho neobmedzovali. Následok bol dlhý čas behu. RGF má defaultne 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 defaultné 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, vtedy sa štandardizujú dáta už pred selekciou.

Po selekcii a klasifikácii spracúvame výsledky. Ukladáme si priemerné skóre po krosvalidácii, pre boosting algoritmy aj poradie najlepšieho kola aby sme vedeli posúdiť či došlo ku overfittingu. Ak najlepšie kolo bolo veľmi skoro, tak neskôr algoritmus overfittoval čo má za následok pokles skóre pri testovaní. Pôvodne, keď sme mali každý atribút označený unikátnym menom, sme hľadali pre každý algoritmus najväčší prienik s ostatnými prieniky sme ukladali. Teraz pre každú selekciu zisťujeme aké množstvo atribútov si zobrala z jednotlivých tried a koľko z nich patrilo do druhej a tretej skupiny.

Výber metód

Pri klasifikácii je výber metód dôležitejší, lebo výsledok každej metódy selekcie musí zbehnúť na všetkých metódach klasifikácie. Pri druhej skupine (5567 atribútov) sme za prah určili 98%, pri tretej (766 atribútov) 95%. CatBoost bol príliš pomalý, už pri tretej skupine bežal 21 minút, preto sme ho vylúčili.

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

Okrem LGBM forest sú najslabšie algoritmy konzistentné (v oboch datasetoch boli pod hranicou), preto ich vynecháme. Okrem toho vynecháme aj XGBoost histogram, ktorý teoreticky mal mať rýchlejšie trénovanie kvôli tvorbe histogramu atribútov ale u nás mal minimálne dvojnásobne dlhšie. Tiež vynecháme XGBoost forest kde sa paralelne boostuje viac stromov, lebo v oboch prípadoch mal rovnaký najlepší výsledok ako pri jednom strome, len v závislosti od počtu stromov stúpol čas tréningu. Vynechali sme aj LGBM random forest, 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. Keď sme robili klasifikáciu na celom datasete tak zo stromov ju zvládlo len RFC, tak sme ho nechali, nech máme stromový algoritmus na celý dataset. LGBM, LGBM GOSS aj LGBM RF skončili s chybou lebo im došla pamäť. XGBoost aj XGBoost RF nezbehli ani po niekoľkých hodinách. Keďže kernely sú pomalšie, tak na celý dataset sme použili len linearSVC(libSVM) a SGD. Len dve algoritmy sú málo vzhľadom na to, že SVM malo doteraz najlepší výsledok, preto sme nakoniec nechali aj linearSVC (liblinear) ale s l1 regularizáciou, lebo pôvodný s l2 regularizáciou bol veľmi pomalý.

Analýza výsledkov

Zatiaľ tu mám len otázky ktoré chcem zodpovedať keď budem mať všetky výsledky.

Prvý dataset

Mala druhá a tretia skupina atribútov porovnateľné výsledky klasifikácie so všetkými atribútmi? Vybral FCBF dobrý počet atribútov (pomer počtu a výsledku)? Aký počet atribútov mal najlepšie výsledky? Sú embedded metódy dobré len na vlastnom klasifikátore?

Ktoré metódy selekcie vylepšili pôvodné skóre? Ktoré z nich boli konzistentné (pre rôzne počty atribútov a rôzne metódy selekcie)? Do ktorej skupiny patrili (rýchle, pomalé, embedded)? Pre aký počet atribútov mali najlepšie skóre? Boli niektoré z týchto metód úspešnejšie pri jednom labelingu (len prvý dataset)? Aké atribúty vyberali? Ktoré skupiny atribútov sa vyskytli v najväčšom počte úspešných selekcií?

Druhý dataset

Rovnaké otázky ako v predošlom.

Tretí dataset

Rovnaké otázky ako v predošlom.

Globálne pozorovania

Boli rozdiely medzi najlepšími metódami selekcie medzi rôznymi datasetmi? Bol v každom datasete lepší labeling cez klastrovanie? Vyberali jednotlivé najlepšie selekcie atribúty z rovnakých skupín vo všetkých datasetoch? Ak nie, ktoré atribúty sa pridali medzi najlepšie a ktoré vypadli?

Záver

V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom.

Zoznam použitej literatúry

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. 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. 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. 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. 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.

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 score 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](https://www.virusradar.com/en/Win32_Chir.B/description 157)

[157](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](https://docs.scipy.org/doc/numpy/reference/generated/numpy.genfromtxt.html 160)

[160](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](https://docs.scipy.org/doc/numpy-1.10.0/user/basics.types.html 162)

[162](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.

Prílohy

1. CD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe, program na selekciu atribútov
2. Podobné práce
3. Nástroje na analýzu programov
4. Výsledky prvého datasetu

Príloha B – práce o klasifikácii malvéru

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| článok | dáta | interpretácia | metóda | úspešnosť |
| 2018-1 | byte sequences | image | CNN | 97.5 |
| 2018-2 | byte frequency, byte 4-grams, function names, assembly features, opcode frequency, count, asm file pixel density, segment count | n-gram, strings, obrázky | ensemble of DT | 0.0028 (log loss), 99.83 (accuracy) |
| 2018-3 | 4-gram of bytes | hash map - image | CNN | 99.0 (accuracy) |
| 2018-4 | n-gram of bytes | image | CNN | 69.1 |
| 2018-5 | n-gram of opcode, metadata, bytes | image, text | CNN, RNN (LSTM), Logistic Regression | 99.36 (accuracy) |
| 2018-6 | byte sequences | text | CNN (DRN) | 98.6 |
| 2018-7 | binaries | image | CNN | 94.0 |
| 2018-8 | byte sequences | stream of entropy | CNN, MRCNN | 98.96 |
| 2017-1 | function call graphs | graph feature vector | RF | 99.33 (accuracy) |
| 2017-2 | system, library call graphs | graph | graph uniqueness matching | 77.44 |
| 2017-3 | binaries | image | CNN | 98.0 |
| 2017-4 | binaries | image | CNN | 98.63 |
| 2017-5 | kombinácia statických a dynamických dát | n-grams | DT, KNN, LR, NB, RF, SVM | RF - 97 DT - 97, KNN - 73, LR - 71, SVM - 70, NB - 65 |
| 2017-6 | byte sequences | ? | CNN | 94.91 |
| 2017-7 | PE metadata, opcodes | n-grams | CNN + FFNN | 93.0 |
| 2016-2 | binaries | image | RF | 95.62 |
| 2016-3 | byte sequences | 4-grams | ensemble of DT, LR | 98.83 |
| 2016-4 | byte, asm sequences | strings | gene sequence alignment | 96-98 |
| 2016-5 | binaries | image | KNN, (A)NN, SVM | 96.6, 95.6, 94.6 |
| 2016-6 | kombinácia statických dát | obrázky, n-gramy, strings | ensemble of DT | 99.77 (accuracy), 0.0096 (log loss) |
| 2016-7 | opcode | n-grams | SVM, NB, DT, RF | 98 pre SVM - best |
| 2016-10 | system calls | vectors | CNN + RNN, FFNN, CNN | 89.4, 79.8, 89.2 |
| 2016-13 | kombinácia statických a dynamických dát | ? | Latent Dirichlet Allocation, Hierarchical Dirichlet Processes | 93.4, 90.5 |
| 2015-1 | instructions | control flow graph, vector of ranks | pageRank + RF | 72.0 |
| 2015-3 | binaries | image, graf entropie | similarity measurement for histogram comparison | 97.9 |
| 2014-1 | API dependecies | graph | NB + alg. Pre podobnosť grafov | 93.0 |
| 2014-3 | API call sequencies | strings | gene sequence alignment | ? |
| 2014-4 | opcode sequencies | image | vlastný alg. pre podobnosť okolia farebných obrázkov | 98.96 a 97.32 |
| 2014-7 | pakety | stavová matica | evolučné algoritmy - XCS, UCS, GAS, SLAVE aj neevolučné (DT, SVM, KNN, NB | 94.53, 99.7, 99.19, 84.91, a 99.42, 99.01, 99.55, 99.70 (accuracy) |
| 2014-8 | API calls | behaviour graph, modality vector | RF | 92.07 |
| 2013-1 | API call frequence | ? | RF | 66.8 |
| 2013-4 | binaries | sekvencie n-gramov | DT, SVM, FFNN | 91.25, 96.64, 88.31 |
| 2013-5 | API calls, binaries | n-grams, strings | LR, NN | 89.3, 91.5 |
| 2013-8 | binaries, opcodes (+freq), PE headers, exec trace | n-gram | NB, KNN, SVM, DT | mnoho rôznych výsledkov |
| 2013-10 | function call graphs | graph | KNN, SVM | 95.31, 90.86 |
| 2013-11 | binaries | n-gram | DT, FFNN, SVM | 91.25, 88.31, 96.64 |
| 2013-12 | kombinácia statických a dynamických dát | rôzne | SVM, KNN, DT, RF | 92.65, 94.67, 95.67, 97.05 |
| 2013-13 | inštrukcie, system calls | MIST | PHMM | 96.4 |
| 2013-14 | pcap | behaviour graph | DT | 94.57 |
| 2013-15 | API calls from DLLs | vector | FFNN | max avg. 95.36 |
| 2012-8 | statické aj dynamické dáta | rôzne | SVM, DT, RF, KNN | mnoho rôznych výsledkov |
| 2012-9 | rôzne dynamické dáta | vector | NB, RI, DT, KNN | 79.88, 91.30, 92.83, 95.30 |
| 2011-1 | opcode sequencies | n-gram | majú tzv. subfamily centroid vector pre rátanie podobnosti | 99.05 |
| 2011-3 | binaries | image-> GIST features | KNN | 99.2 (accuracy) |
| 2011-6 | API calls, frequencies | strings | SVM, KNN, DT, RF | 94.3, 90, 94.5, 94.5 |
| 2011-9 | instructions (MIST), system calls | vector | LSH, nearest prototype classification (alternatíva ku KNN) | 98.1 |
| 2011-10 | statické a dynamické dáta | image-> GIST features a JSON-> vector | KNN | 98, 95 |
| 2010-3 | dynamické dáta | vector | SVM | 83.3 |
| 2010-4 | system calls | graph | maximal common subgraph | rôzne, podľa rodiny |
| 2010-6 | API call sequencies | strings, frequencies | SVM (SMO), KNN, DT, RF | 94.5, 93.1, 92.5, 95.7 |
| 2010-8 | disassembled code, binaries | function length, strings (PSI) | NB, SVM, KNN, DT, RF | 94.31, 97.77, 97.63, 98.15, 96.69 |
| 2009-3 | PSI (strings) | binary vector | NB, SVM, KNN, DT, RF | 90.4, 95.5, 95.4, 95.9, 94.8 |
| 2009-6 | function call graphs | graph | KNN, grafové algoritmy | 91.3 |
| 2008-1 | rôzne dynamické dáta | vector | SVM | 88.0 |

Príloha C – nástroje na analýzu programov

|  |  |
| --- | --- |
| **Name** | **Data** |
| NTcore explorer suite | headers, imports, exports, dependencies, resources from PE file |
| pype32 | metadata,sections data from PE file. No documentation |
| pestr | printable strings from PE file |
| pefile | PE - metadata, resources, sections, strings, exports, imports |
| pescan and pescanner | Metadata from PE header |
| LIEF | PE - headers, imports, exports, sections, resources, overlay, metadata, symbols |
| pev | headers, sections, imports, exports |
| dumpbin | PE headers, sections,… raw data. |
| strings linux | strings for all types of files |
| strings - sysinternals | most common strings |
| IDA, ida python | Disassembled file |
| sigcheck | Entropy |
| radare, rabin | disassembler, many ouput types (rabin). |
| mastiff | static analysis, many plugins, reports |
| multiScanner | many moduls for data gathering, AV, sandboxes, reports |
| pyew | PE, ELF file analysis, strings, functions |
| generic parser | Sections, imports, exports, section entropy, size |
| peframe | Output from virustotal API plus suspicious features |
| objdump | Metadata, disassembling, debugging and more |
| exiftool | Metadata, many file formats |
| hexdump | hexadecimal file |

Príloha D – výsledky prvého datasetu

Tab. 1 klasifikácia troch skupín atribútov pre konsenzus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Počet atribútov | Stromy min. | Stromy max. | SVM min. | SVM max. |
| 486136 | 96.29% | 96.29% | 95.88% | 96.9% |
| 5567 | 95.98% | 98.96% | 95.78% | 98.14% |
| 766 | 94.43% | 98.46% | 88.55% | 95.17% |

Tab. 2 časy metód selekcie pri výbere 500 atribútov z 486136

|  |  |
| --- | --- |
| Metóda | Čas |
| Chi-square | 0:00:02.469151 |
| Mutual information | 0:04:31.192547 |
| ANOVA F-score | 0:00:04.812868 |
| Gini score | 0:30:33.937980 |
| Fisher | 0:00:25.455066 |
| Laplacian | 0:04:30.895651 |
| SPEC | 0:38:25.074741 |
| Relieff | 0:11:54.963856 |
| XGBoost gain | 0:18:41.851148 |
| XGBoost split | 0:18:33.153848 |
| LGBM split | 0:18:33.181851 |
| LGBM gain | 0:19:08.409007 |
| RFC | 0:00:09.469471 |
| SVC L1 | 0:03:16.405594 |
| SGD L1 | 0:04:12.613009 |
| SGD L2 | 0:00:47.244694 |
| SGD elasticnet | 0:02:44.965700 |
| SVM linear | 0:05:44.800013 |

Tab. 3 časy metód selekcie pri výbere 1000 atribútov z 5567

|  |  |
| --- | --- |
| Metóda | Čas |
| MIFS | 4:09:29.529556 |
| mRMR | 4:32:43.128295 |
| CIFE | 4:11:28.817448 |
| JMI | 5:18:14.093954 |
| CMIM | 4:30:27.683071 |
| DISR | 14:34:12.942039 |
| Trace ratio | 0:00:01.734508 |
| CatBoost | 0:07:18.986581 |
| RGF | 0:01:32.694567 |
| Chi-square | 0:00:00.015627 |
| Mutual information | 0:00:03.234622 |
| ANOVA F-score | 0:00:00.046879 |
| Gini score | 0:01:53.289886 |
| Fisher | 0:00:00.328141 |
| Laplacian | 0:00:03.000219 |
| SPEC | 0:00:25.970721 |
| Relieff | 0:00:10.657062 |
| XGBoost gain | 0:00:14.079189 |
| XGBoost split | 0:00:14.704237 |
| LGBM split | 0:00:15.766818 |
| LGBM gain | 0:00:15.766818 |
| RFC | 0:00:01.375096 |
| SVC L1 | 0:00:07.344310 |
| SGD L1 | 0:00:00.671925 |
| SGD L2 | 0:00:00.125001 |
| SGD elasticnet | 0:00:00.453150 |
| SVM linear | 0:00:01.765759 |
| SVC L2 | 0:01:46.148716 |

Tab. 4 porovnanie presnosti pre klastrovanie a konsenzus na dátach bez outlierov

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Počet atribútov | Stromy min. | Stromy max. | SVM min. | SVM max. |
| Klastrovanie |  |  |  |  |
| 486136 | 94.95% | 94.95% | 96.71% | 98.36% |
| 5567 | 95.05% | 99.45% | 96.08% | 98.46% |
| 766 | 94.3% | 98.9% | 90.12% | 96.5% |
| Konsenzus: |  |  |  |  |
| 486136 | 95.64% | 95.64% | 95.62% | 96.63% |
| 5567 | 95.53% | 99.01% | 95.75% | 97.93% |
| 766 | 94.1% | 98.56% | 88.81% | 95.52% |

Nasledujúce tabuľky sú selekcie pre konsenzus labeling.

Tab. 5 selekcia 79 atribútov z 486136

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metóda | Stromy min. | Stromy max. | SVM min. | SVM max. |
| ANOVA\_F-score | 32.89% | 40.36% | 39.53% | 40.25% |
| chi-square | 94.54% | 97.82% | 84.31% | 92.56% |
| FCBF | 96.6% | 98.66% | 94.22% | 98.26% |
| Fisher | 32.89% | 40.25% | 39.53% | 40.25% |
| Gini | 20.64% | 30.97% | 30.97% | 30.97% |
| Laplacian | 80.51% | 92.16% | 36.63% | 60.51% |
| LGBM gain | 97.32% | 99.27% | 95.03% | 98.55% |
| LGBM split | 97.74% | 98.95% | 92.46% | 96.79% |
| mutual\_info | 91.11% | 95.76% | 58.02% | 86.98% |
| ReliefF | 84.52% | 93.27% | 18.88% | 67.27% |
| RFC | 90.2% | 92.78% | 83.47% | 88.43% |
| SPEC | 76.8% | 90.29% | 48.74% | 72.66% |
| XGBoost gain | 96.8% | 99.06% | 95.66% | 97.22% |
| XGBoost split | 97.11% | 99.07% | 94.84% | 98.46% |
| SGD\_elasticnet | 77.5% | 90.41% | 69.61% | 80.6% |
| SGD\_L1 | 86.79% | 94.62% | 84.94% | 94.6% |
| SGD\_L2 | 50.02% | 53.36% | 36.12% | 52.19% |
| SVC\_L1 | 95.02% | 98.34% | 91.96% | 95.48% |
| SVM | 86.77% | 94% | 28.24% | 67.06% |
| pôvodne | 96.29% | 96.29% | 95.88% | 96.9% |

Tab. 6 selekcia 500 atribútov z 486136

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metóda | Stromy min. | Stromy max. | SVM min. | SVM max. |
| ANOVA\_F-score | 51.33 | 90.15 | 41.38 | 53.27 |
| chi-square | 96.5 | 98.34 | 85.24 | 97.4 |
| Fisher | 50.11 | 89.96 | 41.38 | 53.27 |
| Gini | 76.08 | 94.82 | 59.7 | 78.95 |
| Laplacian | 56.02 | 92.35 | 36.22 | 61.86 |
| LGBM gain | 97.73 | 99.28 | 92.57 | 99.07 |
| LGBM split | 97.84 | 99.37 | 92.02 | 98.65 |
| mutual\_info | 92.26 | 96.99 | 60.23 | 94.43 |
| ReliefF | 90.41 | 96.79 | 37.04 | 86.06 |
| RFC | 97.11 | 98.76 | 90.9 | 96.91 |
| SPEC | 87.85 | 94.1 | 55.77 | 88.95 |
| XGBoost gain | 97.73 | 99.48 | 93.7 | 99.28 |
| XGBoost split | 97.73 | 99.48 | 94.01 | 99.28 |
| SGD\_elasticnet | 94.55 | 98.65 | 95.46 | 98.25 |
| SGD\_L1 | 96.18 | 98.86 | 96.18 | 98.97 |
| SGD\_L2 | 95.38 | 98.75 | 95.38 | 98.65 |
| SVC\_L1 | 97.21 | 98.64 | 96.91 | 99.48 |
| SVM | 92.78 | 97.21 | 56.17 | 91.11 |
| pôvodne | 96.29% | 96.29% | 95.88% | 96.9% |

Tab. 7 selekcia 1000 atribútov z 486136

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metóda | Stromy min. | Stromy max. | SVM min. | SVM max. |
| ANOVA\_F-score | 75.47 | 94.61 | 42.52 | 80.74 |
| chi-square | 96.70 | 98.44 | 83.48 | 97.73 |
| Fisher | 76.09 | 94.51 | 42.52 | 80.74 |
| Gini | 94.46 | 98.44 | 68.47 | 96.29 |
| Laplacian | 59.06 | 92.65 | 26.53 | 65.9 |
| LGBM gain | 97.63 | 99.26 | 93.08 | 99.06 |
| LGBM split | 97.42 | 99.16 | 94.02 | 98.66 |
| mutual\_info | 92.16 | 97.31 | 57.3 | 94.45 |
| ReliefF | 92.68 | 97.32 | 58.72 | 94.94 |
| RFC | 96.6 | 98.85 | 91.62 | 98.05 |
| SPEC | 90.94 | 95.86 | 56.68 | 90.49 |
| XGBoost gain | 97.63 | 99.48 | 95.05 | 99.28 |
| XGBoost split | 97.63 | 99.48 | 94.84 | 99.28 |
| SGD\_elasticnet | 96.81 | 98.65 | 97.12 | 98.96 |
| SGD\_L1 | 96.49 | 98.66 | 97.52 | 98.86 |
| SGD\_L2 | 96.7 | 98.65 | 97.12 | 98.96 |
| SVC\_L1 | 96.6 | 99.27 | 96.91 | 99.58 |
| SVM | 93.41 | 98.13 | 78.86 | 96.9 |
| pôvodne | 96.29% | 96.29% | 95.88% | 96.9% |

Tab. selekcia 79 atribútov z 5567

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metóda | Stromy min. | Stromy max. | SVM min. | SVM max. |
| ANOVA\_F-score | 5839 | 9037 | 42.52 | 80.74 |
| CatBoost | 9659 | 9885 |  |  |
| chi-square | 937 | 974 | 83.48 | 97.73 |
| CIFE | 9041 | 9617 |  |  |
| CMIM | 8647 | 9463 |  |  |
| DISR | 9504 | 9844 |  |  |
| Fisher | 5807 | 9036 | 42.52 | 80.74 |
| Gini | 7793 | 9513 !!!dokonč | 68.47 | 96.29 |
| Laplacian | 59.06 | 92.65 | 26.53 | 65.9 |
| LGBM gain | 97.63 | 99.26 | 93.08 | 99.06 |
| LGBM split | 97.42 | 99.16 | 94.02 | 98.66 |
| mutual\_info | 92.16 | 97.31 | 57.3 | 94.45 |
| ReliefF | 92.68 | 97.32 | 58.72 | 94.94 |
| RFC | 96.6 | 98.85 | 91.62 | 98.05 |
| SPEC | 90.94 | 95.86 | 56.68 | 90.49 |
| XGBoost gain | 97.63 | 99.48 | 95.05 | 99.28 |
| XGBoost split | 97.63 | 99.48 | 94.84 | 99.28 |
| SGD\_elasticnet | 96.81 | 98.65 | 97.12 | 98.96 |
| SGD\_L1 | 96.49 | 98.66 | 97.52 | 98.86 |
| SGD\_L2 | 96.7 | 98.65 | 97.12 | 98.96 |
| SVC\_L1 | 96.6 | 99.27 | 96.91 | 99.58 |
| SVM | 93.41 | 98.13 | 78.86 | 96.9 |
| pôvodne | 96.29% | 96.29% | 95.88% | 96.9% |