UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZARE: INFORMATICĂ

Lucrare de Licenţă

COORDONATOR:
Prof. Dr. Viorel Negru
Drd. Mario Reja

STUDENT:
Tufiş-Schwartz
Alexandru-Sebastian

TIMIȘOARA
2018

UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZARE: INFORMATICĂ

DETECTAREA APLICAŢIILOR MALIŢIOASE FOLOSIND METODE DE ÎNVĂŢARE AUTOMATĂ

(MALWARE DETECTION USING MACHINE LEARNING)

COORDONATOR:
Prof. Dr. Viorel Negru
Drd. Mario Reja

STUDENT:
Tufiş-Schwartz
Alexandru-Sebastian

TIMIȘOARA
2018

Rezumat

Scopul lucrării este de a prezenta o abordare a diferițiilor algoritmi de "machine learning" pentru a depista dacă un fișier este infectat sau curat. În această lucrare ne vom axa pe tipul de fișiere executabile ale sistemului de operare Windows. Având în vedere creșterea abundență a malware-urilor avem nevoie de diferite abordări automate pentru depistarea acestor fișiere infectate.

În acest proiect vom studia și implementa un script de extragere a datelor esențiale dintr-un executabil pentru a ne crea o baza de date cu fișier maliționase și curate, pe care ne vom putea antrena algoritmii de machine learning. Algorithmii aleși pentru acest proiect sunt: K-NN, XGBoost și Random Forest.

In ultimul capitol algoritmii sunt testati pe intreg setul de date care contine 54 de caracteristici, acestea avand o acuratete de peste 90%. Aplicarea unui algoritm de feature selection a mai crescut acest procent in cazul tuturor algoritmilor de learning.

Rezumat

The scope of this paper is to present a malware detection approach using machine learning.In this paper we will focus on windows executable files. Because of the abnormal growth of these malicious software's ne need to use different automated approaches to find theses infected files.

In this project we are going to study and implement a script used for data extraction from the PE-files to create a data set with infected and clean files, on which we are gonna train our machine learning algorithms:K-nn, XGBoost and Random Forest.

The last chapter of this paper the algorithms are tested with all the data set features. The accuracy of all algorithms is over 90%. After applying a Feature selection algorithm over the data set, the accuracy has been improved for all the learning algorithms.

Cuprins

Introducere

Lucrarea de față își propune să prezinte funcționalitatea și acuratețea a trei algoritmi diferiți de machine learning pentru a depista dacă un executabil este infestat sau este curat.

În primul capitol va fi prezentată o descriere a fenomenului de Malware, programe software sau bucăți de cod care au ca scop acapararea sistemelor informatice pentu a fura informații sau pentru distrugerea acestora. Vom aprofunda acest subiect pentu a avea o oarecare înțelegere a acestor programe malițioase. După o scurtă introducere a acestui fenomen vom prezența evoluția malwareurilor de-a lungul timpului. Urmează prezentarea diferitelor tehnici de protecție.

În cel de-al doilea capitol va fi introdus domeniul de Machine Learning și beneficiile acestuia.Mai departe vom discuta despre importanța machine learningului în abordarea acestei situații. Vor fi prezentați algoritmii folosiți în această lucrare cât și beneficiile acestora. Machine learning este foarte folosit în acest domeniu de către programele antivirus și antimalware cât și de către aceste programe malițioase, de exemplu Polimorfic Malware folosește algorithmi de machine learning pentru a se encripta într-un mod diferit de fiecare dată cînd infestează o mașină nouă, devenind din ce in ce mai greu de depistat.

Ultimul capitol se ocupă cu prezentarea în detaliu a algoritmiilor K-NN, XGBoost și Random Forest. Implementarea acestor algoritmi au fost realizată cu ajutorul limbajul de programare Python folosind scikit-learn. Scikit-learn este un software gra-

tuit care conține librarii de machine learning pentru Python. Acesta dispune de diferiți algoritmi de clasificare regresie și clustering și este proiectat să interacționeze cu bibliotecile numerice și stintifiice NumPY și SciPi.

Capitolul 1

Malware

1.1 Definitie

"Malware"[1] este prescurtare pentru "malicious software"(programe malițioase), este folosit ca un singur termen pentru a se referi la Viruși, Cai Troieni, Viermi etc. Aceste programe au o varietate de funcționalități cum ar fi furtul, criptarea sau ștergerea datelor sensibile, modificarea sau deturnarea funcțiilor de bază ale computerelor și monitorizarea activității computerului fără permisiunea utilizatorilor.

1.2 Tipuri de Malware

Virus informatic

Este în general un program care se instalează fără voia utilizatorului și poate provoca pagube atât sistemului de operare cât și elementelor hardware(fizice) ale unei mașini de calcul.

Efecte generate de viruși:

- distrugerea unor fișiere.
- modificarea dimensiunii fișierelor.
- ștergerea totală a informațiilor de pe disc, inclusiv formatarea acestuia.

- distrugerea tabelei de alocare a fișierelor, care duce la imposibilitatea citirii informațiilor de pe disc.
- diverse efecte grafice/sonore inofensive, dar deranjante
- incetinirea vitezei de lucru a calcuatorului pana la blocare

Viermi informatici (Worms)

Viermii informatici sunt programe cu efecte distructive ce utilizează comunicarea între computere pentru a se răspândi. Viermi au trăsături comune cu virușii. Viermii sunt capabili să se multiplice asemenea virușilor, însă nu local, ci pe alte calculatoare. Folosesc rețelele de calculatoare pentru a se răspândi pe alte sisteme. Tipuri de viermi informatici:

- Viermi de E-Mail
- Viermi de mesagerie instantanee
- Viermi de internet
- Viermi de IRC
- Viermi de fișiere partajate în rețea

Cai troieni

Caii troieni sunt programe „deghizate" ce încearcă să creeze breşe în sistemul de operare pentru a permite unui utilizator accesul în sistem. Troienii nu au facilitatea de a se auto-multiplica precum viruşii informatici.

Caii troieni se pot împărţi în mai multe categorii:

- backdoors: permite atacatorului să preia controlul asupra calculatorului victimă prin Internet;
- password stealer: programe ce fură parole (citesc datele de la tastatură şi le stochează în fişiere ce pot fi citite ulterior de către atacator sau pot fi trimise direct către contul de e-mail al atacatorului);

- logical bombs: când sunt întrunite anumite condiţii aceşti troieni pot efectua operaţii ce compromit securitatea sistemului;
- Denial of Service tools: programe ce trimit anumite secvenţe de date către o ţintă (de obicei un site web), cu intenţia de a întrerupe serviciile de Internet ale acelei ţinte.

Ransomware

Ransomware-ul este un tip de malware care blochează accesul victimei la calculator şi cere plata unei recompense. Recompensa şi motivul oficial, pentru care victima ar trebui să plătească, depinde de tipul virusului. Unele versiuni de ransomware pretind că plata ar trebui efectuată pentru a evita pedepsirea de către o autoritate guvernamentală (de obicei, FBI sau o agenţie locală), alţii informează că acesta este singurul mod de a decripta datele criptate.

Efecte generate de ransomware:

- sunt capabili să cripteze datele sensitive ale utilizatorilor
- pot şterge documentele predeterminate, obiectele multimedia şi orice alte fişiere care conţin informaţii importante. De asemenea, pot încerca să şteargă componente esenţiale din sistem sau părţi importante dintr-un alt software.
- Ameninţările ransomware pot fi utilizate pentru a fura nume de autentificare, parole, documente personale valoroase, date despre identitate şi alte informaţii confidenţiale
- pot termina rapid activitatea unui anti virus, anti-spyware sau orice alt software de securitate, blocându-i procesele şi dezactivând serviciile esenţiale din sistem.

root kit

Un software de tip rootkit este in general un program ce reuseste printr-o vulnerabilitate a sistemului gazda sa capete drepturi depline pe un sistem, sistem pe care il modifica pentru a-i putea folosi resursele nedetectat.

- poate modifica utilitarul "ps" de pe un sistem Linux, utilitar care afiseaza procesele active, pentru a-l face sa NU afiseze si procesul rootkitului
- poate ascunde anumite fișiere (proprii în general) pentru cazul în care un program tip antivirus scanează sistemul în căutarea să

spyware

Spyware este o categorie de ameninţări cibernetice, ce descrie programele maliţioase create pentru a infecta sistemele PC-urilor după care să iniţieze activităţi ilegale în acestea. În majoritatea cazurilor, funcţionalitatea acestor ameninţări depinde de intenţiile furnizorilor lor: unele părţi din ameninţările spyware pot fi folosite pentru a colecta informaţii personale (nume de autentificare, parole şi alte date personale identificabile) şi să le trimită proprietarilor prin conexiuni de internet ascunse, în timp ce alţi viruşi de tip spyware îşi pot urmări victimele şi colectează informaţii despre obiceiurile lor de navigare. Acestea sunt folosite pentru a urmări oamenii şi să le înregistreze cele mai vizitate website-uri precum şi acţiunile luate atunci când au fost vizitate. Această informaţie, în general, este utilizată de către diverse terţe în scopuri de marketing şi promovare, deci spyware-urile pot conduce şi la mărirea numărului de spam-uri.

Pentru ce poate fi utilizat un malware de tip spyware:

- Pentru a fura informaţii sensibile. Astfel de programe sunt interesate de informaţiile personale, precum autentificări, parole, date bancare şi alte informaţii similare. În plus, pot

monitoriza activitatea online a utilizatorului, să îi urmărească obiceiurile de navigare pe web şi să trimită toate aceste date pe un server de la distanţă.

- Să afişeze reclame nedorite. Spyware-ul poate afişa un număr mare de reclame de tip pop-up enervante. O astfel de activitate este mai mult asociată paraziţilor de tip adware.

- Redirecţionarea utilizatorilor către website-uri chestionabile sau maliţioase contrar dorinţei lor. În plus, unele tipuri de ameninţări spyware sunt capabile să modifice setările browserului web şi să modifice motorul de căutare şi pagina de start.

- Să creeze numeroase link-uri în rezultatele căutărilor efectuate de victimă şi să îl/o redirecţioneze către locurile dorite (site-uri spyware terţe, website-uri şi alte domenii asociate).

- Să cauzeze modificări esenţiale în setările sistemului. Aceste modificări pot diminua securitatea generală şi pot iniţia probleme legate de performanţă.

- Conectarea la un calculator compromis utilizând backdoors. Majoritatea ameninţărilor spyware sunt capabile să ofere hackerilor acces de la distanţă în sistem fără ştirea utilizatorului.

- Degradarea performanţei generale a sistemului şi cauzarea instabilităţii acestuia.

1.3 Istorie

Primele versiuni de Malware erau primitive, acestea infestau diferite masini prin intermediul floppy discurilor. Odata cu evo-

lutia Networking-ului si maturizarea internetului autorii de malware si-au adaptat codurile malitioase pentru a profita in intregime de de aceast mediu nou de comunicare. Mai jos o scurtă prezentare [2] a evoluţiei malware de-a lungul timpului.

1.3.1 Anii 1971-1999

- 1971-Creeper: Un experiment conceput pentru a testa modul în care un program se poate deplasa între computere.
- 1974-Wabbit: Un program care se multiplica pe sine însuşi la un pas accelerat, până când scade viteza sistemul în așa măsură încât performanța sistemului este redusă și eventual se prăbușește.
- 1982-Elk Cloner: Scris de către un copil de 15 ani, Elk Cloner este unul dintre primii viruși foarte răspândiți, care se multiplica pe sine însuşi si afiseaza o scurta "poezie" persoanei infectate: "It will get on all your disks; It will infiltrate your chips; Yes, it's Cloner!"
- 1986-Brain Boot Sector Virus: Considerat ca fiind primul virus care infectează computerele MS-DOS.
- 1986—PC-Write Trojan: Autorii malware au deghizat unul dintre cei mai vechi troieni ca un program popular numit "PC-Writer". Odată ajuns pe un sistem, acesta șterge toate fișierele unui utilizator.
- 1988—Morris Worm: a infectat un procent substanțial de computere conectate la ARPANET, predecesorul internetului, care a adus la inghenunchierea retelei în 24 de ore. acest Vierme a marcat un nou început pentru software-ul rău intenționat.

- 1991—Michelangelo Virus: virusul a fost conceput pentru a șterge informațiile de pe hard-discuri pe data de 6 martie, ziua de naștere a renumitului artist renascentist.

- 1999 — Melissa Virus: a folosit adresele Outlook din mașinile infectate și sa trimis la 50 de persoane deodata.

1.3.2 Anii 2000-2010

- 2000–ILOVEYOU Worm: viermele a infectat aproximativ 50 de milioane de computere. Daunele au provocat corporații majore și organelor guvernamentale, inclusiv porțiuni ale Pentagonului și Parlamentului britanic, să-și închidă serverele de e-mail. Viermii s-au răspândit la nivel global și au costat mai mult de 5,5 miliarde de dolari ca daune.

- 2003–SQL Slammer Worm: Unul dintre cei mai rapizi viermi de răspândire din toate timpurile, SQL Slammer a infectat aproape 75.000 de computere în zece minute. Viermele a avut un efect major la nivel mondial, încetinind traficul Internet în întreaga lume prin negarea serviciilor(denial of service).

- 2004–Cabir Virus: Deși acest virus a cauzat puține daune, este demn de remarcat pentru că este recunoscut pe scară largă ca primul virus de telefon mobil.

- 2005–Koobface Virus: Una dintre primele cazuri de malware care infectează PC-urile și apoi se propagă pe site-uri de socializare. Dacă rearanjați literele din "Koobface" veți obține "Facebook". Virusul a atacat, de asemenea, alte rețele sociale precum MySpace și Twitter.

- 2008–Conficker Worm: O combinație a cuvintelor "configure" și "ficker", acest vierme sofisticat a provocat unele

dintre cele mai grave prejudicii observate de când Slammer a apărut în 2003.

1.3.3 2010-prezent

- 2010–Stuxnet Worm: La scurt timp după lansarea sa, analiștii de securitate au speculat deschis că acest cod malware a fost conceput cu scopul explicit de a ataca programul nuclear al Iranului și a inclus capacitatea de a afecta hardware-ul și software-ul. Viermele incredibil de sofisticat este considerat a fi o lucrare a unei întregi echipe de dezvoltatori, făcând-o una dintre cele mai intensive resurse de malware create până în prezent.

- 2011—Zeus Trojan: Deși a fost detectat pentru prima oară în 2007, autorul troianului Zeus a lansat codul publicul în 2011, oferind o viața nouă malware-ului. Uneori numit Zbot, acest troian a devenit una dintre cele mai de succes bucăți de software botnet din lume, cu impact asupra a milioane de mașini.

- 2013–Cryptolocker: a avut un impact semnificativ la nivel global și a contribuit la alimentarea erei ransomware.

- 2014–Backoff: Malware conceput pentru a compromite sistemele Point-of-Sale (POS) pentru a fura datele de pe cardul de credit.

- 2016–Cerber: Unaul dintre cele mai prolifice amenințări cripto-malware. La un moment dat, Microsoft a găsit mai multe PC-uri ale companie infectate cu Cerber decât orice altă familie de ransomware.

- 2017–WannaCry Ransomware: Exploatând o vulnerabilitate descoperită mai întâi de Agenția Națională de Securitate, WannaCry Ransomware a ingenunchiat un numar

mare de sisteme din Rusia, China, Marea Britanie și Statele Unite, blocând accesul la date și cerând o răscumpărare sau piardeau totul. Virusul a afectat cel puțin 150 de țări, inclusiv spitale, bănci, companii de telecomunicații, depozite și multe alte industrii.

1.4 Tehnici de analiza Malware

Analiza malware este necesară pentru dezvoltarea unor tehnici eficiente de detectare a fiserelor infestate. Această analiză reprezintă procesul de observare a scopului și a funcționalității unui program malware. Există 3 tehnici de analiză care au același scop: de a explică cum funcționează un malware și care sunt efectele acestuia asupra sistemului, dar timpul și cunoștiintele necesare sunt foarte diferit.

1.4.1 Analiza statica

Se mai numește și analiză de cod[3]. Adică codul software de malware este observat pentru a obține cunoaștinte despre funcționarea funcțiilor malware. Această tehnică de inginerie inversă este efectuată prin utilizarea instrumentelor de dezasamblare, decompilare, depanatoare si a instrumentelor de analiza a codului sursa.

1.4.2 Analiza dinamica

Se mai numește și analiză de comportament[3]. fișierele infectate sunt analizate în timpul execuției într-un mediu izolat cum ar fi o mașină virtuală, similator sau emulator. După executarea fișierului comportamentul și efectele acestuia asupra sistemului sunt monitorizate.

1.4.3 Analiza hibrid

Această tehnică este propusă pentru a depăși limitatiile analizei statice și dinamice. În primul rând analizează specificația semnăturii pentru orice cod malware și apoi o combină cu ceilalți parametri de comportament pentru îmbunătățirea analizei complete a programelor malware. Datorită acestei abordări, analiză hibrid[4] depășește limitele analizelor statice și dinamice

1.5 Tehnici de Detectie Malware

Tehniciile de detecție malware sunt folosite pentru a detecta malware și a preveni infestarea sistemului, protejându-l de potențiale pierderi de informații și compromiterea sistemului. Ele se categorizeaza în: detectare de semnături, detectare de comportament și detectare de caracteristici.

1.5.1 Detectare de semnaturi

Detectarea bazată pe semnături[5] este un proces în care se stabilește un identificator unic despre o amenințare cunoscută, astfel încât amenințarea să poată fi identificată în viitor. În cazul unui scanări de virusi, acesta poate fi un model unic de cod care se atașează la un fișier sau poate fi la fel de simplu ca și hash-ul unui fișier rău cunoscut. În cazul în care acel tipar specific sau semnătură este descoperit din nou, fișierul poate fi semnalat ca fiind infectat.

Deoarece malware-ul a devenit mai sofisticat, autorii malware au început să folosească noi tehnici, cum ar fi polimorfismul, pentru a schimba modelul de fiecare dată când obiectul sa răspândit de la un sistem la altul. Ca atare, o potrivire simplă a modelului nu ar fi utilă dincolo de o "mână mică" de dispozitive descoperite

1.5.2 Detectare de comportament

Spre deosebire de scanarea bazată pe semnături, care arată că se potrivesc semnăturile găsite în fișiere cu cea a unei baze de date cu malware cunoscut, scanarea euristică[5] utilizează reguli și/sau algoritmi pentru a căuta comenzi care pot indica intenții rele. Folosind această metodă, unele metode euristice de scanare sunt capabile să detecteze malware fără a avea nevoie de o semnătură. Acesta este motivul pentru majoritatea programelor antivirus utilizează în combinație atât metode de semnătură, cât și metode euristice, pentru a captura orice malware care ar putea încerca să se sustragă detectării.

1.5.3 Detectare de caracteristici

Detectarea de caracteristici este o derivată a detectării bazate pe comportament care încearcă să depășească rată tipică de alarme false asociată cu această. Detectarea pe baza de caracteristici se bazează pe caracteristicile programului care descriu comportamentul destinat securității programelor critice. Această implică monitorizare execuțiilor programului și detectarea deviațiilor de la specificații în comportamentul acestuia, în loc de a detecta aparență tiparelor specifice de atac. Această tehnică este similară cu detectarea anomaliilor, diferența fiind că această se bazează pe caracteristici dezvoltate manual pentru a captura comportamentul sistemului în locul bazarii pe technici de machine learning. Avantajul acestei tehnici este că poate detecta instanțe cunoscute și necunoscute de malware, iar nivelul pozitivelor false este mai mic, dar nivelul negativelor false este înalt și nu la fel de eficient că detectarea pe baza de comportamet.

Capitolul 2

Machine learning

2.1 Definitie

Machine Learning[6] este o categorie de algoritmi care permit aplicațiilor soft să prezică mult mai bine rezultate fără a fi specific programate. Premisa de bază a machine learnigului este de a construi algoritmi care primesc date de intrare și se folosesc de analiză statistică pentru a prezice date de ieșire în timp ce datele de ieșire sunt actualizate precum mai multe date de intrare devin valabile.

Procesele implicate de machine learning sunt similare cu procesele de data mining[7] și modelare predictivă. Ambele necesită căutare unor anumite tipare prin dată, și ajustarea acțiunilor programului în mod corespunzător. Mulți oameni sunt familiarizați cu machine learningul din shoppingul pe internet și din reclamele care le sunt arătate în funcție de ce cumpără. Aceasta se întâmplă din cauza că motoarele de recomandare folosesc machine learning pentru a personaliza reclamele care sunt livrate online, aproape în timp real. Pe deasupra marketingului personalizat, alte cazuri cunoscute în care este folosit machine learningul este detectarea de fraude, filtrarea de spam, descoperirea amenințărilor în rețea, mentenanță predictivă și construirea fluxului de noutăți.

2.2 Cum funcționează machine learning

Algoritmii de machine learning sunt categorizați ca supervizați[8] și nesupervizați[8]

- Algoritmii supervizați necesită un cercetător de date, sau analist de date, care posedă cunoștințe de machine learning pentru a aproviziona datele de intrare, și de ieșire, dorite, pe lângă livrarea de feedback despre acuratețea predicțiilor făcute în timpul antrenării algoritmului. Cercetătorii de date determină care variabile, sau caracteristici, ar trebui să fie analizate de model și folosite pentru a dezvolta predicții. Odată ce antrenamentul este complet, algoritmul va aplica ce a învățat asupra unor date noi.

- Algoritmii nesupervizați nu au nevoie de antrenament cu datele de ieșire. În schimb, ele folosesc o metodă numită deep learning pentru a revizui data și a ajunge la concluzii. Algoritmii nesupravegheați de învățare, cunoscuți ca și rețele neurale, sunt folosite pentru procese mai complexe decât algoritmii supravegheați, care includ recunoașterea imaginilor, speech-to-text și generare natural a limbii. Aceste rețele neurale funcționează prim combinarea milioanelor de exemple de antrenament cu date și a identifica automat corelații subtile dintre multiple variabile. Odată antrenat, algoritmul se poate folosi de asociațile făcut pentru a interpreta date noi. Acești algoritmi devin realizabili doar în era cu informații mari, deoarece necesită cantități masive de date pentru a se antrena.

2.3 Algoritmi folositi in aceasta lucrare

2.3.1 K-NN (k-Nearest Neighbors)

KNN[9] este un algoritm de învățare supervizată bazat pe asocieri care nu necesită o etapă de antrenare propriu-zisă. Se bazează pe învățarea prin analogie și stabilește clasa corespunzătoare unui exemplu de testare pe baza similarității acestuia cu k exemple, cele mai similare, din setul de date de antrenament. Cele k exemple luate în considerare vor stabili clasa exemplului de test pe baza votului majoritar. Fiecare exemplu de antrenament este un vector în spațiul de reprezentare al datelor și are asignat o singură etichetă. Etapa de antrenare pentru algoritmul KNN constă doar în memorarea vectorilor de trăsături și a etichetelor corespunzătoare claselor pentru exemplele de antrenament. În faza de clasificare propriu-zisă (în etapa de testare), la un element din setul de testare îi atribuim clasa corespunzătoare ca fiind cea mai frecventă clasă dintre clasele celor k exemple de antrenament, cele mai apropiate de exemplul de testare. Parametrul k este o constată specificată de utilizator și de obicei are o valoare mică. Cea mai bună alegere a lui k depinde de date; în general, o valoare mare pentru k va reduce influența zgomotului asupra clasificării, dar va face ca zonele de separare dintre clase să fie mai puțin distincte.

Pașii algoritmului KNN:

1. Se stabilește valoarea lui k în raport cu numărul de exemple de antrenament pe care le avem la dispoziție.

2. Pentru fiecare exemplu din setul de testare se stabilește clasa acestuia astfel:

3. Se calculează similaritatea dintre exemplul de testare și toate exemplele avute în setul de antrenare. Pentru calculul similarității se pot folosi oricare dintre metricile de

similaritate descrise mai jos.

4. Se iau primele k exemple dintre cele de antrenare care sunt cele mai similare cu exemplul curent de testare și pe baza lor se stabilește clasa exemplului de testare folosind votul majoritar.

5. Se verifică dacă clasificarea este sau nu corectă pe baza informațiilor deținute în fișierul de testare.

6. Atâta timp cât mai sunt exemple de testare se reia de la pasul 3.

7. Se evaluează calitatea clasificării pentru valoarea lui k curenta, folosind metricile externe de evaluare a algoritmilor de învățare cum ar fi acuratețea de clasificare, precizia, recall, true negative rate etc. . .

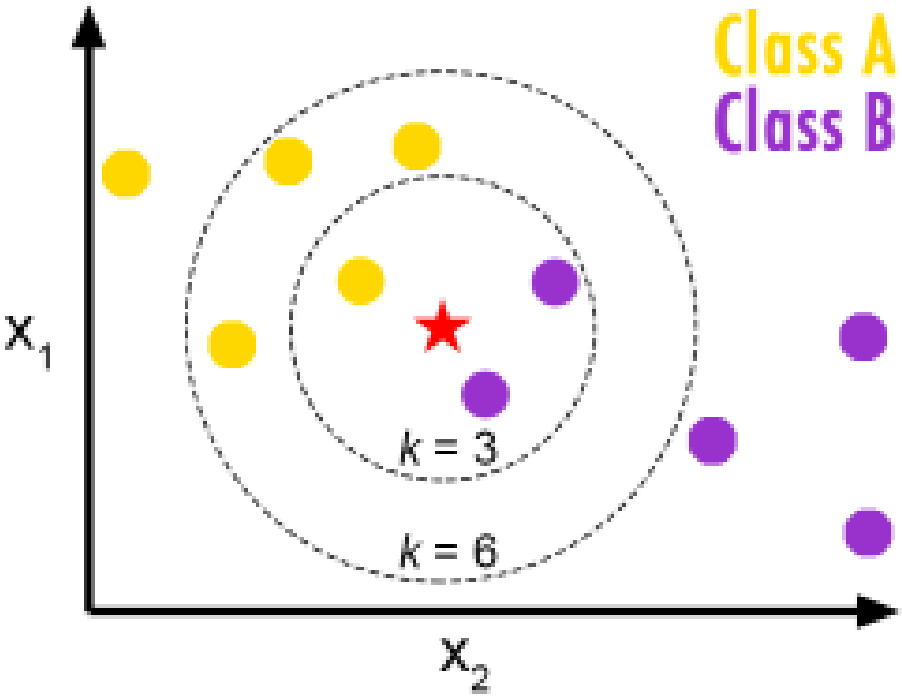


Figura 2.1: Vizualizare KNN

2.3.2 Random Forest

O alternativă eficientă este utilizarea arborilor cu structuri fixe și trăsături aleatorii[10]. Colecțiile de arbori sunt numite păduri, iar clasificatorii construiți astfel se numesc păduri aleatorii. Algoritmul de formare aleatorie a pădurilor necesită trei argumente: datele, o adâncime dorită a arborilor de decizie și un număr K din totalul arborilor de decizie care trebuie construiți. Algoritmul generează fiecare dintre arborii K independent, care face foarte ușoară paralelizarea. Pentru fiecare arbore, construiește un complet arbore binar. Caracteristicile folosite la ramurile acestui arbore sunt selectate aleatoriu, de obicei cu înlocuire, ceea ce înseamnă că aceeași caracteristică poate apărea de mai

multe ori, chiar și într-o singură ramură.

frunzele acestui copac, unde se fac previziuni, sunt completate pe baza datele de instruire. Ultimul pas este singurul punct la care se folosesc datele de antrenament. Clasificatorul rezultat este doar o votare a K- mulți arbori aleatorii.

Cel mai uimitor lucru despre această abordare este că, de fapt funcționează remarcabil de bine. Tind să funcționeze cel mai bine atunci când toate caracteristicile sunt cel puțin puțin relevante, deoarece numărul de caracteristici selectate pentru un anumit copac este mic. Un motiv intuitiv că funcționează bine este următorul. Unii arbori vor interoga caracteristici inutile. Acești arbori vor face, în esență, previziuni aleatorii. Dar câteva din arbori se va întâmpla să interogheze pe caracteristici bune și va face previziuni bune (deoarece frunzele sunt estimate pe baza datele de instruire).

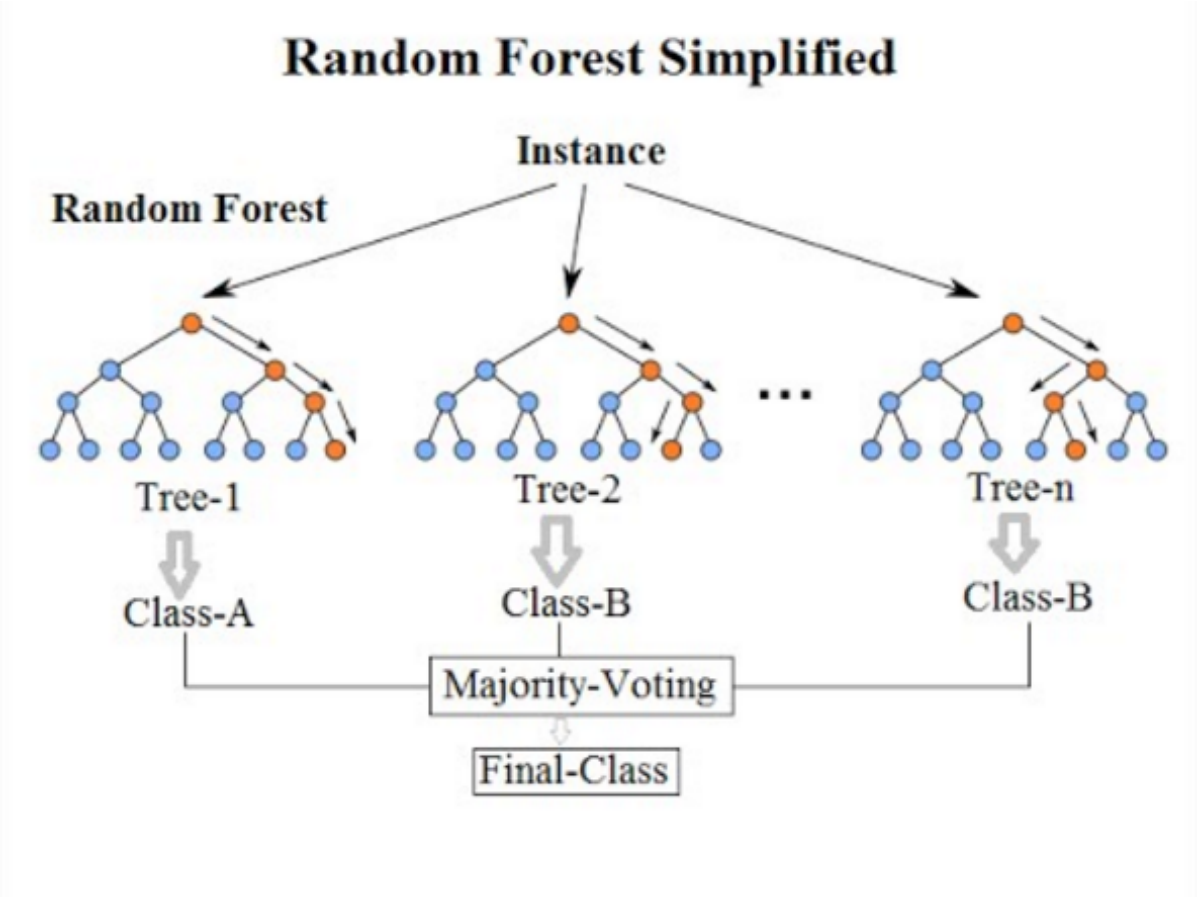


Figura 2.2: Vizualizare Random Forest

2.3.3 XGBoost(eXtreme Gradient Boosting)

XGBoost[11] a fost dezvoltat de Tianqi Chen și face parte dintr-o colecție mai largă de biblioteci open-source dezvoltate de Distributed Machine Learning Community (DMLC). XGBoost este o implementare scalabilă și precisă a mașinilor de amplificare a gradientului și sa dovedit a împinge limitele puterii de calcul pentru algoritmi de arbori amplificați deoarece a fost construit și dezvoltat pentru singurul scop al performanțelor modelului și vitezei computaționale. Mai exact, a fost proiectat să exploateze fiecare bit de resurse de memorie și hardware pentru algoritmi

de amplificare a arborilor.

Implementarea XGBoost oferă câteva funcții avansate pentru reglarea modelului, mediile de calcul și îmbunătățirea algoritmului. Este capabil să realizeze cele trei forme principale de amplificare a gradientului (Gradient Boosting (GB), Stochastic GB și Regularized GB) și este suficient de robust pentru a sprijini reglarea fină și adăugarea parametrilor de regularizare.

Capitolul 3

Implementare

În primul capitol al acestei lucrări am prezentat câteva din amenințările la siguranța sistemelor informatice și posibile metode de prevenire a acestora. Pentru scopul acestei lucrări ne vom folosi tehnică de analiză statică care este prezentată la secțiunea 1.4.1.

3.1 crearea setului de date

Pentru a ne putea apuca de treaba avem nevoie mai întâi de un set de date pe ceare ne putem antreana algotimii. Pentru crearea setului de date am folosit fisere executabile infestate pe ceare le-am downloadat de pe "https://virusshare.com", iar pentru fișierele curate nu am găsit o modalitate de a downloada un număr mare de fișiere executabile[12], astfel multe fișierelor sunt adunate din interiorul windowsului și mai exact fișierele .dll . Pentru a avea acces la baza de date a celor de la virusshare este necesar să ai un cont care trebuie cerut prin e-mail la "admin@virusshare.com". Setul de date conține 10539 PE-files dintre care 6999 infestate și 3540 curate.

Pentru a extrage parametrii din PE-file am folosit biblioteca "pefile". Această este lista finală cu caracteristiciile folosite pentru acest proiect: Name, md5, Machine, SizeOfOptio-

nalHeader, Characteristics, MajorLinkerVersion, MinorLinkerVersion, SizeOfCode, SizeOfInitializedData, SizeOfUninitializedData, AddressOfEntryPoint, BaseOfCode, BaseOfData, ImageBase, SectionAlignment, FileAlignment, MajorOperatingSystemVersion, MinorOperatingSystemVersion, MajorImageVersion, MinorImageVersion, MajorSubsystemVersion, MinorSubsystemVersion, SizeOfImage, SizeOfHeaders, CheckSum, Subsystem, DllCharacteristics, SizeOfStackReserve, SizeOfStackCommit, SizeOfHeapReserve, SizeOfHeapCommit, LoaderFlags, NumberOfRvaAndSizes, SectionsNb, SectionsMeanEntropy, SectionsMinEntropy, SectionsMaxEntropy, SectionsMeanRawsize, SectionsMinRawsize, SectionMaxRawsize, SectionsMeanVirtualsize, SectionsMinVirtualsize, SectionMaxVirtualsize, ImportsNbDLL, ImportsNb, ImportsNbOrdinal, ExportNb, ResourcesNb, ResourcesMeanEntropy, ResourcesMinEntropy, ResourcesMaxEntropy, ResourcesMeanSize, ResourcesMinSize, ResourcesMaxSize, LoadConfigurationSize, VersionInformationSize.

```
output = "data.csv"
csv_delimiter = "|"
columns = [
"Name",
"md5",
"Machine",
"SizeOfOptionalHeader",
"Characteristics",
"MajorLinkerVersion",
"MinorLinkerVersion",
"SizeOfCode",
"SizeOfInitializedData",
"SizeOfUninitializedData",
"AddressOfEntryPoint",
"BaseOfCode",
"BaseOfData",
"ImageBase",
"SectionAlignment",
"FileAlignment",
```

```
"MajorOperatingSystemVersion",
"MinorOperatingSystemVersion",
"MajorImageVersion",
"MinorImageVersion",
"MajorSubsystemVersion",
"MinorSubsystemVersion",
"SizeOfImage",
"SizeOfHeaders",
"CheckSum",
"Subsystem",
"DllCharacteristics",
"SizeOfStackReserve",
"SizeOfStackCommit",
"SizeOfHeapReserve",
"SizeOfHeapCommit",
"LoaderFlags",
"NumberOfRvaAndSizes",
"SectionsNb",
"SectionsMeanEntropy",
"SectionsMinEntropy",
"SectionsMaxEntropy",
"SectionsMeanRawsize",
"SectionsMinRawsize",
"SectionMaxRawsize",
"SectionsMeanVirtualsize",
"SectionsMinVirtualsize",
"SectionMaxVirtualsize",
"ImportsNbDLL",
"ImportsNb",
"ImportsNbOrdinal",
"ExportNb",
"ResourcesNb",
"ResourcesMeanEntropy",
"ResourcesMinEntropy",
"ResourcesMaxEntropy",
"ResourcesMeanSize",
"ResourcesMinSize",
"ResourcesMaxSize",
"LoadConfigurationSize",
"VersionInformationSize",
"legitimate"
]
```

```
ff = open(output, "a")
ff.write(csv_delimiter.join(columns) + "\n")

# Launch legitimate
for ffile in os.listdir('legitimate'):
   print(ffile)
   try:
      res = extract_infos(os.path.join('legitimate/', ffile))
      res.append(1)
      ff.write(csv_delimiter.join(map(lambda x:str(x), res)) + "\n")
   except pefile.PEFormatError:
      print('\t -> Bad PE format')

for ffile in os.listdir('/hdd/Downloads/virusi'):
   print(ffile)
   try:
      res = extract_infos(os.path.join('/hdd/Downloads/virusi/',
          ffile))
      res.append(0)
      ff.write(csv_delimiter.join(map(lambda x:str(x), res)) + "\n")
      shutil.copy("/hdd/Downloads/virusi/"+ffile,"malicious/")
   except pefile.PEFormatError:
      print('\t -> Bad PE format')
   except:
      print('\t -> Weird error')
ff.close()
```

Fișierele curate și fișierele infestate sunt puse în foldere separate. Pentru a păstra doar fișierele executabile din cei 100 GB de viruși downloadati de pe virusshare.com am folosit librăria "shutil" pentru a le copia într-un folder diferit "malicious - 13,4 GB". Acest lucru poate fi observat în ultimile linii ale codului de mai sus. Fișierele executabile curate sunt păstrate în folderul "legitimate 8,6 GB". ultima coloana din setul de date va conține un "1" dacă fișierul este curat sau un "0" dacă fișierul este infestat, acest lucru ne va ajută la antreanarea algoritmiilor de machine learning și la testarea acurateții acestora.

3.1.1 Pregatirea setului de date

Machine learning utilizează numai date întregi sau de tip float ca elemente de detectare. Pentru pregătirea setului de date vom folosi librăria "panda". Primul pas este să importăm setul de date:

```
import pandas as pd
dataset = pd.read_csv('data.csv', sep = '|')
X = dataset.drop(['Name', 'md5', 'legitimate'], axis = 1).values
y = dataset['legitimate'].values
```

În X sunt ținute toate datele înafara de nume,md5 și coloana care reprezintă dacă un fișier este curat sau infestat. Acea coloana este tinuta în y. După ce X și y sunt pregătite trebuie să împărțim întreg setul de date în train and test proporție de 80/20

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.20, random_state = 0)
```

Multe elemente folosite în funcția obiectivă a unui algoritm de învățare presupun că toate caracteristicile sunt centrate în jurul valorii de 0 și au variație în aceeași ordine. Dacă o caracteristică are variante de mărime mai mari decât altele, această ar putea să domine funcția obiectivă și să facă evaluatorul incapabil să învețe din alte caracteristici. Pentru a evita acest lucru vom folosi "StandardScaler"

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

În acest moment setul de date este pregătit pentru a folosii algoritmii de machine learning.

3.2 Algoritmi de Learning

3.2.1 KNN

Pentru acest algoritm este necesar sa specificam un numar K care reprezinta numarul de vecini. Putem incerca sa rulam algoritmul cu diferiti K pentru a vedea care K este mai bun pentru problema noastra. Pentru a afla care este cel mai optim K vom folosi Cross-Validation(este o metodă statistică utilizată pentru a estima abilitățile modelelor de învățare automată.)

```
neighbors = list(range(1,50))
# empty list that will hold cv scores
cv_scores = []

# perform 20-fold cross validation
for k in neighbors:
knn = KNeighborsClassifier(n_neighbors=k)
scores = cross_val_score(knn, X_train, y_train, cv=20,
    scoring='accuracy')
cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]
MSE_list = np.array(MSE)
neighbors_list = np.array(neighbors)
```

```
In [20]:optimal_k =
    neighbors[MSE_list.tolist().index(min(MSE_list))]
   ...:print ("The optimal number of neighbors is %d" % optimal_k)
   The optimal number of neighbors is 3
```

```
plt.plot(neighbors_list, MSE_list)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
```

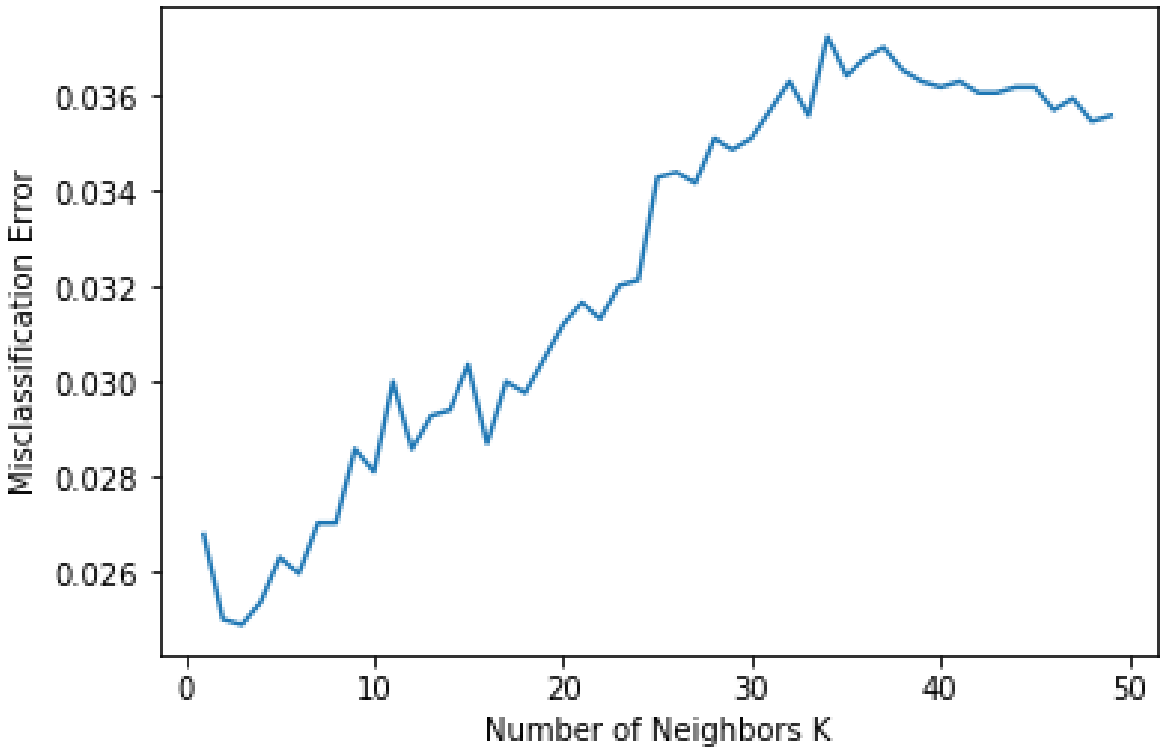


Figura 3.1: plot misclassification error vs k

Din figura 3.1 ptem observa ca cel mai optim K pentru aceasta situatie este: 3.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3, metric =
    'minkowski', p = 2)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

y_pred reprezinta prezicerile facute de algoritmul nostru. Pentru a testa acuratețea acestuia este necesar sa creem matricea de confuzie bazat pe y_test si y_pred.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred).
```

Matricea de confuzie este un tabel specific care permite vizualizarea performanței unui algoritm.

- pozitia 00 reprezinta True Positives
- pozitia 11 reprezinta True Negatives
- pozitia 01 reprezinta False Positives
- pozitia 10 reprezinta False Negatives

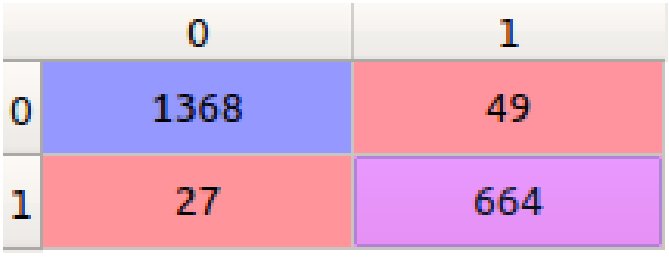


Figura 3.2: matricea de confuzie

Din cate putem observa din matricea de confuzie argoritmul nostru a prezis corect 2032 fisiere iar 76 sunt preziceri gresite, ceea ce duce la o acuratețe de 96.4%.

3.2.2 Random Forest

Pentru acest algoritm este nevoie sa specificam numarul de estimatii. Dupa mai multe incercari am ajuns la concluzia ca pentru problema de fata cel mai potrivit numar de estimatii este 50 iar cel de-al 2-lea parametru folosit este Funcția de măsurare a calității unei divizări. Criteriile acceptate sunt "gini" pentru impuritatea Gini și "entropia" pentru câștigul de informație.

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 50, criterion =
    'entropy')
```

```
classifier.fit(X_train, y_train)

#predict the test results
y_pred = classifier.predict(X_test)

#Makeing the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

	0	1
0	1408	9
1	28	663

Figura 3.3: matricea de confuzie

Din cate putem observa din matricea de confuzie argoritmul nostru a prezis corect 2071 fisiere iar 37 sunt preziceri gresite, ceea ce duce la o acuratețe de 98.24%.

3.2.3 XGBoost

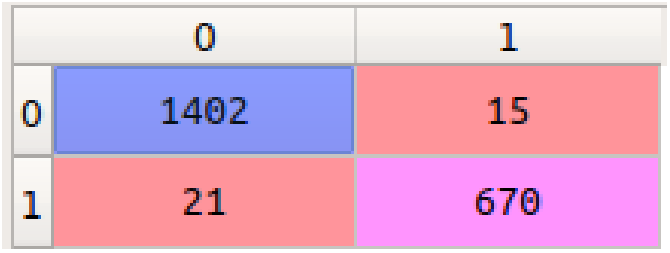
Parametrii folositi pentru acest Algoritm sunt adancimea maxima(max_depth), rata de invatare(learning_rate)si numarul de estimatii(n_estimators).

```
from xgboost import XGBClassifier
classifier = XGBClassifier(max_depth=10, learning_rate=0.1,
    n_estimators=50)
classifier.fit(X_train, y_train)

#predict the test results
y_pred = classifier.predict(X_test)

#Makeing the confusion matrix
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```



	0	1
0	1402	15
1	21	670

Figura 3.4: matricea de confuzie

Din cate putem observa din metrix-ul de comfuzie argoritmul nostru a prezis corect 2072 fisiere iar 36 sunt preziceri gresite, ceea ce duce la o acuratețe de 98.29%

3.2.4 Selectare de caracteristici

Scopul selectarii de caracteristici este de a micsora setul nostru de date format din 54 de caracteristici intr-un set de date care sa contina doar caracteristicile mai mult relevante pentru a diferentia fisierele curate de cele infestate. Pentru a observa daca aceasta tehnica are un impact pozitiv asupra acurateții algoritmiilor folositi vom folosi Tree-based feature selection.

```
from sklearn.feature_selection import SelectFromModel
import sklearn.ensemble as ske
fsel = ske.ExtraTreesClassifier().fit(X, y)
model = SelectFromModel(fsel, prefit=True)
X_new = model.transform(X)
nb_features = X_new.shape[1]
indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
for f in range(nb_features):
print("%d. feature %s (%f)" % (f + 1,
    dataset.columns[2+indices[f]],
    fsel.feature_importances_[indices[f]]))

In [140]: X.shape
```

```
Out[140]: (10539, 54)

In [142]: X_new.shape
Out[142]: (10539, 11)
```

In cazul de fata algoritmul a selectat 11 caracteristici din cele 54 de caracteristici esentiale.

```
In [143]:nb_features = X_new.shape[1]
    ...indices =
       np.argsort(fsel.feature_importances_)[::-1][:nb_features]
    ...for f in range(nb_features):
    ...print("%d. feature %s (%f)" % (f + 1,
       dataset.columns[2+indices[f]],fsel.feature_importances_[indices[f]]))
```

Aceasta este lista cu caracteristicile pastrate si importanta acestora.

1. feature MajorSubsystemVersion (0.155935)
2. feature Characteristics (0.118596)
3. feature MajorOperatingSystemVersion (0.110355)
4. feature ImageBase (0.108349)
5. feature Machine (0.068662)
6. feature DllCharacteristics (0.050951)
7. feature SectionsMaxEntropy (0.048521)
8. feature LoadConfigurationSize (0.038567)
9. feature ResourcesMaxEntropy (0.038000)
10. feature MajorLinkerVersion (0.032729)
11. feature ResourcesMinSize (0.022452)

	0	1
0	1387	30
1	24	667

Figura 3.5: knn cu selectare de caracteristici

Acum vom aplica aceasi algoritmi de machine learning pe noul set de date X_new si vom observa in ce masura se modifica acuratețea acestora.

Dupa cum putem observa in figura 3.5 rezultatele algoritmului pe noul set de date sunt: 54 preziceri gresite si 2054 preziceri corecte, ceea ce rezulta o acuratețe de 97.44%. Acuratețea algoritmului a crescut cu 1.04%

	0	1
0	1405	12
1	24	667

Figura 3.6: Random Forest cu selectare de caracteristici

In figura 3.6 observam ca algoritmul a efectuat 2072 preziceri corecte si 36 preziceri gresite. Acuratețea acestuia este de 98.29%. Acuratețea acestuia a crescut cu 0.05%

	0	1
0	1409	8
1	27	664

Figura 3.7: XGBoost cu selectare de caracteristici

In figura 3.7 numarul de predictii corecte este de 2073 iar numarul de predictii gresite este de 25. Acuratețe este de 98.34, cu 0.05% mai mare.

Capitolul 4

Concluzie

Scopul lucrarii este de a prezenta o abordare machin learning asupra problemei malware. Datorita cresterii bruste a malware-urilor avem nevoie de metode automate de depistare a fisierelor infestate.

In prima fază a lucrarii este creat setul de date folosind executabile infestate si curate, pentru a extrage datele necesare creări setului de date am folosit un script creat in Python. Dupa crearea setului de date, acesta trebuie sa fie pregatit pentru a antrena algoritmii de machine learning. Algoritmii folositi sunt: Knn, Random Forest si XGBoost prezentati comparativ. Dupa aplicarea algoritmilor cea mai buna acurateţe a avut-o algoritmul XGBoost cu o acurateţe de 98.29%, iar dupa aplicarea unui algoritm care selecteaza caracteristici asupra setului de date, acurateţea algoritmilor a crescut: Knn cu 1.04%, Random Forest cu 0.05% iar algoritmul cu cea mai buna acurateţe a fost XGBoost 98.34%, îmbunătăţindu-se cu 0.05%. Lucrarea de faţă demonstrează că XGBoost este cel mai optim algoritm pentru depistarea programelor maliţioase. Pe viitor această acurateţe poate fi îmbunătăţită daca adaugam un numar mult mai mare de fisiere in setu de date pentru a antrena algoritmii. Fiecare algoritm are mai multi parametri care pot fi testati cu diferite valori pentru a creste acurateţea acestora. Acest proiect pote

ajunge la nivel de aplicatie cu ajutorul unei biblioteci nimite pickle, pentru a salva ceea ce a invatat algoritmul iar apoi putem testa un fisier nou pentru a vedea daca este curat sau infectat.

Bibliografie

[1] Malware Types and Classifications, *Bert Rankin,* 28.03.2018, publicat în *LastLine*, ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[2] A Brief History of Malware — Its Evolution and Impact, *Bert Rankin,* 05.04.2018, publicat în *LastLine*, ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[3] Detecting malware through static and dynamic techniques, *Jeremy Scott,* 14.09.2017, publicat în *NTT Security*, ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[4] Hybrid Analysis and Control of Malware, *Kevin A. Roundy and Barton P. Miller*, International Workshop on Recent Advances in Intrusion Detection, pag. 317−338, 2010, Springer.

[5] Advanced Malware Detection - Signatures vs. Behavior Analysis *John Cloonan Director of Products, Lastline,* 11.04.2017, publicat în *Infosecurity Magazine*, ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[6] What is Machine Learning?, *Daniel Faggella,* 12.08.2017, publicat în *techemergence*, ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[7] Data mining, *Margaret Rouse*, *Search SQL Server* ultima accesare 12.09.2018, articolul se regăseşte *aici*.

https://searchsqlserver.techtarget.com/definition/data-mining

[8] Supervised and Unsupervised Machine Learning Algorithms, *Jason Brownlee,* 16.03.2016, publicat în *Machine Learning Algorithms*, ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[9] Nearest Neighbors, *scikit-learn.org* ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[10] RandomForestClassifier, *scikit-learn.org* ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[11] GradientBoostingClassifier, *scikit-learn.org* ultima accesare 12.09.2018, articolul se regăseşte *aici*.

[12] Malware Researcher's Handbook,

Resources Infosecinstitute, ultima accesare 12.09.2018, articolul se regăseşte *aici*.