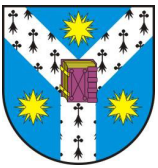

Universitatea Alexandru Ioan Cuza
Facultatea de Informatică

Colecție de Modele Hibride folosite în detecția Atacurilor Cibernetice

Autor:

Cristina
VATAMANU

Îndrumător:

Prof. Dr. Henri
LUCHIAN

0.1 Istoria evoluției programelor malițioase

Programele malițioase au evoluat de-a lungul timpului în mai multe forme, devenind din ce în ce mai complexe în fiecare zi. Începutul acestora se bazează pe farse între indivizi care au avut o pasiune pentru informatică. Începând cu creșterea Internetului, lucrurile au căpătat un aspect financiar în spațiul cibernetic și ceea ce a fost până atunci un concept, începând cu acel moment, ar fi putut produce bani. Începând cu anii 2005, 2006 au fost create diferite tipuri de programe, care deserveau un număr mare de scopuri, dar cu același rezultat, câștigul de bani: furtul de credențiale bancare, promovarea anunturilor publicitare, înșelăciunea persoanelor prin phishing și înșelătorii privind cărțil de credit și așa mai departe. Deoarece accentul a fost acela de a câștiga bani, creatorii acestor amenințări trebuiau să se asigure că fisierele lor vor rămâne nedetectate cât mai mult posibil, de aceea, în această perioadă, au fost dezvoltate multe tehnici de evitare a detectiei unui produs de securitate. Au existat trei abordări principale:

1. schimbarea fișierul binar mai repede decât soluția de securitate se poate actualiza (tehnici defensive)

2. actiuni care dac analiza și detectarea cât mai grele posibile (tehnici de protecție)
3. măsuri la nivel de sistem pentru a rămâne nedetectați (tehnici ofensive)

Numărul mare de eșantioane malițioase văzute în piață este o problemă cu care soluțiile de securitate încă se luptă și în prezent. Dar, începând cu anul 2010, s-a observat că aspectul financiar nu era singurul scop. În ultimii ani, amenințările cibernetice au fost folosite ca arme pentru scopuri mult mai specifice, cum ar fi furtul de informații, colectarea proprietăților intelectuale, spionaj național realizat de guverne împotriva altor guverne sau pentru supravegherea populației. Este clar că peisajul amenințării cibernetice a evoluat în timp și este în continuă schimbare, devenind mai sofisticat, complex și agresiv. Pentru a face față tuturor atacurilor cibernetice, s-au făcut multe cercetări în această direcție, iar soluțiile de securitate au început să integreze tehnologiile bazate pe modelele de învățare automată. Metodele de detectare, bazate pe algoritmi de învățare automată, trebuie să ia în considerare numeroase aspecte, pentru a fi eficiente și sigure, cum ar fi:

1. Un model trebuie să se poată adapta rapid pentru a ține pasul cu peisajul în continuă schimbare.
2. Cele mai multe dintre tehnologiile folosite pentru detecție funcționează într-un mod blocant. Este important ca impactul de performanta introdus într-un sistem (timpul necesar pentru a lua o decizie dacă un fișier este curat sau nu) să fie cât mai mic posibil.
3. Modelele integrate într-un produs de securitate trebuie să ocupe cat mai puțină memorie pentru a evita o blocarea sistemului și pentru a asigura funcționarea corectă a acestuiai.
4. Una dintre principalele preocupări în această perioadă a fost aceea de a crea modele proactive, agresive capabile să detecteze și să blocheze noi amenințări necunoscute, dar în același timp să mențină o rată de alarme false cât mai scăzută.

Această teză acoperă unele mecanisme hibride, destinate a fi utilizate împotriva unor seturilor mari de fișiere, abordări care încearcă să găsească cel mai bun echilibru între toate cele patru restricții menționate mai sus.

0.2 Modelele hibride utilizate pentru detectarea amenințărilor cibernetice

Amenințările cibernetice au o istorie de aproximativ 25 de ani și au și-au schimbat formele de-a lungul anilor, fiind foarte dinamice, acoperind o arie largă de tehnici

malitioase, interese ascunse și pagube cauzate. Obiectivul unei soluții de securitate este de a proteja pe cât mai bine posibil utilizatorii împotriva acestor amenințări. Integrarea unui model de clasificare în cadrul unui produs de securitate este un proces complicat. Deși, teoretic, un model generează rezultate bune, există patru restricții principale, stricte, impuse de utilizarea practică:

1. *Rata de detecție: TPR* - Principalul obiectiv a unui produs de securitate este de a proteja și detecta amenințările, acest lucru fiind ilustrat în mod direct de rata de detecție a diferitelor tehnologii integrate.
2. *Rata de alarme false: FPR* - O soluție de securitate trebuie să protejeze, dar nu ar trebui să detecteze totul. Un număr redus de fișiere legitime incorecte clasificate este obligatoriu.
3. *Impactul de performanta (sau Timpul de Evaluare): ET* - Impactul asupra performanței este definit de timpul necesar modelului pentru a efectua calculul necesar pentru a lua decizia dacă o înregistrare aparține unei clase sau alteia.
4. *Dimensiunea modelului: MF* - Dimensiunea se referă la spațiul necesar pentru a stoca modelul rezultat, ilustrat de obicei de ID-urile atributelor și de setul de valori corespunzătare acestora.

Atunci când analizăm un model, trebuie să măsuram întotdeauna aceste restricții, deoarece orice modificare a parametrilor algoritmului ar afecta unul sau mai mulți dintre acești indicatori de performanță.

0.3 Baza de date folosită în experimentele practice DB-CAS-BDT

	Count
Elemente Benigne	2082322
Elemente Malițioase	242805
Total Elemente	2325127
Numar de atribute	6275

0.4 Clasificatorul Liniar

Unele dintre cercetările de față se bazează pe un clasificator liniar modificat, denumit One Side Class Perceptron - OSCP ([1]) și această secțiune va oferi câteva detalii despre proprietățile sale principale. Clasificatorul modifică algoritmul Perceptron al lui Rosenblatt ([2]) pentru a satisface anumite restricții practice ale industriei de securitate cibernetică.

Principalul obiectiv al algoritmului OSCP este clasificarea corectă a tuturor înregistrărilor dintr-o anumită

clasă. Acesta a fost dezvoltat pentru a minimiza numărul de alarme false. Modul în care reușește să facă acest lucru este prin adăugarea unui pas suplimentar la sfârșitul fiecărei iterații de antrenare, un pas care modifică poziția hiper-planului astfel încât să se garanteze că toate elementele dintr-o anumită clasă sunt corect clasificate. Toate iterațiile de antrenare care utilizează toate elemntele sunt urmate de un astfel de pas, intitulat *"Pas de minimizare"*, care va continua să antreneze modelul folo-sind doar ele-mentele unei clasei. La sfârșitul fiecărei iterații, intr-un subspațiu vor fi *toate* elementele din prima clasă si câteva din a doua clasă, iar al doilea subspațiu vor fi *doar* elemente din clasa a doua.

Pentru o ilustrare simplă, un exemplu dintr-un spațiu bidimensional al unei iterații de antrenare este ilustrat în figura 1. Prima imagine (a) reprezintă setul de date D. După pasul de antrenament, care utilizează toate înregistrările, poziția hiper-planului va arăta ca în figura (b). Înainte de a trece la următoarea iterație de antrenare,

se aplică pasul de minimizare, unde modelul va fi antrenat numai cu înregistrările aparținând unei clase S, înregistrările benigne in acest exemplu. După acest pas, hiperplanul se va schimba astfel încât toate înregistrările legitime vor fi corect clasificate (figura (c)).

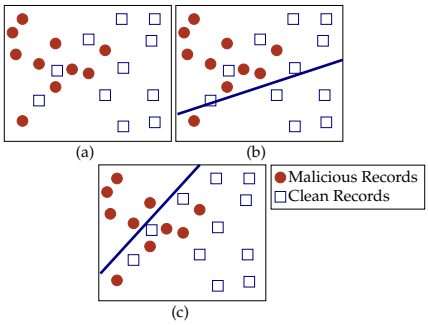

Figure 1: One Side Class Perceptron - OSCP iteratie de antrenare. Pas de antrenare (b). Pasul de minimizare (c)

Există mai multe avantaje ale acestui algoritm în ceea ce privește utilizarea practică:

- Are zero alarme false în timpul procesului de antrenare
- Are un număr foarte mic de alarme false pe evaluate
- Este rapid în ceea ce privește timpul de antrenare și timpul de evaluare
- Din cauza modului în care a fost dezvoltat poate fi paralelizat, astfel încât procesul de antrenare poate fi distribuit pe mai multe mașini
- Folosește o cantitate mică de memorie

Acest algoritm a fost dezvoltat pentru prima oară ca filtru pentru minimizarea numărului de alarme false. Modul în care funcționează, are și unele dezavantaje:

- Rata de detectie mică (mai mică decât cea obținută de perceptronul clasic al lui Rosenblatt)
- Rezultatele sunt foarte scăzute dacă elementele care sunt etichetate diferit au același set de caracteristici, de aceea este foarte important ca baza de date sa fie filtrată.

0.5 Învătarea Cascadată

Având algoritmul OSCP optimizat pentru a avea rezultate bune pe trei indicatori (rata de alarme false, amprenta de memorie și impactul performanței), următorul pas a fost să îmbunătățim rata de detecție. Prima abordare luată în considerare a fost învățarea în cascadă. Cascadarea este o clasă specială de învățare ensemble, care combină diferite modele de învățare automata. Ca și în teoria ensemble, cascadarea folosește informațiile furnizate de ieșirea unui clasificator ca date suplimentare pentru următorul clasificator (Adapting Boosting). Această parte a cercetării se bazează pe un clasificator în mai multe etape. În fiecare etapă, un model, care este capabil să izoleze un subset de elemente ca fiind corect

clasificate, va elimina acel subset din baza de date. În acest fel, următorul clasificator se va concentra doar pe datele care sunt mai dificil de clasificat. Algoritmul 1 descrie abordarea luată în considerare.

Algorithm 1 Modele Cascadate

1: $D \leftarrow \bigcup_{i=1}^{n} x_i$ - baza de date
2: $M \leftarrow \{M_1, M_2, ...M_n\}$ - setul de modele
3: **function** $Cascade(D, M)$
4: **while** $|D| > 0$ **do**
5: Alege M_i din M, luand in considerare D
6: $C \leftarrow M_i(D), 0 < |C| < |D|$
7: $D \leftarrow D \setminus C$
8: **end while**
9: **end function**

Ideea din spatele acestei abordări este că seturi de atribute diferite descriu mai bine o anumite zone a setului de date decât alte seturi de atribute. Aceasta înseamnă că se pot construi diferite modele care utilizează anumite seturi de atribite pentru a descrie cel mai bine o parte din setul de date sau, cu alte cuvinte, modele specializate pentru anumite părți ale bazei de date.

Deoarece la fiecare etapă de cascadă, un clasificator trebuie să izoleze un subset de înregistrări, care ar trebui eliminat din bazele de date a următoarelor etape, acest studiu a profitat de proprietatea clasificatorului OSCP. Algoritmul asigură că pe o parte a hiper-planului sunt

toate elementele dintr-o clasă și unele înregistrări clasificate greșit din clasa a doua, iar pe cealaltă parte înregistrările corect clasificate aparținând doar clasei a doua, aceste ultime înregistrări pot fi eliminate din baza de date a următoarei etape de cascadă. Cu alte cuvinte, la fiecare etapă de cascadă, un algoritm OSCP este antrenat obținând la sfârșitul etapei un model și setul de date pentru etapa următoare. Algoritmul se oprește fie atunci când toate înregistrările sunt corect clasificate, fie când se atinge numărul maxim de etape (condiția impusă în practică). În timpul acestui studiu, diferite combinații de algoritmi OSCP au fost utilizate în etapele de cascadă:

- Cascading OSCP one class
- Cascading OSCP alternative classes
- Cascading OSCP k-alternative classes
- Cascading OSCP best class
- Cascading OSCP both classes

Cele mai bune rezultate au fost obținute de ultima soluție care va fi utilizată în continuare. Un exemplu intr-un spațiu bidimensional pe care este aplicat un OSCP Both Classes este ilustrat în Figura 2.

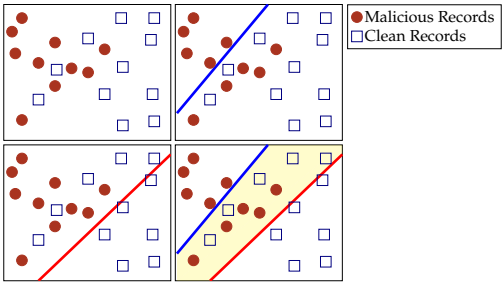

Figure 2: OSCP Both Classes

După cum se poate observa, algoritmul izolează două subseturi de elemente clasificate corect (unul aparținând clasei de fișiere benigne și unul în clasa fișierelor malițioase) și un set de înregistrări care nu sunt clasificate de nici una dintre cele două hiper-planuri. Un aspect interesant care trebuie luat în considerare în învățarea cascadata este reprezentat de atriute. Având un numar mare de atribute care descriu datele impune necesitatea unui proces de selecție a acestora. De obicei, selecția atributelor este efectuată o singură dată, la începutul procesului de antrenare. Cu toate acestea, când vine vorba despre învățarea în cascadă, putem alege să realizăm acest proces la începutul fiecărei etape, selectând de fiecare dată un set diferit de atribute. Astfel, fiecare model va fi antrenat folosind atribute potrivite pentru subsetul curent. O altă observație în timpul acestui studiu a fost că, în

toate experimentele efectuate, algoritmul a ieșit prin conditia de stop impusă de practică, numărul maxim de etape a fost întotdeauna atins. Pentru a identifica limita maximă adecvată, s-au efectuat mai multe teste, în special testele au încercat să execute algoritmul cu cât mai multe etape posibil pentru a clasifica corect toate datele. Deoarece setul de date este foarte mare, au fost selectate subseturi din acesta. Primul set de date considerat a avut 33240 de înregistrări (3194 fișiere malițioase și 30046 benigne). Algoritmul utilizând un clasificator OSCP-BC, antrenat pentru 500 de iterații, a reușit să clasifice corect toate datele în 19 etape de cascadă. Următorul pas a fost să se crească de 10 ori dimensiunea setului de date. Următoarea bază de date de testare conținea 332404 de înregistrări: 31940 fișiere malițioase și 300464 legitime. La etapa 39, modelul curent nu a reușit să izoleze nici o înregistrare care să fie eliminată. Din acest moment, numărul de iterații de antrenare s-a dublat, iar pentru etapele următoare clasificatorul OSCP a fost antrenat pentru 1000 de iterații. Din păcate, algoritmul a fost blocat din nou la etapa a 46-a și din nou la etapa a 58-a. La sfârșitul celei de-a 58-a etape, 113560 înregistrări

au rămas neclasificate. Concluzia experimentului a fost că algoritmul are și va continua să aibă o scădere foarte lentă. Numai primele trei etape au reușit să clasifice corect și să elimine din setul de date mai mult de jumătate din înregistrări. Din etapa a șasea, scăderea a început să fie și a rămas aproape imperceptibilă, cu excepția etapelor 39 și 46, care corespund celor două etape de cascadă pentru care numărul de iterații de antrenare pentru clasificatorul OSCP a fost dublat. După efectuarea acestor teste, limita maximă pentru algoritmul cascadă a fost de 10 etape și a fost considerată o alegere rezonabilă. În timpul procesului de analiză a datelor, a fost evaluată și antrenamentul clasificatorului. S-a constatat că, în unele cazuri, datele au un comportament ciudat. Începând cu cea de-a doua etapă de cascadă, în timpul antrenamentului OSCP, între 50% și 70% din iterații au produs această acțiune: numărul de înregistrări clasificate corect au alternat între o valoare mică și o valoare ridicată și numai în ultimele iterații, rata de detecție pare să aibă o creștere liniară, dar rata de detecție OSCP nu a reușit să atingă un prag. Pentru a observa performanța sa, a fost realizat un experiment și, începând cu a doua etapă

de cascadă, algoritmul OSCP-Both Class a fost antrenat pentru 3000 de iterații. Experimentul este ilustrat în Figura 3.

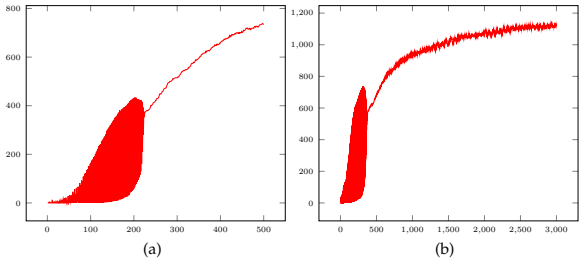

Figure 3: Procesul de antrenare pentru OSCP-Both Classes cu 500 de iterații (a) și 3000 de iterații (b)

După cum se poate observa, clasificatorul OSCP-Both Class a ajuns la un prag în termeni de rata de detecție începând cu iterația 2000. Deoarece numai 10% din iterații au prezentat comportamentul descris mai sus (alternanța ratei de detecție), crestrea ratei de detecție a fost mai vizibilă.

Experimentul a evidențiat faptul că rezultatele mai bune sunt strâns legate de numărul de iterații pentru care este antrenat clasificatorul. Dar, în practică, posibilitatea de a antrena un clasificator într-o perioadă decentă de timp, de obicei vine cu stabilirea de restricții asupra parametrilor săi, iar numărul de iterații este unul dintre ele. Având

în vedere faptul că abordarea în învățare cascadată, luată în considerare în acest studiu, elimina în fiecare etapă o serie de înregistrări, seturile de date pentru etapele următoare vor deveni mai mici, ceea ce înseamnă că timpul de prelucrare va scădea și el. Se poate profita de această proprietate și se poate schimba câștigul în timpul procesării cu îmbunătățirea următorilor clasificatori. Prima tehnică aplicată în această parte a studiului a fost creșterea progresivă a numărului de iterații de antrenare a OSCP cu fiecare etapă de cascadă. În prima etapă clasificatorul a fost antrenat pentru un număr constant de iterații, impuse din motive practice. Începând cu a doua etapă, când aproape jumătate din înregistrări sunt eliminate, numărul de iterații de antrenare crește cu un factor. Împreună cu numărul de iterații de antrenare, un alt parametru de mare importanță îl reprezintă setul de atribute. Deoarece modelul este destinat să fie utilizat în practică, acest lucru înseamnă, de obicei, că este necesar să se efectueze selecția atributelor. Într-o manieră similară cu cea descrisă anterior, se poate efectua o ajustare în ceea ce privește numărul acestora. În prima etapă clasificatorul a fost antrenat cu un număr constant de

atribute, impuse de motive practice. Începând cu cea de-a doua etapă, acest număr a crescut în fiecare etapă cu un factor. De asemenea, a fost luată în considerare o combinație a celor două tehnici, în care atât numărul de iterații de antrenare, cât și numărul de atribute utilizate in antrenarea clasificatorului OSCP-Both Class au fost ajustate la fiecare etapă de cascadă. Studiile descrise anterior au fost testate pe baza de date DB-CAT-BDT. Numărul de iterații de antrenare și numărul de atribute utilizate în antrenament, impuse din motive practice, au fost ambele setate la 500. Cele 500 de atribute cele mai importante au fost selectate folosind funcția scor F [3]. Toți algoritmi au fost testați efectuând o validare de tipul 3 fold. Au fost create patru modele noi în cascadă, factorul w luând în mod succesiv 4 valori diferite:

$$nrAtribute = 500 + nrEtapa \times w,$$

$$unde: w \in \{100, 200, 300, 400\}, nrEtapa = \overline{0,9}$$

În al doilea rând, aceeași tehnică a fost aplicată pentru numărul de iterații de antrenare ale clasificatorului OSCP-BC, obținând două noi modele:

$$nrIteratii = 500 + nrEtapa \times k,$$

$$unde:\ k\ \in\ \{100,\ 200\}, nrEtapa\ =\ \overline{0,9}$$

Ultimul experiment se referă la ajustarea atât a numărului de atribute, cât și a numărului de iterații de antrenare. S-a creat un nou model, CAS-200F-200IT, ambii parametri fiind ajustați cu un factor egal cu 200. Toate rezultatele sunt rezumate în Tabelul 1:

Algoritm	FPR	TPR	Timp Antrenare
OSCP-BC	0.0069 %	29.7832 %	2:05:13
CAS	0.0159 %	34.9699 %	3:09:24
CAS-100F	0.0334 %	45.9307 %	5:07:26
CAS-200F	0.0374 %	49.9215 %	6:07:44
CAS-300F	0.0342 %	49.4652 %	7:00:54
CAS-400F	0.0299 %	48.5496 %	7:51:11
CAS-100IT	0.0220 %	36.1729 %	6:49:06
CAS-200IT	0.0242 %	36.6923 %	9:16:27
CAS-200F-200IT	0.0537 %	56.8767 %	13:39:01

Table 1: Rezultate obținute in experimentele practice pentru algoritmi studiati în învatarea cascadată

0.6 Colecție de modele hibride bazate pe Arbori binari de decizie - BDT

Invatarea bazata pe arbori binari de decizie reprezintă un model care traduce observațiile cu privire la anumite elemente în concluzii privind valoarea sau eticheta

avesyora. Un arbore de decizie constă dintr-un nod rădăcină, mai multe noduri de decizie și noduri terminale (frunzele).

0.6.1 Divizarea bazei de date folosind un Arbore Binar de Decizie

În prezentul studiu nodul rădăcină deține toate înregistrările din baza de date DB-CAS-BDT. La fiecare nod de decizie este evaluată o condiție privind un anumit atribut. Aceste condiții sunt utilizate pentru împărțirea setului de date curent în două subgrupuri. Dacă o înregistrare are atributul evaluat setat pe 1, atunci acea înregistrare este salvată în nodul din dreapta (subclasa), altfel va fi salvată în nodul stâng. Setul de atribute utilizate în evaluările succesive este ales folosind scoruri diferite. La fiecare nod de decizie, atributul cu cel mai mare scor este aleas pentru evaluare. Funcția scor utilizată în aceast studiu este denumit Median Close (MC) și este descris de ecuația 1.

Considerând:
$D \leftarrow \bigcup_{i=1}^{n} \left\{ x_i | x_i \in \mathbb{R}^m \right\} - baza\ de\ date$
$F \leftarrow \bigcup_{j=1}^{m} F_j - setul\ de\ atribute$

$$\forall F_j, j = \overline{1, m}$$
$$MC_{F_j} = \frac{(1 - |\frac{countClean}{totalClean} - 0.5|) + (1 - |\frac{countMalicious}{totalMalicious} - 0.5|)}{2} \quad (1)$$

unde:

- $countClean$ - numărul de înregistrări aparținând clasei de fisiere benigne pentru care atributul dat este setat 1
- $totalClean$ - numărul total de înregistrări aparținând clasei de fisiere benigne
- $countMalicious$ - numărul de înregistrări aparținând clasei de fisiere malițioase pentru care atributul dat este setat 1
- $totalMalicious$ - numărul total de înregistrări aparținând clasei de fisiere malițioase

0.6.2 Modele hibride folosind Arborele Binar de Decizie si Clasificatorul OSCP

Mai întâi, arborele de decizie binar a fost combinat cu clasificatorul OSCP-Both Classes (Figura 4). Pe fiecare cluster un algoritm OSCP-BC a fost antrenat pentru 500 de iterații, folosind 500 de atribute, selectate cu funcția de scor F. Rata de detecție, rata alarme false, timpul de antrenament și amprenta de memorie au fost calculate pentru 11 adâncimi BDT (*Depth 0* ilustrează rezultatele obținute utilizând întregul set de date DB_CAT_BDT) și

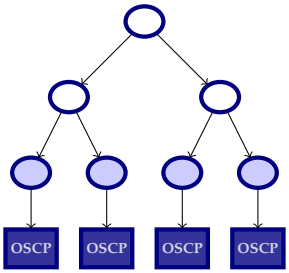

Figure 4: Arbore Binar de Decizie combinat cu modele OSCP

sunt exemplificate în Figura 5 Așa cum se poate observa din figura 5, metoda considerată îmbunătățește rata de detecție, dar există câteva observații care trebuie făcute în ceea ce privește acest experiment:

- Rata de detecție crește de la 29,78% pentru baza de date inițială la 92,62% atunci când se utilizează 1024 de clustere, ceea ce dă o îmbunătățire de aproximativ 62%.
- Rata de alarme false crește de 500 de ori, de la 0,006 % la 1,73 %. Ratele de alarme false de peste 1% nu sunt recomandate, deoarece devine foarte dificilă rezolvarea fișierelor legitime incorecte cu un proces de whitelisting.
- Dimensiunea modelului este, de asemenea, afectată și crește exponențial cu numărul de niveluri BDT utilizate.
- Performanța este, de asemenea, afectată. A fost măsurată în timp de antrenament și durează de la aproape 10 minute la jumătate de oră.

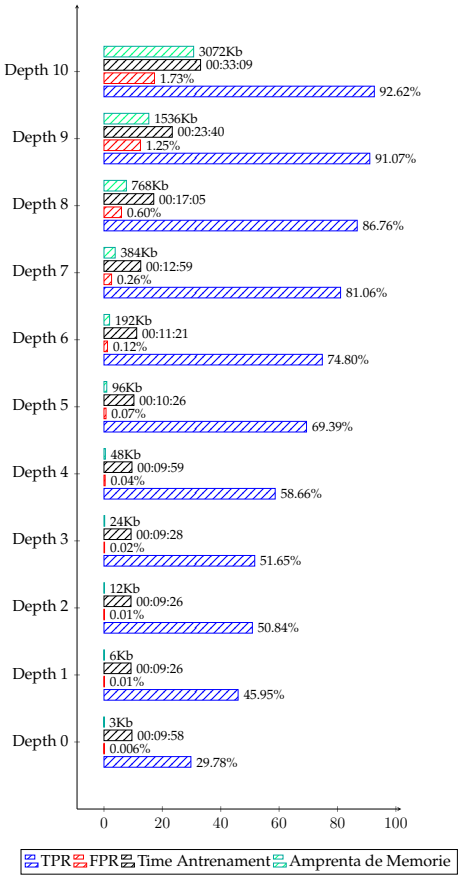

Figure 5: Rata de detecție, Rata de alarme false, Timp de Antrenament si Amprenta de Memorie pentru modelul hibrid bazat pe BDT si OSCP

0.6.3 Modele hibride folosind Arborele Binar de Decizie si Clasificatoare bazate pe Support Vector Machine

Următorul pas în studiul de față a fost evaluarea metodei de divizare a bazei de date utilizand un BDT cu diferite

clasificatoare bazate pe Support Vector Machine. Deoarece acesta a fost un studiu comparativ, o bibliotecă python numită *sklearn* a fost utilizată pentru implementarea modelelor SVM [4]. Au fost luate în considerare două tipuri de funcții kernel: Funcția liniară și Radial Basis (RBF). Fiecare model a fost antrenat și evaluat pe clusterele finale obținute prin divizarea BDT (Figura 6).

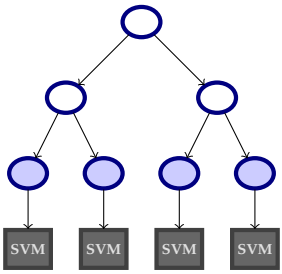

Figure 6: Arbore binar de decize combinat cu modele SVM

Deoarece este foarte dificil antrenarea modelelor bazate pe SVM pe seturi de date mari, seturile de date obținute cu divizările BDT folosind adâncimile 0, 1 și 2 nu au fost utilizate în acest experiment. În afară de funcția kernel, algoritmii au folosit un parametru C. O valoare mica pentru C face suprafața de decizie netedă, o valoare ridicata pentru C vizează clasificarea corectă a tuturor elementelor utilizate in antrenament. Implicit,

acest parametru este egal cu 1. Au fost testate trei astfel de valori: 0,1, 1 și 10. Mai întâi, rezultatele obținute de primii trei clasificatori (funcția kernel liniar) sunt ilustrate în figura 7. Au fost observate câteva fapte intere-

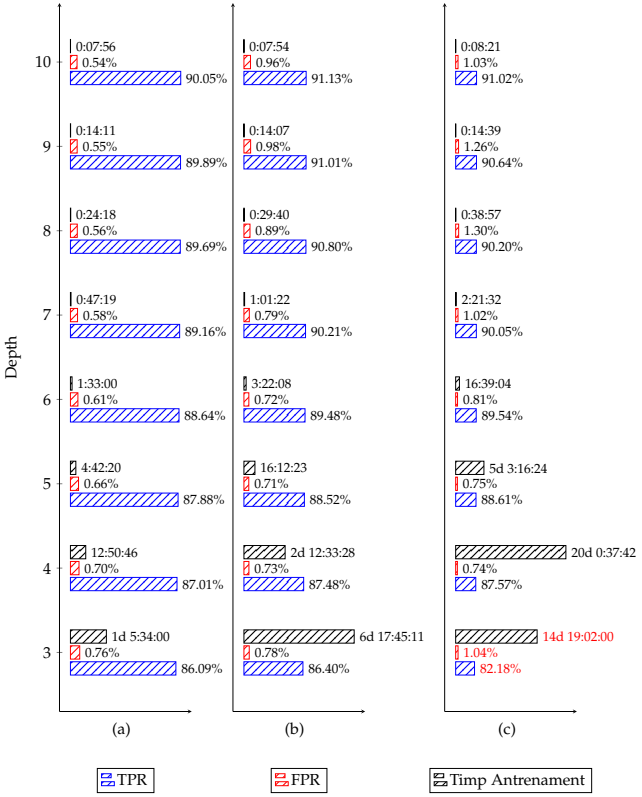

Figure 7: Rata de detectie, Rata de alarme false si Timpul de antrenament pentru modelul hibrid vazat pe BDT si SVM-liniar (C= 0.1 (a), C=1.0 (b), C=10.0 (c)

sante in analiza rezultatelor:

- Așa cum era de așteptat, a fost nevoie de mult timp pentru a antrena modelele SVM cu funcție de kernel liniar pe grupuri mari. Mai mult, cu cât este mai mare parametrul C, cu atât este nevoie de mai mult timp pentru antrenament. Antrenarea modelelor pe setul de date obținute prin divizarea BDT cu trei nivele nu a fost fezabilă, acest proces durând mai mult de o zi. Cel de-al treilea model ($C = 10$) a fost terminat forțat după 14 zile și 19 ore, deoarece după acest timp au fost generate și evaluate doar două modele. În acest fel, rezultatele corespunzătoare sunt evidențiate în roșu.
- Deși există o creștere a ratei de detecție corelată cu numărul de clustere, aceasta nu este la fel de considerabilă ca cea observată atunci când clasificatorul OSCP a fost testat în același mod.
- Parametrul C nu pare să influențeze foarte mult rata de detecție, pentru toate cele trei valori (0,1, 1, 10) rata de detecție rămâne între 86% și 91%.
- Nu în toate cazurile, modelele cu cea mai mare rată de detecție au, de asemenea, și cea mai mare rată de alarme false, așa cum se poate observa în experimentele care implică clasificatorul OSCP, iar parametrul C a influențat ratele de alarme false, cu atât este mai mare valoarea lui C, cu atât mai mari sunt procentele pentru alarmele false.

Aceleași experimente au fost efectuate pentru celelalte trei modele, utilizând funcția kernel RBF. Rezultatele obținute după antrenarea clasificatorului în cele 8 baze de date sunt evidențiate în Figura 8.

Figure 8: Rata de detectie, Rata de alarme false si Timpul de antrenament pentru modelul hibrid vazat pe BDT si SVM-RBF (C= 0.1 (a), C=1.0 (b), C=10.0 (c)

Au fost făcute mai multe observații cu privire la aceste rezultate:

- Deși timpul de antrenare depinde încă de dimensiunea setului de date, aceste modele s-au comportat mult mai bine decât cele care utilizează funcțiq kernel liniară. Seturile de date construite după împărțirea făcută prin BDT cu adâncimi trei și patru rămân încă imposibil de utilizat ân practică. Procesele de antrenament efectuate pe grupuri mai mici (adâncimi 8, 9, 10) au fost finalizate sub o oră.
- Se pare că, odată cu creșterea numărului de clustere, rata de detecție scade, spre deosebire de experimentele efectuate anterior în cazul în care cu cât este mai mare adâncimea arborelui de decizie, cu atât este mai mare rata de detecție. Mai mult, această scădere este influențată de valoarea parametrului C ($\approx$ 30% atunci când $C = 0.1$, $\approx$ 5% atunci când $C = 1$, $\approx$ 2 % când $C = 10$).
- Dacă anterior parametrul C nu părea să influențeze rata de detecție a modelelor, aici se poate vedea cu ușurință că modelul cu cea mai mare valoare pentru parametrul C are cele mai mari rate de detecție.
- Ratele de alarme false obținute de aceste modele sunt mai mici decât cele din experimentul anterior, majoritatea sub 0,5 %. De data aceasta, parametrul C nu pare să influențeze numărul de fișiere benigne clasificate incorect.

0.7 Modele hibride folosind Arborele Binar de Decizie și Învațarea Cascadată

După studierea acestor două metode de imbunatatire a ratei de detecție, următoarea etapă a cercetării a fost aceea de a combina divizarea BDT cu diferite abordări în cascadă studiate anterior pentru a vedea dacă rata de detecție poate fi îmbunătățită în continuare.

0.7.1 Modele hibride folosind Arborele Binar de Decizie și Clasificatorul in Cascadă

Mai întâi, pe fiecare cluster final obținut printr-o divizare BDT, a fost aplicată o abordare în cascadă așa cum se poate vedea în figura 9.

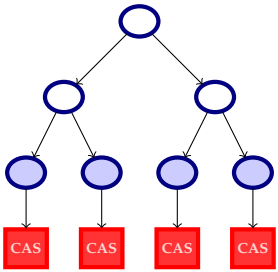

Figure 9: Arbore Binar de Decizie combinat cu clasificator in cascadă

0.7.2 Modele hibride ce intercaleaza un Arborele Binar de Decizie și Clasificatorul in Cascadă

Prezentul studiu a analizat o altă abordare a modului de combinare a tehnicii divizării BDT și a clasificatorului in cascadă. Ideea a început cu observația că numărul de înregistrări eliminate scade la fiecare etapă de cascadă. Mai exact, majoritatea elementelor au fost eliminate în timpul primei etape. Această metodă încearcă să rezolve această problemă prin intercalarea divizării BDT cu etapele în cascadă și este ilustrată în figura 10

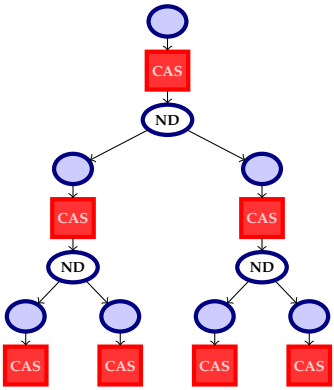

Figure 10: Intercalarea stagiilor unui clasificator in cascada cu divizarea BDT

Primul pas este aplicarea unei etape în cascadă pe setul de date inițial. Înregistrările clasificate corect de cele două modele ($Model_{clean}$ și $Model_{malicious}$) sunt eliminate,

astfel încât următoarele operații se vor concentra doar pe înregistrările pe care modelele anterioare nu le-au putut clasifica. Al doilea pas este aplicarea divizării BDT pe subsetul clasificat incorect returnat de algoritmul anterior. Nodurile frunză (grupurile finale) vor fi apoi trimise ca parametru algoritmului anterior. Rezultatele obținute sunt ilustrate în Tabelul 2

Algorithm	FP Rate	Detection Rate	Training Time
OSCP-BC	0.006 %	29.78 %	2:05:13
CAS	0.015 %	34.96 %	3:09:24
BDT3-CAS	0.109 %	59.94 %	4:16:12
BDT4-CAS	0.261 %	70.07 %	4:48:08
BDT4-CAS-3000	0.508 %	76.46 %	6:56:02
BDT4-CAS-200F-200IT	0.599 %	83.10 %	5:11:00
CAS-BDT1-200F-200IT	0.869 %	84.02 %	5:15:12

Table 2: Rezultatele obținute in experimentele practice pentru algoritmii ce conbiba Arorele binar de decizie și învatarea cascadată

0.8 Concluzii

Studiul de față propune diferite soluții hibride destinate creșterii ratei de detecție, dar și o analiză detaliată a restricțiilor practice care trebuie luate în considerare la crearea unui mecanism de protecție ce trebuie integrat într-un produs de securitate. Prima abordare se referă la învățarea

în cascadă, unde mai multe clasificatoare sunt combinate astfel încât să se elimine înregistrările clasificate corect, astfel încât următoarul model să se poată concentra numai asupra acelor elemente care sunt dificil de clasificat. Cea de-a doua abordare este un ansamblu care combină o divizare bazată pe arbori de decizie binari cu diferite tipuri de clasificatori. S-a studiat, de asemenea, o combinație între aceste două metode.

Toate modelele sunt antrenate și testate pe un set de date cu mai mult de două milioane de înregistrări. În timpul experimentelor practice sunt analizați patru indicatori de performanță importanți: rata de detecție, rata de alarme false, amprenta de memorie și timpul de evaluare. Astfel de indicatori sunt foarte importanți, mai întâi pentru a vedea dacă abordările sunt adecvate pentru utilizarea practică, dar și pentru a putea integra soluțiile în anumite categorii. Este important să știm care dintre indicatori au o limită strictă pentru a identifica corect tehnologiile complementare, astfel încât să poată fi asigurată o protecție adecvată.

0.9 Appendix: Analiza indicatorilor de performanta pentru toti algoritmii studiati

Ultima secțiune colectează toate rezultatele experimentale obținute în timpul acestui studiu și să evaluează indicatorii de performanță pentru toți algoritmii prezenți în aceast studiu: rata de detecție, rata de alarme false, amprenta de memorie și impactul performanței (măsurată ca timp de evaluare/testare).

Indicatori

Terminologie

- Positive - P: numărul de elemente malitioase
- Negative - N: numărul de elemente benigne
- True Positive - TP: numărul de elemente malitioase identificate ca fiind malitioase de către un clasificator
- True Negative - TN: numărul de elemente benigne identificate ca fiind ligitime de către un clasificator
- False Positives - FP: numărul de elemente benigne identificate ca fiind malitioase de către un clasificator
- False Negatives - FN: numărul de elemente malitioase identificate ca fiind legitime de un clasificator

Rata de detecție

Rata de detecție sau True Positive Rate este definită ca raportul dintre numarul de elemente pozitive corect clasificate și numărul total de pozitive:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Rata de alarme false

Rata de alarme false este definita ca raportul dintre numarul de elemente negative incorect clasificate si numarul total de negative:

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

Amprenta de memorie (dimensiunea modelului)
Amprenta de memorie este definită ca dimensiunea modelului sau cantitatea de spațiu de stocare necesară memorării unui model. Acest indicator depinde de numărul de atribute utilizate. Fie m numărul de atribute utilizate de clasificatorul C și N numărul de vectori de suport în cazul clasificatorilor Vector Machine Support. Fiecare atribut are un ID (2 octeti) și o valoare corespunzătoare (4 octeți). Luând în considerare acest lucru, pot fi definite următoarele:

- $MF_{OSCP} = 6 \times m$
- $MF_{OSCP-BC} = 2 \times (6 \times m) = 12 \times m$
- $MF_{CAS} = noStages \times 12 \times m$
- $MF_{CAS-kIT} = noStages \times (12 \times m)$
- $MF_{CAS-zF} = \sum_{i=0}^{noStages-1} (m + i \times z) \times 12$
- $MF_{CAS-zF-kIT} = \sum_{i=0}^{noStages-1} (m + i \times z) \times 12$
- $MF_{BDTx-OSCP} = 2^x \times (6 \times m)$
- $MF_{BDTx-SVM-linear} = 2^x \times (6 \times m)$
- $MF_{BDTx-SVM-rbf} = 2^x \times (N \times m \times 6 + N \times 6 + N \times 1)$
  $= 2^x \times N \times (6 \times m + 7)$
- $MF_{BDTx-CAS} = 2^x \times (noStages \times (12 \times m))$
- $MF_{BDTx-CAS-3000} = 2^x \times (noStages \times (12 \times m))$
- $MF_{BDTx-CAS-zF-kIT} = 2^x \times (\sum_{i=0}^{noStages-1} (m + i \times z) \times 12)$
- $MF_{CAS-BDTx-zF-kIT} = \sum_{j=0}^{x} 2^j \times (\sum_{i=0}^{noStages-1} (m + i \times z) \times 12)$

Impactul de Performanță

Este definit în prezenta cercetare prin timpul de evaluare, adică timpul necesar pentru a efectua tot calculul necesar pentru a lua o decizie cu privire la o înregistrare, dacă aparține clasei benigne sau clasei de fișiere malitioase.

Timpul necesar pentru efectuarea unei operații diferă de la o operație la alta, dar pentru a simplifica calculul, în studiul de față se are în vedere:

$$T_{mul} = T_{add} = T_{compare} = T_{assign} = \Delta$$

Timpul de evaluare pentru modelele SVM va fi calculat în funcție de numărul mediu de vectori de suport per grup. Următorii timpi de evaluare pot fi definiti pentru fiecare model studiat:

- $ET_{OSCP} = m \times \Delta + 1$
- $ET_{OSCP-BC} = 2 \times (m \times \Delta + 1)$
- $ET_{CAS} = noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{CAS-kIT} = noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{CAS-zF} = \sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$
- $ET_{CAS-zF-kIT} = \sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$
- $ET_{BDTx-OSCP} = x \times \Delta + m \times \Delta + 1 = \Delta \times (m + x) + 1$
- $ET_{BDTx-SVM-linear} = x \times \Delta + m \times \Delta + 1 = \Delta \times (m + x) + 1$
- $ET_{BDTx-SVM-rbf} = x \times \Delta + N \times (2 \times \Delta + m \times \Delta + 3 \times \Delta) + \Delta$
  $= \Delta \times (N \times (m + 5) + x + 1)$
- $ET_{BDTx-CAS} = x \times \Delta + noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{BDTx-CAS-3000} = x \times \Delta + noStages \times (2 \times (m \times \Delta + 1))$
- $ET_{BDTx-CAS-zF-kIT} = x \times \Delta + \sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$
- $ET_{CAS-BDTx-zF-kIT} = x \times \Delta + x \times (\sum_{i=0}^{noStages-1} ((m + i \times z) \times \Delta + 1) \times 2$

Având toți indicatorii definiți pentru fiecare model studiat, tabelul 3 rezumă toate rezultatele:

Algoritm	TPR	FPR	MF (octeti)	ET(Nr. Operații)
OSCP	29.78 %	0.006 %	3000	$500 \times \Delta + 1$
OSCP-BC	29.78 %	0.006 %	6000	$1000 \times \Delta + 2$
CAS	34.96 %	0.015 %	60000	$10000 \times \Delta + 20$
CAS-100F	45.93 %	0.033 %	114000	$19000 \times \Delta + 20$
CAS-200F	49.92 %	0.037 %	168000	$28000 \times \Delta + 20$
CAS-300F	49.46 %	0.034 %	222000	$37000 \times \Delta + 20$
CAS-400F	48.54 %	0.029 %	276000	$46000 \times \Delta + 20$
CAS-100IT	36.17 %	0.022 %	60000	$10000 \times \Delta + 20$
CAS-200IT	36.69 %	0.024 %	60000	$10000 \times \Delta + 20$
CAS-200F-200IT	56.87 %	0.053 %	168000	$28000 \times \Delta + 20$
BDT1-OSCP	45.95 %	0.011 %	6000	$501 \times \Delta + 1$
BDT2-OSCP	50.84 %	0.013 %	12000	$502 \times \Delta + 1$
BDT3-OSCP	51.65 %	0.022 %	24000	$503 \times \Delta + 1$
BDT4-OSCP	58.66 %	0.045 %	48000	$504 \times \Delta + 1$
BDT5-OSCP	69.39 %	0.072 %	96000	$505 \times \Delta + 1$
BDT6-OSCP	74.80 %	0.125 %	192000	$506 \times \Delta + 1$
BDT7-OSCP	81.06 %	0.261 %	384000	$507 \times \Delta + 1$
BDT8-OSCP	86.76 %	0.603 %	768000	$508 \times \Delta + 1$
BDT9-OSCP	91.07 %	1.251 %	1536000	$509 \times \Delta + 1$
BDT10-OSCP	92.62 %	1.732 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-linear-C0.1	86.09 %	0.76 %	24000	$503 \times \Delta + 1$
BDT4-SVM-linear-C0.1	87.01 %	0.70 %	48000	$504 \times \Delta + 1$
BDT5-SVM-linear-C0.1	87.88 %	0.66 %	96000	$505 \times \Delta + 1$
BDT6-SVM-linear-C0.1	88.64 %	0.61 %	192000	$506 \times \Delta + 1$
BDT7-SVM-linear-C0.1	89.16 %	0.58 %	384000	$507 \times \Delta + 1$
BDT8-SVM-linear-C0.1	89.69 %	0.56 %	768000	$508 \times \Delta + 1$
BDT9-SVM-linear-C0.1	89.89 %	0.55 %	1536000	$509 \times \Delta + 1$
BDT10-SVM-linear-C0.1	90.05 %	0.54 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-linear-C0.1	86.40 %	0.78 %	24000	$503 \times \Delta + 1$
BDT4-SVM-linear-C1.0	87.48 %	0.73 %	48000	$504 \times \Delta + 1$

Algoritm	TPR	FPR	MF (octeti)	ET(Nr. Operații)
BDT5-SVM-linear-C1.0	88.52 %	0.71 %	96000	$505 \times \Delta + 1$
BDT6-SVM-linear-C1.0	89.48 %	0.72 %	192000	$506 \times \Delta + 1$
BDT7-SVM-linear-C1.0	90.21 %	0.79 %	384000	$507 \times \Delta + 1$
BDT8-SVM-linear-C1.0	90.80 %	0.89 %	768000	$508 \times \Delta + 1$
BDT9-SVM-linear-C1.0	91.01 %	0.98 %	1536000	$509 \times \Delta + 1$
BDT10-SVM-linear-C1.0	91.13 %	0.96 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-linear-C10.1	82.18 %	1.04 %	24000	$503 \times \Delta + 1$
BDT4-SVM-linear-C10.1	87.57 %	0.74 %	48000	$504 \times \Delta + 1$
BDT5-SVM-linear-C10.1	88.61 %	0.75 %	96000	$505 \times \Delta + 1$
BDT6-SVM-linear-C10.1	89.54 %	0.81 %	192000	$506 \times \Delta + 1$
BDT7-SVM-linear-C10.1	90.05 %	1.02 %	384000	$507 \times \Delta + 1$
BDT8-SVM-linear-C10.1	90.20 %	1.30 %	768000	$508 \times \Delta + 1$
BDT9-SVM-linear-C10.1	90.64 %	1.26 %	1536000	$509 \times \Delta + 1$
BDT10-SVM-linear-C10.1	91.02 %	1.03 %	3072000	$510 \times \Delta + 1$
BDT3-SVM-rbf-C0.1	76.80 %	0.48 %	487687288	$10237869 \times \Delta$
BDT4-SVM-rbf-C0.1	73.77 %	0.43 %	525479264	$5515615 \times \Delta$
BDT5-SVM-rbf-C0.1	70.57 %	0.42 %	566759360	$2974456 \times \Delta$
BDT6-SVM-rbf-C0.1	67.17 %	0.45 %	608135680	$1595807 \times \Delta$
BDT7-SVM-rbf-C0.1	63.10 %	0.45 %	655092992	$859518 \times \Delta$
BDT8-SVM-rbf-C0.1	59.41 %	0.48 %	715906560	$469659 \times \Delta$
BDT9-SVM-rbf-C0.1	53.54 %	0.41 %	783648256	$257055 \times \Delta$
BDT10-SVM-rbf-C0.1	45.37 %	0.31 %	856008704	$140401 \times \Delta$
BDT3-SVM-rbf-C1.0	85.19 %	0.43 %	332189304	$6973549 \times \Delta$
BDT4-SVM-rbf-C1.0	84.56 %	0.42 %	351073264	$3684990 \times \Delta$
BDT5-SVM-rbf-C1.0	83.78 %	0.41 %	373637792	$1960921 \times \Delta$
BDT6-SVM-rbf-C1.0	82.54 %	0.38 %	398559808	$1045862 \times \Delta$
BDT7-SVM-rbf-C1.0	81.11 %	0.37 %	430313728	$564598 \times \Delta$
BDT8-SVM-rbf-C1.0	79.28 %	0.37 %	470342912	$308564 \times \Delta$
BDT9-SVM-rbf-C1.0	77.17 %	0.37 %	518839808	$170195 \times \Delta$
BDT10-SVM-rbf-C1.0	74.63 %	0.38 %	581962752	$95456 \times \Delta$
BDT3-SVM-rbf-C10.1	90.57 %	0.33 %	241281680	$5065154 \times \Delta$
BDT4-SVM-rbf-C10.1	90.25 %	0.35 %	251770096	$2642670 \times \Delta$

Algoritm	TPR	FPR	MF (octeti)	ET(Nr. Operații)
BDT5-SVM-rbf-C10.1	90.08 %	0.36 %	262402848	$1377141 \times \Delta$
BDT6-SVM-rbf-C10.1	89.77 %	0.37 %	274623296	$720642 \times \Delta$
BDT7-SVM-rbf-C10.1	89.54 %	0.38 %	294830336	$386838 \times \Delta$
BDT8-SVM-rbf-C10.1	89.25 %	0.37 %	324082432	$212614 \times \Delta$
BDT9-SVM-rbf-C10.1	88.88 %	0.37 %	364881408	$119695 \times \Delta$
BDT10-SVM-rbf-C10.1	88.43 %	0.36 %	421846016	$69196 \times \Delta$
BDT3-CAS	59.94 %	0.109 %	480000	$10003 \times \Delta + 20$
BDT4-CAS	70.07 %	0.261 %	960000	$10004 \times \Delta + 20$
BDT4-CAS-3000	76.46 %	0.508 %	960000	$10004 \times \Delta + 20$
BDT4-CAS-200F-200IT	83.10 %	0.590 %	2688000	$28004 \times \Delta + 20$
CAS-BDT1-200F-200IT	84.02 %	0.869 %	3066000	$8008 \times \Delta + 16$

Table 3: Valorile obtinute pentru indicatorii de performanta in experimentele practice pentru toți algoritmii studiați

[1] Dragos Gavrilut, Razvan Benchea, and Cristina Vatamanu. Optimized zero false positives perceptron training for malware detection. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 247–253, 2012. doi: 10.1109/SYNASC.2012.34. URL https://doi.org/10.1109/SYNASC.2012.34.

[2] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 386–407, 1958.

[3] Stanley Ng Kwang Loong and Santosh K. Mishra. De novo SVM classification of precursor microRNAs from genomic pseudo hairpins using global and intrinsic folding measures. *Bioinformatics/computer Applications in The Biosciences*, 23:1321–1330, 2007. doi: 10.1093/bioinformatics/btm026.

[4] http://scikit-learn.org/stable/modules/svm.html#.