UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Sistem inteligent de vizualizare a atacurilor DDoS bazat pe entropie și arbori de decizie

propusă de

Victor-Ciprian Roșca

Sesiunea: iulie, 2025

Coordonator științific

Lect. dr. Eugen Nicolae Croitoru

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Sistem inteligent de vizualizare a atacurilor DDoS bazat pe entropie și arbori de decizie

Victor-Ciprian Roșca

Sesiunea: iulie, 2025

Coordonator științific

Lect. dr. Eugen Nicolae Croitoru

Avizat,
Îndrumător lucrare de licență,
Lect. dr. Eugen Nicolae Croitoru.
Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Roșca Victor-Ciprian** domiciliat în **România, jud. Iași, mun. Iași, calea Buzăului, nr. 25, bl. A, et. 5, ap. 45**, născut la data de **01 ianuarie 2018**, identificat prin CNP **1234567891234**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2018, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Sistem inteligent de vizualizare a atacurilor DDoS bazat pe entropie și arbori de decizie** elaborată sub îndrumarea domnului **Lect. dr. Eugen Nicolae Croitoru**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Sistem inteligent de vizualizare a atacurilor DDoS bazat pe entropie și arbori de decizie**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Victor-Ciprian Roșca**

Data: ..... Semnătura: .....

Cuprins

Abstract

Lucrarea prezintă elaborarea unei aplicații software interactive pentru vizualizarea traficului din rețea folosind arbori de decizie modelați pe baza entropiei, având ca intenție evidențierea pattern-urilor anormale, precum atacurile DDoS. Programul este structurat modular și asigură atât analiza în timp real a datelor, cât și interpretarea vizuală cu alegerile luate de modelul de clasificare.

De asemenea, programul asigură condițiile pentru descifrarea felului în care modelul de învățare automată clasifică traficul și oferă o unealtă vizuală valoroasă ca să depisteze atacurile cibernetice din rețele.

Capitolul 1

Introducere

1.1 Ideea și scopul proiectului

Ideea proiectului “ddos Visual Detect” este pentru a combina detectarea atacurilor ddos cu o componentă vizuală interactivă. Această vizualizare servește drept o imagine clară și de ansamblu asupra datelor.

Precondițiile proiectului includ existența unui set de date (csv) pentru trafic de rețea pentru antrenarea și testarea modelului, prezența unui generator de log-uri ce va alimenta componenta vizual dinamică și minimul pentru mediul dezvoltării este Python3 cu suport OpenGL (versiunea minim 3).

Funcționalitatea aplicației pornește de la modelul ML(arbore de dicizie) care preia datele pentru antrenament, calculează entropia, statistici și apoi clasifică ca ”BENIGN” sau ”ddos” iar după vizualizarea în OpenGL afișează arborele de decizie.

Scopul proiectului pe termen scurt a fost generarea arborelui de decizie și dezvoltarea scenei OpenGL cu iluminarea dinamică a path-ului iar viziunea pe termen lung constă în extinderea la alte modele ML și la vizualizarea altor tipuri de atacuri cibernetice.

1.2 Motivație

Învațarea Automata (ML) și multe modele tehnice, considerate ansambluri tehnice complexe, au facut progres remarcabil în domeniul inteligenței artificiale. [10] Pe viitor se asteaptă o transformare rapidă pentru domeniului IA (inteligența artificială), asta va aduce plus valoare în diferite industrii și zone economice. [12]

Pe de altă parte, datorită avansurilor pentru domeniul digital crește progresul atacurilor cibernetice pe zi ce trece. [14] Orice în ziua de astăzi ce utilizează date digitale (precum zone economice) este supus riscului de a fi victima unui atac cibernetic. [13]

La fel, a crescut numărul produselor pe IA (inteligența artificiala) care oferă servicii pe domeniul securității informatice. [11] Totuși, numărul acestor produse ce au vulnerabilități la atacuri cibernetice rămâne în creștere.

Fiind pasionat pe domeniul securității informatice și de tehnologiile IA (ML), am hotărât particip la creșterea acestor sectoare. cu acest proiect ce are la baza învațarea supervizată prin care un algoritm învață să clasifice entități folosind date cunoscute cu rezultate anterior stabilite.[8]

1.3 Structura proiectului

Proiectul a fost structurat în patru module cu funcționalități diferite:

1. Scriptul pentru crearea seturilor de date (train/test) pornind de la fișierul principal ce conține toate informațiile de trafic numit final_dataset.csv iar pe baza acestui csv se generează fișierele train.csv și test.csv ce vor fi importate în modelul ML.

2. Modulul de generare a log-urilor extrage aleator date din dataset-ul de trafic (final_dataset.csv) și creează câte un fișier log o dată la 10 secunde.

3. Modulul de modelare ML construiește un arbore de decizie binar folosind algoritm-ul ID3. De asemenea tot în cadrul acestui modul se află și funcționalitatea de monitorizare și integrare real-time ce utilizează watchdog pentru a supraveghea folderul logs în care sunt stocate fișierele .data generate de modulul 2.

4. Modulul de vizualizare in OpenGL este pornit pe un thread separat pentru randare permițând actualizări independente de fluxul principal.

1.4 Structura lucrării

Lucrarea este compusă din 7 capitole:

Capitolul 1 (Introducere și motivație): se prezintă cererea detectării atacurilor DDoS în contextul securității cibernetice;

Capitolul 2 (Explorarea datelor cu Weka): include istoricul dinamicii și designul

modular al Weka, componentele sale de bază, format ARFF și suport CSV/JSON, clasificatorii ID3 și J48.

Capitolul 3 (Tehnologii și instrumente utilizate): Descrie mediul de dezvoltare și bibliotecile folosite pentru reprezentarea arborilor, monitorizarea fișierelor, partea vizuală a proiectului cât și generarea și preprocesarea seturilor de date.

Capitolul 4 (Arhitectura proiectului): se detaliază interacțiunea dintre log-generator, arborele de decizie și partea vizuală din OpenGL.

Capitolul 5 (Detalii de implementare): exemplificări de cod și explicații pentru principalele funcționalități.

Capitolul 6 (Interfața și scenarii de utilizare): descrie platforma vizuală OpenGL și cum se desfășoară în timp real, cu capturi ale ecranului din demo-urile efectuate.

Capitolul 7 (Concluzii și lucrări viitoare): rezumă contribuțiile proiectului, performanța softului și eventuale extinderi.

Capitolul 2

Arhitectura și Evoluția Weka

2.1 Scurt istoric

Weka este un software open-source destinat analizei datelor și învățării automate, dezvoltat inițial în cadrul Universității din Waikato, Noua Zeelandă, începând cu anul 1993.[1] Proiectul a fost creat de o echipă coordonată de profesorul Ian H. Witten, pentru a genera un mediu accesibil pentru experimente în domeniul inteligenței artificiale și al mineritului datelor. [2]

Versiunea inițială a fost scrisă în limbajul C, după care proiectul să fie complet rescris în Java pentru a ajunge independent de platformă și mai ușor de folosit în mediile educaționale. [1] O componentă importantă este implementarea algoritmului bazat pe arbori de decizie, precum ID3 și J48 (versiune îmbunătățită pentru algoritmul C4.5). [3]

Weka a fost conceput ca un instrument modular, permițând utilizatorilor să încarce seturi de date, să aplice filtre și să experimenteze cu diferiți algoritmi de clasificare, regresie și clustering. [2] Cu timpul, prin utilizarea inițială ca un simplu set de instrumente pentru analiză științifică, a avut loc o conversie către o platformă completă folosită în multe instituții academice și chiar produse comerciale. [4]

Deși interfața sa vizuală este simplă, Weka a devenit un punct de referință în învățarea automată, fiind folosit frecvent în scopuri educaționale pentru a ilustra concepte precum entropia, câștigul informațional și structura arborilor de decizie. [1] Prin caracterul său vizual și interactiv, a inspirat numeroase demersuri viitoare, precum și programe software dedicate înțelegerii vizuale a procesului decizional, echivalente din perspectiva abordării cu proiectul prezent.

2.2 Caracteristici

Weka oferă o platformă modulară și extensibilă pentru procesarea datelor și utilizarea algoritmilor de învățare automată, fiind concepută în jurul unei interfețe grafice prietenoase, precum și a componentelor expuse prin linie de comandă sau API Java. [5] Interfața grafică este împărțită în mai multe secțiuni (Explorer, Experimenter, Knowledge Flow și Simple CLI), fiecare facilitând un stil diferit de analiză și testare a algoritmilor. [1]

O caracteristică centrală a Weka este facilitarea lucrului cu fișiere de date în format ARFF (Attribute-Relation File Format), format specific acestui software, care detaliază complet atributele, valorile permise și etichetele de clasă. [6] Platforma asigură și compatibilitate cu formate CSV, JSON sau baze de date relaționale prin JDBC. [6]

Arborii de decizie, printre care și ID3, pot fi folosiți direct asupra datelor încărcate, iar outputul este expus atat textual (sub formă de reguli logice sau structură de arbore), cat și vizual, ceea ce sustine înțelegerea logicii deciziei. [5] Utilizatorii pot modifica în timp real parametrii algoritmilor sau pot experimenta cu tehnici de validare încrucișată, filtrare a datelor sau selecție de atribute. [5]

Weka dispune de o arhitectură extensibilă, permițând dezvoltatorilor să adauge noi algoritmi sau filtre prin plugin-uri Java. Totodată, pachetele externe pot fi instalate automat printr-un manager integrat. [5] Aceasta permite integrarea rapidă a unor funcționalități suplimentare precum clasificatoare ensemble, metode de reducere dimensională sau vizualizatori avansați. [5]

Deși orientat inițial catre limbajul Java, Weka poate fi folosit și în colaborare cu alte limbaje, datorită proiectelor de legatura cu Python (prin pachetul python-weka-wrapper) sau R. [5] Adresat utilizatorilor avansați, acest program software permite automatizarea completă a sarcinilor prin scripturi și integrarea într-un pipeline de analiză mai larg. [5]

La nivelul de randare și interactivitate, Weka oferă suport pentru vizualizarea distribuțiilor de date, decizii generate de algoritmi și performanțe ale instrumentelor aplicate, prin vizualizări dinamice sau matrice de confuzie. [6] Interfața permite compararea mai multor clasificatori simultan, inclusiv în condiții controlate de testare automată. [6]

Capitolul 3

Tehnologii și instrumente utilizate

3.1 Limbaj de programare

Limbajul de programare de nivel înalt ales este Python, folosit pentru prelucrarea datelor, implementarea algoritmilor pentru învățarea automată (ML) și interacțiuni cu structura de fișiere.

3.2 Biblioteci pentru prelucrarea și analiza datelor

pandas utilizat pentru manipularea tabelelor de date (DataFrame-uri).

re – procesează expresiile regulate (extragerea valorilor din șiruri).

numpy pentru calcule numerice (folosit la calculul entropiei).

glob – listează fișierele din directoare.

os pentru operații pe sistemul de fișiere.

subprocess – rulează comenzile shell (ex: generarea imaginilor).

PyOpenGL utilizat pentru interfața vizuală a proiectului plus faptul că garantează interfața cu placa video:

GL desenează forme (noduri dreptunghiulare, linii) și text scalat.

GLUT creează fereastra și detectează input-ul utilizatorului (ex.: rotirea scroll-ului pentru zoom).

GLU poate sprijini operațiuni de transformare a proiecției (dacă ar fi cerute).

threading - permite funcționarea simultană (concurentă) a două componente care în mod normal ar bloca execuția:

loop de evenimente OpenGL/GLUT care redă arborele și răspunde la input.

bucla de monitorizare a folderului logs cu biblioteca watchdog care așteaptă fișiere noi și le procesează pe masură ce apar.

time oferă o periodicitate de 10 secunde între creațiile consecutive de fișiere de log.

datetime pentru obținerea unui timestamp unic și lizibil (data și ora curentă) în momentul creării fișierului log.

pathlib dedicat reprezentării căii către fișiere.

csv acest format pentru stocarea datelor a fost ales pentru că oferă rapiditate de implementare, compatibilitate imediată cu pandas și portabilitate.

anytree exportă structura internă a arborelui într-un fisier DOT, care apoi, cu ajutorul Graphviz, se transformă într-o imagine PNG.

3.3 Învațare Automată & algoritmi

Având în vedere complexitatea sarcinii, am ales un arbore decizional pentru că această structură de date face ca fiecare creangă să corespundă unui rezultat potențial pentru un set dat de intrări, facilitând împărțirea problemelor dificile în etape mai ușor de controlat. [9]

ID3 (Iterative Dichotomiser 3) implementare personalizată a unui algoritm de construcție a arborelui decizional bazat pe entropie.

Entropie concept din teoria informației folosit pentru determinarea celui mai bun atribut de împărțire a datelor. [7]

Split binar abordare pentru împărțirea atributelor continue folosind puncte de tăiere calculate. [6]

Metoda principală utilizată este învățarea supervizată, care presupune utilizarea unor date de antrenament etichetate astfel încât fiecare exemplu are asociat predicția corectă. [6]

3.4 Structuri de date

Arbori binari de decizie – creați recursiv pe baza entropiei minime.

Capitolul 4

Arhitectura proiectului

4.1 Fluxul de date și interacțiunea componentelor

Proiectul este organizat într-o arhitectură modulară clar separată în patru componente funcţionale, fiecare având responsabilităţi bine definite:

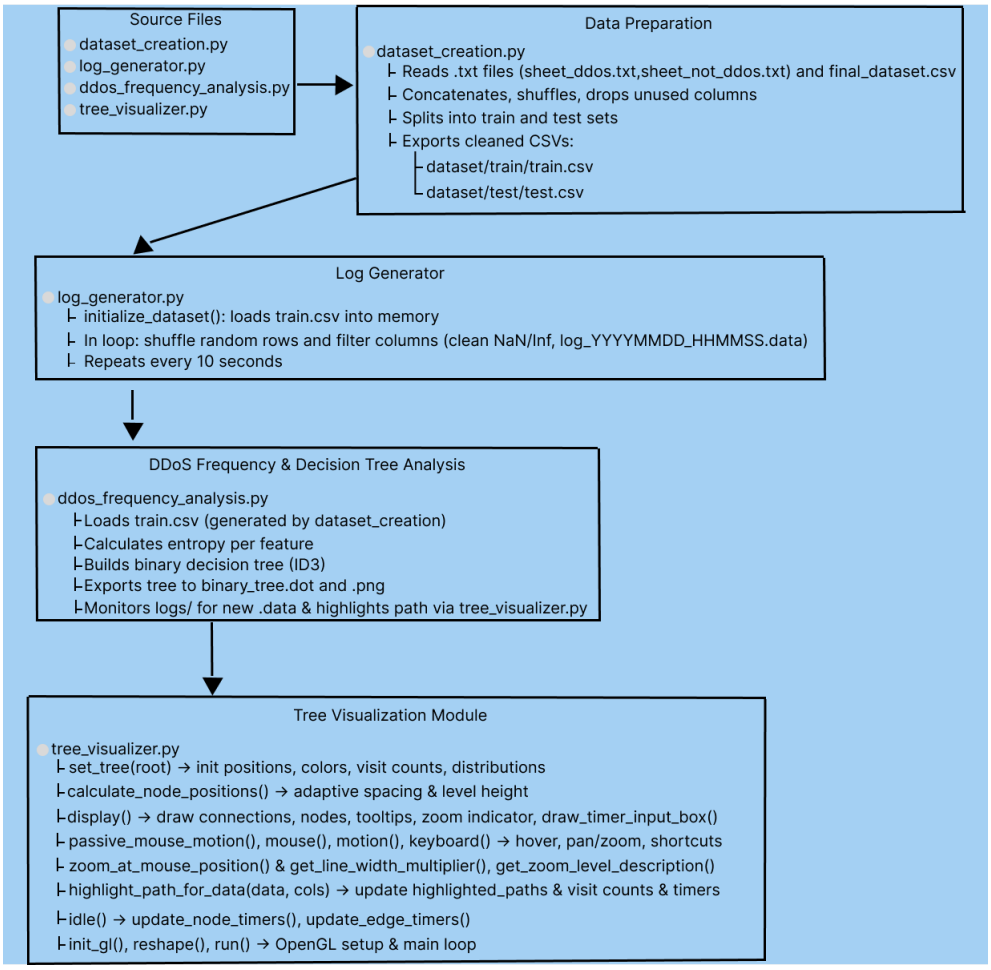

Figura 4.1: Interacţiunea Modulelor

4.2 Data Preparation

Modulul care citeşte şi curăţă datele brute pentru antrenarea şi testarea modelului pentru detecţie DDoS.

Din cadrul dataset_creation.py se încarcă fişierele sheet_ddos.txt, sheet_not_ddos.txt şi final_dataset.csv. (Datasetul final a fost descărcat de pe internet, în timp ce fișierele .txt au fost generate utilizând scriptul dataset_creation.py)
Se concatenează, amestecă (shuffle) şi şterge coloanele nesemnificative. Se împarte în seturi pentru antrenament şi test şi exportă CSV-urile curate către path dataset/train/train.csv şi dataset/test/test.csv.

4.3 Log Generation

Componentă pentru simulare a fluxului de date în timp real.
log_generator.py porneşte prin încărcarea train.csv în memorie.
La fiecare 10 secunde selectează aleatoriu câteva zeci de înregistrări, elimină coloanele inutile, filtrează NaN/Inf şi scrie un fişier log_YYYYMMDD_HHMMSS.data.
Continuă ciclic, populând directorul logs/ cu date noi privind reţeaua.

4.4 DDoS Frequency & Decision Tree Analysis

Modulul de construire şi actualizare a modelului decizional pe baza frecvenţei evenimentelor.
ddos_frequency_analysis.py încarcă fişierele train.csv sau test.csv conform contextului.
Calculează entropia fiecărui atribut (pentru ID3), selectează split-uri optime şi realizează arborele binar de decizie.
Exportă structura în binary_tree.dot şi imagine PNG.
Monitorizează logs/ şi, la descoperirea fişierelor noi, extrage fiecare linie de date şi determină calea de decizie corespunzătoare.

4.5 Tree Visualization Module

Interfaţa OpenGL care afișează arborele şi evidenţiază în timp real calea de clasificare.

tree_visualizer.py primeşte rădăcina arborelui (set_tree) şi calculează poziţii adaptive pentru fiecare nod.

În callback-ul display() desenează întâi conexiunile (linia de decizie), apoi nodurile (cu culori şi contori de vizite), afişează tooltip-uri la hover, ajutor şi indicator de zoom.

Gestionează interacţiunea: hover (passive_mouse_motion), pan/zoom (mouse, motion, zoom_at_mouse_position), scurtături de la tastatură (keyboard).

În highlight_path_for_data() primeşte datele din log, parcurge arborele pentru a descoperi calea de decizie, incrementează contorii de vizite şi redesenează vizualizarea pentru a reflecta traseul.

Acest modul conține și funcționalitatea legată de timer, care face ca după un anumit timp culorile nodurilor și muchiilor să revină la culoarea inițială, adică alb.

4.6 Fluxul de procesare și integrarea modulelor

Fiecare modul colaborează astfel:

Datele sunt pregatite după cum urmează:

1. Generatorul emite log-uri.

2. Componenta de analiză construiește și actualizează arborele.

3. Stadiul randării OpenGL se ocupă cu afișarea și evidențierea deciziilor pentru orice eveniment.

Capitolul 5

Detalii de implementare

5.1 Implementarea modulului Data Preparation

Stabilirea cadrului de lucru:

Fișierul final_dataset.csv a fost descărcat de la adresa:

https://www.kaggle.com/datasets/devendra416/ddos-datasets/data

Se descarcă fișierul zip de 3GB se dezarhivează iar astfel vom obține un director numit archive care conține două foldere: unul numit ddos_imbalanced și celălalt numit ddos_balanced care include fișiesul final_dataset.csv de dimensiune 6.32 GB.

Fișierul final_dataset.csv reprezintă setul inițial de date brute, care conține informații despre traficul de rețea, incluzând atât datele etichetate ca ddos, cât și benign. Acesta este utilizat ca sursă principală din care se extrag și analizează datele utile pentru crearea mulțimilor de antrenament și testare.

Transformăm un fișier mare pentru trafic de rețea, cu etichete ”ddos” și ”Benign” în două seturi proporționale pentru antrenament(train.csv) și test(test.csv). Datele pentru învațare vor fi salvate în fișierul train.csv din folderul train aflat în directorul dataset iar datele de testare se vor stoca în fișierul test.csv din folderul test, amplasat în același director.

Din fișierul final_dataset.csv extragem aleatoriu 30000 de randuri ce au eticheta (Label) ”ddos” pe care le stocăm în fișierul sheet_ddos.txt. La fel pentru clasa ”benign” similar se preiau la întamplare 30000 de linii care vor fi stocate în fișierul sheet_not_ddos.txt. Astfel sunt două seturi cu aceeași mărime.

Acum pentru a genera train.csv vom utiliza cele 2 seturi create anterior. Inserăm în train.csv 10000 de rânduri din fișierul sheet_ddos.txt pornind de la linia 1 până la 10000

apoi se adaugă în train.csv 10000 de linii din fișierul sheet_not_ddos.txt. Momentan pentru train.csv se amestecă (shuffle) liniile cu date pentru a dispersa aleator probele celor două clase.
Deoarece fișierul final_dataset.csv conține multe coloane care nu sunt relevante pentru analiza atacului DDoS, am salvat într-o listă acele coloane pe care dorim să le ștergem. Lista este generată manual și este folosită ca parametru pentru funcția drop_and_format(df, header row, columns_to_drop), care realizează ștergerea coloanelor din final_dataset.csv și, de asemenea, exclude coloana ce include indexul original al rândurilor, adăugat automat de Pandas.
Pentru datele de testare s-au selectat aleator 40000 de rânduri din fișierul final_dataset.csv dată de instrucțiunea:
sampled=data_f.sample(n=min(40000,len(data_f)),random_state=42).reset_index(drop=True)
La fiecare rulare, datorită setării random_state=42, selecția datelor pentru test va fi fix aceeași — aceleași rânduri, în succesiune identică. Asta menține testul consistent și comparabil între diferite execuții, lipsit de schimbări cauzate de alegerea întamplatoare a rândurilor.

5.2 Implementarea modulului Log Generation

Procesul pornește din contextul în care trebuie creat la intervale regulate un fișier log care să conțină un eșantion relevant, curat și usor de lucrat în următoarele etape.
Încărcarea unică a dataset-ului:
Pentru a nu citi fișierul constant când vrem un nou log, citim o singură dată la pornire. Astfel:
Încărcăm întregul fișier CSV o singură dată și îl păstrăm în memorie.
Toate celelalte etape preiau datele din acest container deja încărcate.
Acest principiu reduce dramatic numărul de procese I/Output și optimizează perioada execuției fiecărei runde pentru logging.
Pentru a nu produce mereu un subset constant și a acoperi diversitatea datelor:
Alegem de fiecare dată un numar aleatoriu de rânduri, cuprins între un prag minim de 10 și prag maxim cu valoarea 20, pe care să le includem în log.
Eșantionarea se face fără înlocuire, astfel încât nu apar duplicate în cadrul aceluiași log.
Prin această tehnică se permite analizarea traficului în ferestre distincte, oferind con-

textul favorabil de a observa comportamentul și schimbările acestuia pe baza unor eșantioane diverse.

Acum la fel ca în implementarea modulului Data Preparation folosim aceeași listă care este construită manual ca să fie implementată pentru ștergerea coloanelor din final_dataset.csv.

Această fază pentru filtrare este esențială pentru reducerea dimensionalității și pentru accelerarea etapelor care urmează.

Pe masură ce extragem pot apărea intrări ce conțin valoarea infinit sau cărora le lipsește un camp. Astfel de rânduri ce conțin valori infinite ori lipsesc câmpuri sunt eliminate. Aceasta tehnică este comună în ștergerea datelor nevalide care ar putea cauza erori pentru analize următoare.

Generarea fișierului și formatul final:

Fiecare eșantion curat e scris direct în fișierul recent creat al cărui nume include un timestamp exact (an, lună, zi, oră, minut, secundă). Orice fișier proaspăt creat este trimis în directorul logs.

Folosim o convenție clară pentru denumire ca să ușurăm ordonarea și determinarea succesiunii temporale.

După fiecare salvare, algoritmul masoară intervalul de timp execuției sarcinilor de eșantionare, filtrare și scriere. Dacă toate acestea au fost executate într-un interval mai scurt decât timpul stabilit (10 secunde), așteptăm perioada rămasă. Dacă a durat un timp extra, nu întârziem și pornim imediat un nou ciclu, raportând doar simplu mesaj de atenționare. În acest fel, generatorul funcționează constant la o frecvență fixă, adaptându-se la schimbarile de încărcare.

Pe parcursul buclei repetitive(while True) care creează log-uri la fiecare 10 secunde, funcția pentru creare a log-urilor (create_log_file) asigură că dataset-ul este configurat corect și poate emite erori atunci când este cazul. Suplimentar, funcția main conține un handler pentru KeyboardInterrupt care asigură capturarea evenimentului opririi (exemplu. Ctrl+C) și afișează mesaj de confirmare anterior ieșirii din program. Astfel, pipeline-ul rămâne fault-tolerant și controlabil cu ușurintă.

5.3 Implementarea modulului DDoS Frequency & Decision Tree Analysis

5.3.1 Introducere și context

Pentru a clasifica pachetele de trafic de rețea în două categorii -DDoS și benign- m-am inspirat din algoritmul ID3 care crează un arbore de decizie pe baza câștigului de informație. Strategia de bază este: pentru fiecare pas, alegem atributul (coloana) care oferă separarea adecvată (information gain) între cele două clase, apoi împărțim datele conform valorilor acelei caracteristici. Procesul se repetă recursiv pe fiecare subset până la atingerea unor condiții de oprire.

În implementare, am creat o versiune binară pentru arborele de decizie ca să abordez atât atribute categorice, cât și continue într-un mod uniform, pe baza unui singur prag de split.

5.3.2 Calculul entropiei

Entropia este masura incertitudinii într-o distribuție de clase. Notăm prin n număr de instanțe din clasa ”Benign” și cu m numărul de exemple din clasa ”DDoS” din cadrul unui anumit subset.

Definiția entropiei pentru două clase este:

def calculate_entropy(n, m):

$$H[n+, m-] = \frac{n}{n+m} * \log_2\left(\frac{n+m}{n}\right) + \frac{m}{n+m} * \log_2\left(\frac{n+m}{m}\right)$$

$$H[34+, 38-] = \frac{34}{72} * \log_2\left(2.11764705882\right) + \frac{38}{72} * \log_2\left(1.89473684211\right) = 0.99779166666$$

Figura 5.1: Formulă calcul entropie

Cu cât valorile celor două fracțiuni se apropie de 0.5, cu atât incertitudinea crește. Dacă un subset conține doar exemple dintr-o singură clasă, entropia este zero.
Implementare și optimizări:
Funcția calculate_entropy(a, b) calculează entropia și tratează cazurile de limită pentru evitarea logaritmilor de zero:
Dacă, întoarce 0.
Dacă sau se aplică formula simplificată pentru singura clasă prezentă.
Aceasta asigură stabilitate numerică și conformitate cu definiția teoretică.

5.3.3 Calculul entropiei totale

Pentru fiecare atribut folosit în construcția arborelui de decizie, asociem un nod parinte și un set de noduri copil corespunzătoare valorilor unice ale atributului. Funcția total_entropy(saved_H0) realizează următorii pași pentru calculul entropiei totale a acelui nod părinte.

Structura datelor de intrare

saved_H0 este o listă de string-uri:

```
saved_H0 = [
  "[61+,38-]Protocol",  # nod părinte: 61 benign, 38 instanțe ddos
  "[34+,38-]",          # copil 1: 34 benign, 38 ddos
  "[25+,0-]",           # copil 2: 25 benign, 0 ddos
  "[2+,0-]"             # copil 3: 2 benign, 0 ddos
]
```

Formatul general al fiecărui string este ”[n+,m-]”, unde:
n = numărul de instanțe din clasa „benign”
m = numărul de instanțe din clasa „ddos”

Calculul totalului de instanțe din nodul părinte

Nodul părinte este întotdeauna reprezentat de string-ul de pe poziția 0 din saved_H0. Acest string conține, pe lângă perechea[n+,m-], și numele coloanei pentru care calculăm entropia totală. Valorile n și m extrase din acel string corespund numărului total de instanțe benign, respectiv ddos, pentru un anumit dataset în contextul dataset-ului curent pentru coloana respectivă.
Notă:"setul de date curent" se actualizează la fiecare nivel al arborelui: pe masură ce coborâm în adâncime, filtrăm datele după regulile impuse de nodurile anterioare și inseram atributele exploatate în lista pentru ignorate, în așa fel încât n și m corespund întotdeauna cu numărul de instanțe rămase la acel nivel.

Calculul entropiei pentru fiecare copil

Pentru fiecare copil "[n+,m-]" din lista saved_H0[1:] se calculează entropia locală cu funcția:
H(n,m) = calculate_entropy(n, m).
Fiecare valoare de entropie locală este apoi adăugată în lista entropies: pe prima poziție din listă se află entropia primului copil, pe a doua poziție entropia celui de-al doilea copil și așa mai departe.
În același pas, se calculează numărul de instanțe al copilului, n + m, și se adaugă la finalul listei counts (astfel counts[0] corespunde primului copil, counts[1] celui de-al doilea etc.).

Entropia totală: medie ponderată

După ce am calculat entropia locală pentru fiecare copil și am aflat numărul de instanțe per copil, calculăm entropia totală ca medie ponderată a entropiilor locale.

Formula pentru entropia totală este:

$$\text{total\_sum\_H} = \sum_{i=0}^{\text{len(entropies)}} \left(\frac{\text{counts[i]}}{\text{numar\_total\_instante}} * \text{entropies[i]} \right)$$

Figura 5.2: Medie Ponderată

În cod aceasta corespunde fragmentului:

```
# Calculeaza suma ponderata
total_sum_H = 0
for i in range(len(entropies)):
    total_sum_H += (counts[i] / total) * entropies[i]

return total_sum_H
```

Unde:
len(entropies) = numărul de copii (noduri-copil)
counts[i] = numărul total de instanțe din copilul i (n + m pentru copilul respectiv)
numar_total_instante = numărul total de instanțe din nodul părinte (61 + 38 = 99 în exemplul nostru)
entropies[i] = entropia locală calculată pentru copilul i
Ponderea fiecărui copil: Raportul counts[i] / numar_total_instante reprezintă ponderea (greutatea) copilului i în calculul entropiei totale. Aceasta reflectă proporția instanțelor care ajung în acel copil față de totalul instanțelor din nodul părinte.
Contribuția fiecarui copil: oricare copil influențează entropia totală proporțional cu numărul de instanțe pe care le conține. Un copil cu un set mai extins de instanțe are o influență superioară asupra entropiei totale.
Calculul pas cu pas pentru exemplul nostru:
total_sum_H = (72/99) * 0.99 + (25/99) * 0 + (2/99) * 0
total_sum_H = 0.7273 * 0.99 + 0.2525 * 0 + 0.0202 * 0
total_sum_H = 0.72

Semnificația rezultatului:
Entropia totală calculată reflectă masura haosului rămas după împărțirea instanțelor pe baza atributului analizat. O entropie totală redusă reflectă o segmentare mai bună a datelor, cu noduri copii mai omogene în raport cu clasele.
Având în vedere contextul în care cream arborele de decizie, această entropie totală va fi folosită pentru a calcula castigul de informație (information gain) al atributului respectiv, care va decide dacă acest atribut este potrivit pentru divizarea nodului actual.

5.3.4 Structurarea datelor Entropiei

Înaintea calculului de câștig de informație, trebuie să determinăm distribuția claselor pentru fiecare potențială divizare pe atributul dat. Obiectivul este să putem calcula rapid entropia fiecărui subset și să o cântărim după mărimea acestuia.
Ca să evaluez proprietățile unei coloane, este relevant să știu cum se împarte distribuția claselor pe fiecare valoare a acelui atribut. Din acest motiv am creat funcția def build_H(frequency_table: pd.DataFrame, col: str)

Analiza Parametrilor de Intrare

Parametrul frequency_table reprezintă un DataFrame Pandas care conține setul de date structurat. Din prisma învățării automate, acest parametru descrie spațiul cu caracteristici multiple în care fiecare rând corespunde unei observații (sample) și orice coloană unui atribut (feature). Structura tabelară permite aplicarea operațiilor vectorizate pentru calculele statistice.
Parametrul col (str) specifică numele coloanei (atributul) pentru care se construiește structura H.

Determinarea totalurilor globale (nodul părinte)

Pentru a încerca să separăm datele dupa valori ale unui atribut, este obligatoriu să cunoaștem configurația inițială pentru întregul subset de date aflat la nivelul curent al arborelui de decizie. Această configurație, denumită ”nod parinte” este determinată prin cei doi indicatori principali: numărul total de instanțe etichetate ”benign” și totalul de cazuri ”ddos”.
Pentru coloana data ca parametru salvăm în total_benign numărul total de rânduri ce au label benign în frequency_table la fel pentru total_ddos.

Identificarea și ordonarea valorilor unice ale atributului

Fiecare valoare distinctă dintr-un atribut devine un subgrup (nod copil) în arborele de decizie.
Păstrarea secvenței autentice din DataFrame asigură consecvența între procesarea vectorizata și afișarea ulterioară a rezultatelor. Extragem lista cu ajutorul metodei Pandas unique():

values = frequency_table[col].unique().tolist()

Calculul frecvențelor pe clasă pentru fiecare valoare (noduri copil)

Pentru fiecare valoare(nod copil), numărăm separat instanțele ”benign”de cele ”ddos” și stocăm perechea(benign_count, ddos_count)

Construirea structurii finale H

Scopul este de a obține o listă structurată care include atât informația nodului părinte, cât și pe cea pentru fiecare nod copil.

Construcția se face în două etape:

Elementul 0: un șir de forma ”[total_benign+,total_ddos-]col”, de exemplu ”[120+,80-]Protocol”.

Elementele 1…k: câte un șir ”[benign_i+,ddos_i-]” pentru fiecare valoare unică, folosind tuplurile din value_counts.

Implementarea Python este:

```
H = [[f"[{total_benign}+,{total_ddos}-]{col}"] +  \
[f"[{value_counts[value][0]}+{value_counts[value][1]}-]" \
for value in values]]
```

Rezultatul este o listă astfel:

```
H = [
  ["[total_benign+,total_ddos-]Coloana"],      # nodul părinte
  "[benign1+,ddos1-]",                         # copil 1
  "[benign2+,ddos2-]",                         # copil 2
  ...                                          # restul copiilor
]
```

Exemplu complet pentru cifrele anterioare:

```
H = [
    [
        "[61+,38-]Protocol",  # nod părinte
        "[34+,38-]",          # copil 1
        "[25+,0-]",           # copil 2
        "[2+,0-]"             # copil 3
    ]
]
```

Output-ul funcției build_H

După parcurgerea tuturor pașilor descriși anterior, funcția build_H returnează trei obiecte separate, fiecare având un rol bine specificat în procedura de creare și analiză a arborelui de decizie:
H – Structura pregătită pentru calculul entropiei totale
value_counts – Acest dicționar are ca chei valorile unice ale coloanei analizate, iar elementele sunt tuple (benign_count, ddos_count) care indică distribuția claselor pentru fiecare dintre aceste valori.
values – Lista valorilor unice din coloana col specificată ca parametru.

Rezultatul H este apoi folosit ca argumentul saved_H0 în funcția pentru calculul entropiei totale (5.3.3), unde:
saved_H0[0] oferă Perechea (n,m) a nodului părinte și denumirea atributului.
saved_H0[1:] aduce informațiile cerute fiecărui copil pentru entropia locală și pondere.

5.3.5 Procesarea tuturor atributelor

Funcția process_columns(f_table: pd.DataFrame, exclude_cols=None) servește drept dirijor pentru calculul entropiilor fiecărui atribut din setul de date Pandas. Pornind având ca punct de plecare o tabelă DataFrame etichetată cu clasele ”benign” și ”ddos”, această procedură apelează în bucla build_H (5.3.4) și după aceea realizează pașii pentru calculul entropiei locale și totale (5.3.2–5.3.3) la fiecare atribut în parte. Rezultatul final este un dicționar cu valorile entropiilor și un altul cu structurile de frecvență (H) pentru fiecare atribut.

Parcurgem coloanele DataFrame-ului dat ca parametru:

for col in f_table.columns:

Înlocuim etichetele cu entropiile calculate pentru fiecare valoare:

```
H[0][1:] = [calculate_entropy(counts_dict[val][0], counts_dict[val][1]) for val in unique_vals]
└──────# ['[61+,38-]Protocol', np.float64(0.997772472089982), np.float64(0.0), np.float64(0.0)]
```

Figura 5.3: Etichetele devin valorile entropiei

Entropia totală pentru coloana col:

```
# Entropia totala pentru coloana {col}:
tot_ent = total_entropy(saved_H)
└──────Entropia totala pentru coloana Protocol: 0.7256527069745323
```

Figura 5.4: valoarea entropiei totale pentru coloana curentă

Stocăm entropia totală pentru coloana col:

```
# Stocam entropia totala intr-un dictionar unde cheia este coloana (atributul) col, iar valoarea este entropia calculata pentru acea coloana
entropy_results[col] = tot_ent
```

Figura 5.5: Stocarea entropiei totale

După ce am parcurs toate coloanele:

```
# Returnam dictionarul care contine valorile entropiilor calculate pentru fiecare coloana (atribut), precum si tabelele de contingenta ale atributelor, stocate in H_structures
return entropy_results, H_structures
# entropy_results = {
    Protocol: 0.6248037930698694,
    Tot Fwd Pkts: 0.6988470583725098,
    TotLen Fwd Pkts: 0.2754887502163468,
    Flow Byts/s: 0.2754887502163468,
    Fwd Pkts/s: 0.0
}
```

Figura 5.6: dicționarul cu valorile entropiei calculate pentru coloane

5.3.6 Selecția atributului cu entropia minimă

Funcția process_entropy_results are ca obiectiv determinarea și selecția atributului care oferă cel mai bun punct de split la nivelul actual al arborelui de decizie. Aceasta sortează entropiile calculate pentru fiecare atribut, compară rezultatele ca să evidențieze ordinea de impact și alege, în cele din urma, atributul cu entropia minimă(se utilizează selecția aleatorie între cele egale). Prin urmare, se încheie ciclul de determinare a atributului cu cel mai mare impact, pregătind divizarea (split-ul) nodului actual în arborele de decizie DDoS vs benign.

Intrări și semnificația lor

entropy_results este un dicționar în care cheia este numele fiecărei coloane (str), iar elementul asociat este entropia totală obținută în process_columns(5.3.5).

Returnarea rezultatului

Funcția returnează numele coloanei(selecția se face random dacă există mai multe coloane).

return node

5.3.7 Determinarea pragului optim pentru atribute continue

Funcția find_binary_split: are rolul de a stabili, pentru un atribut numeric, valoarea care definește pragul (”split”) care minimizează entropia totală a celor două sub-mulțimi rezultate.

Intrări & ieșiri

Intrări:
df (pd.DataFrame) este subsetul actual de date filtrat până în cadrul nodului existent, cu etichetele în coloana Class.
col (str): numele coloanei (atributului) pentru care căutăm un split binar.

Ieșire:
best_split (float sau None) este valoarea prag pentru care entropia rezultată este minimă.

Pașii detaliați ai algoritmului

Mai întâi se extrag valorile distincte:

vals = df[col].unique()

După se stabilește pragul optim pentru atribute continue: Se obține lista ordonată a valorilor numerice distincte. best_ent ține cea mai mică entropie ponderată întalnită, inițial infinit, iar best_split stochează pragul ideal corespunzător.

Iterația peste fiecare interval adiacent din lista ordonată a valorilor numerice distincte.

```
for i in range(len(nums) - 1):
    split = (nums[i] + nums[i+1]) / 2
```

Pentru fiecare pereche de valori consecutive (nums[i], nums[i+1]), calculăm pragul mijlociu: split = (nums[i] + nums[i+1]) / 2

Acum împărțim df în două subseturi:

left : instanțele cu valoarea atributului <= split

right : instanțele cu valoarea atributului > split

Dacă unul dintre ele este gol, pragul nu este valid și se trece la următorul.

Calculul entropiei ponderate:

Numărăm instanțele din fiecare clasă în cele două subseturi:

```
l_b = (left['Class'] == 'benign').sum()
l_d = (left['Class'] == 'ddos').sum()
r_b = (right['Class'] == 'benign').sum()
r_d = (right['Class'] == 'ddos').sum()
total = l_b + l_d + r_b + r_d
```

Aplicăm formula media ponderată a entropiilor locale (5.3.3):

$$w\_ent = \frac{l\_b + l\_d}{total} H(l\_b, l\_d) + \frac{r\_b + r\_d}{total} H(r\_b, r\_d)$$

Figura 5.7: Calculul entropiei ponderate

unde H(n,m) = calculate_entropy(n, m) (5.3.2)

Actualizam cel mai bun split:

```
if w_ent < best_ent:
    best_ent   = w_ent
    best_split = split
```

Returnarea rezultatului:

return best_split if best_split is not None else nums[0]

Dacă nu a găsit split valid (de exemplu, toate diviziunile ar lăsa un subset gol), returnăm pur și simplu nums[0], primul element numeric.

Altminteri, returnăm best_split, pragul optim selectat.

Această procedură, find_binary_split extinde metoda pentru selecție a split-urilor.

5.3.8 Generarea recursivă a arborelui de decizie binar

Funcția build_decision_tree implementează algoritmul pentru construirea unui arbore de decizie folosind criteriul de selecție bazat pe entropie. Aceasta funcționează prin divizarea recursivă a datelor pe baza caracteristicilor care oferă reducere maximă a entropiei.

Intrări și ieșiri

df(pd.DataFrame)este subsetul actual de eșantioane selectate anterior de-a lungul arborelui cu etichetele în coloana Class. max_depth este adâncimea maximă permisă a arborelui; recursivitatea se finalizează când depth >= max_depth. min_samples simbolizează numărul minim de mostre cerute pentru a permite un nou split; dacă len(df) <= min_samples, nodul devine frunză. depth denotă nivelul actual (0 la rădăcină), incrementat la fiecare recursiune. ignored (List[str]): lista atributelor folosite anterior în split-urile precedente(si 'Class'), care nu trebuie luate în considerare încă o dată.

Ieșire (Dict):

un dicționar ce descrie fie un nod intern:

```
{
  'attribute': <nume_col>,
  'split_value': <valoare_prag>,
  'left':  <subarbore_stâng>,
  'right': <subarbore_drept>
}
```

fie o frunză:

```
{
    'Class': 'benign' or 'ddos'
}
```

Funcția recursivă build_binary_decision_tree cuprinde: Testarea condiției de bază – condiția pentru oprire a apelului recursiv:

Atunci când nu se găsesc destule monstre (total <= min_samples), în situația în care o clasă lipsește (b_count==0 sau d_count==0), sau s-a atins adâncimea maximă (depth>=max_depth), nodul devine terminal.

Alegerea etichetei 'benign'/'ddos' se face prin majoritate pentru a minimiza eroarea de etichetare pe acest nod.

Determinarea celui mai bun atribut de split

```
entropy_results, _ = process_columns(df, exclude_cols=ignored)

if not entropy_results:
    return {'Class': 'benign' if b_count >= d_count else 'ddos'}

best_col = process_entropy_results(entropy_results)
best_split = find_binary_split(df, best_col)

if best_split is None:
    return {'Class': 'benign' if b_count >= d_count else 'ddos'}
```

(5.3.5)process_columns calculează entropiile totale pentru toate atributele neignorate.

-5.3.6– process_entropy_results alege atributul cu entropia minimă(best col).

5.3.7 utilizează find binary split determină pragul optim(best split) al lui best col.

Dacă nu a găsit niciun atribut nou (entropy_results gol) sau lipsa unui prag valid (best_split is None), nodul devine frunză.

Divizarea datelor

```
if isinstance(best_split, (int, float)):
    left_df = df[df[best_col].astype(float) <= best_split]
    right_df = df[df[best_col].astype(float) > best_split]
```

Pentru atribute continue, folosim pragul numeric: $<=$ sau $>$.

Odată selectată cea mai bună caracteristică, datele se împart în submulțimi corespunzătoare valorilor unice ale caracteristicii respective.

Construirea nodului intern și apeluri recursive

```
return {
'attribute': best_col,
'split_value': best_split,
'left': build_binary_decision_tree(left_df, max_depth, \
min_samples, depth+1, ignored + [best_col]),
'right': build_binary_decision_tree(right_df, max_depth, \
min_samples, depth+1, ignored + [best_col])
}
```

Creăm un dicționar care descrie nodul intern, stochează atributul și pragul selectat.

Se adaugă la lista ignored cu best_col pentru a evita refolosirea lui la nivele inferioare.

Ulterior se apelează recursiv pe sub-seturile left_df și right_df, mărind depth cu 1.

Exemplu de construire a arborelui de decizie binar

Mai departe, ilustrăm un exemplu explicit care urmează pașii recursivi pentru funcția build_binary_decision_tree ca să evidențiez felul în care fiecare nivel al arborelui, se selectează cel mai informativ atribut, și creează sub-tabelele pentru fiecare valoare unică și decide dacă nodul devine frunză sau se mai poate diviza. Pentru claritate, vom utiliza dicționarul, care asociază fiecarei coloane lista valorilor unice întâlnite în dataset:

Structura arborelui rezultat

Arborele generat are urmatoarea structură:

Noduri interne: Conțin informații despre caracteristica folosită pentru divizare și valorile de test.

Noduri terminale (frunze): include decizia concluzivă(clasa prezisă).

```
def build_binary_decision_tree(df, max_depth, min_samples, depth=0, ignored=None):

Input:
    df:
    F1    F2    Class
    2     1.5   benign
    3     2.4   ddos
    5     3.1   ddos
    7     4.8   benign
    1     1.2   benign
    8     5.0   ddos

    max_depth = len(data.columns) - 1
    min_samples=1
    depth=0
     ignored=None
```

Figura 5.8: Exemplu de pornire a crearii unui arbore binar de decizie folosind un set de date redus

```
Apel initial:
build_binary_decision_tree(df, max_depth=2, min_samples=1, depth=0, ignored=['Class'])

depth=0:
   total = 6
   b_count = 3
   d_count = 3

   #nu oprim (mai mult de min_samples, doua clase, depth<max_depth)
   if total <= min_samples or b_count == 0 or d_count == 0 or depth >= max_depth:

   entropy_results, _ = process_columns(df, exclude_cols=ignored)
      #[3+,3-]F1, [1+,0-], [0+,1-], [0+,1-], [1+,0-],[1+,0-],[0+,1-]
      #Structura H cu entropie pentru fiecare valoare: ['[3+,3-]F1', np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0)]
      #Entropia totala pentru coloana F1: 0.0
      #[3+,3-]F2, [1+,0-], [0+,1-], [0+,1-], [1+,0-],[1+,0-],[0+,1-]
      #Structura H cu entropie pentru fiecare valoare: ['[3+,3-]F2', np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0)]
      #Entropia totala pentru coloana F2: 0.0
      #entropy_results = { 'F1': 0.0, 'F2': 0.0}
```

Figura 5.9: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

```
#nu oprim
if not entropy_results:


best_col = process_entropy_results(entropy_results)
#Deoarece F1 si F2 au aceeasi entropie functia process_entorpy_result va alege coloana random
#best_col = F1


best_split = find_binary_split(df, best_col)
   #nums = [1 2 3 5 7 8]

   #split = (1+2)/2 = 1.5
   #left =
       0  2      0 False
       1  3      1 False
       2  5  →   2 False → 4 True → [5, 1, 1.2, benign]
       3  7      3 False
       4  1      4 True
       5  8      5 False
   #left = [ [5, 1, 1.2, benign] ]
   #right =
       0  2      0 True     0 True → [2, 1.5, benign]
       1  3      1 True     1  True → [3, 2.4, ddos]
       2  5  →   2 True → 2  True → [5, 3.1, ddos]
       3  7      3 True     3 True  → [7, 4.8, benign]
       4  1      4 False    5 True  → [8, 5.0, ddos]
       5  8      5 True
   #right = [[2, 1.5, benign], [3, 2.4, ddos], [5, 3.1, ddos], [7, 4.8, benign], [8, 5.0, ddos]]
   #l_b  = (left['Class'] == 'benign').sum()
       #l_b  = 1
   #l_d = (left['Class'] == 'ddos').sum()
       #l_d = 0
   #r_b = (right['Class'] == 'benign').sum()
       #r_b = 2
   #r_d = (right['Class'] == 'ddos').sum()
       #r_d = 3
   #total = l_b + l_d + r_b + r_d
       #total = 1+0+2+3 = 6
   #w_ent = ((l_b + l_d)/total)*calculate_entropy(l_b, l_d) + ((r_b + r_d)/total)*calculate_entropy(r_b, r_d)
       #w_ent = (1/6)*0+(5/6)*0 = 0
```

Figura 5.10: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

```
    #0<1.5
    #if w_ent < best_ent:
        #best_ent = w_ent
        #best_split = split
    #pentru simplitate nu mai arat cum are loc comparatie si pentru spliturile pentru perechile urmatoare: (2,3); (3,5); (5,7); (7,8) - deoarece
    #deoarece pentru perechea (1,2) am obtinut deja w_ent = 0 mai mic nu o sa gasim w_ent
    #return best_split
    #best_split =1.5
if isinstance(best_split, (int, float)):
    left_df = df[df[best_col].astype(float) <= best_split]
    #left = [ [1, 1.2, benign] ]
    right_df = df[df[best_col].astype(float) > best_split]
    #right = [[2, 1.5, benign], [3, 2.4, ddos], [5, 3.1, ddos], [7, 4.8, benign], [8, 5.0, ddos]]
return {
        'attribute': best_col,
        'split_value': best_split,
        'left': build_binary_decision_tree(left_df, max_depth, min_samples, depth=1, ignored + [best_col]),
        'right': build_binary_decision_tree(right_df, max_depth, min_samples, depth=1, ignored + [best_col])
    }
```

Figura 5.11: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

```
depth=1: #left = [ [1, 1.2, benign] ]
    total = 1
    b_count = 1
    d_count = 0

    #intram in if cu d_count = 0
    if total <= min_samples or b_count == 0 or d_count == 0 or depth >= max_depth:
        return {'Class': 'benign' if b_count >= d_count else 'ddos'}
        #b_count > d_count
        #return {'Class': 'benign'}
depth=1: #right = [[2, 1.5, benign], [3, 2.4, ddos], [5, 3.1, ddos], [7, 4.8, benign], [8, 5.0, ddos]]
    total = 5
    b_count = 2
    d_count = 3

    #nu oprim (mai mult de min_samples, doua clase, depth<max_depth)
    if total <= min_samples or b_count == 0 or d_count == 0 or depth >= max_depth:

    entropy_results, _ = process_columns(df, exclude_cols=ignored)
        #entropy_results = {'F2': 0.0}
```

Figura 5.12: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

```
#nu oprim
if not entropy_results:


best_col = process_entropy_results(entropy_results)
#best_col = F2


best_split = find_binary_split(df, best_col)
   #nums = [1.5 2.4 3.1 4.8 5.0]
   #split = (1.5+2.4)/2 = (3.9)/2=1.95
   #left =
       0  1.5      0 True
       1  2.4      1 False
       2  3.1  →   2 False → 0 True → [1, 2, 1.5, benign]
       3  4.8      3 False
       4  5.0      4 False
   #right =
       0  1.5      0 False
       1  2.4      1 True → [3, 2.4, ddos]
       2  3.1  →   2 True →  [5, 3.1, ddos]
       3  4.8      3 True  → [7, 4.8, benign]
       4  5.0      4 True → [8, 5.0, ddos]
   #l_b  = (left['Class'] == 'benign').sum()
       #l_b  = 1
   #l_d = (left['Class'] == 'ddos').sum()
       #l_d = 0
   #r_b = (right['Class'] == 'benign').sum()
       #r_b = 1
   #r_d = (right['Class'] == 'ddos').sum()
       #r_d = 3
   #total = l_b + l_d + r_b + r_d
       #total = 1+0+1+3 = 5
   #w_ent = ((l_b + l_d)/total)*calculate_entropy(l_b, l_d) + ((r_b + r_d)/total)*calculate_entropy(r_b, r_d)
       #w_ent = (1/6)*0+(5/6)*calculate_entropy(1,3) =
       0+(0.833)*(0.5+(0.75*0.415))=(0.833)*(0.5+0.311)=(0.833)*(0.811)=1.644
```

Figura 5.13: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

```
#1.64<1.95
#if w_ent < best_ent:
    #best_ent = w_ent
    #best_split = split
#pentru simplitate nu mai arat cum are loc comparatie si pentru spliturile pentru perechile urmatoare: (2.4,3.1); (3.1,4.8); (4.8,5.0);
#presupunem ca best_ent este 1.64 in acest caz
#return best_split
#best_split =1.95

if isinstance(best_split, (int, float)):
    left_df = df[df[best_col].astype(float) <= best_split]
    #left = [[2, 1.5, benign]]

    right_df = df[df[best_col].astype(float) > best_split]
    #right = [[3, 2.4, ddos], [5,3.1,ddos], [7,4.8,benign], [8,5.0,ddos]]

return {
      'attribute': best_col,
      'split_value': best_split,
      'left': build_binary_decision_tree(left_df, max_depth, min_samples, depth=2, ignored + [best_col]),
      'right': build_binary_decision_tree(right_df, max_depth, min_samples, depth=2, ignored + [best_col])
   }
```

Figura 5.14: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

```
depth=2: #left = [[2, 1.5, benign]]
    total = 1
    b_count = 1
    d_count = 0

    #intram in if cu d_count = 0 si cu depth>=max_depth (2>=2)
    if total <= min_samples or b_count == 0 or d_count == 0 or depth >= max_depth:
      return {'Class': 'benign' if b_count >= d_count else 'ddos'}
      #b_count > d_count
      #return {'Class': 'benign'}

depth=2: #right = [[3, 2.4, ddos], [5,3.1,ddos], [7,4.8,benign], [8,5.0,ddos]]
    total = 4
    b_count = 1
    d_count = 3

    #intram cu depth>=max_depth (2>=2)
    if total <= min_samples or b_count == 0 or d_count == 0 or depth >= max_depth:
      return {'Class': 'benign' if b_count >= d_count else 'ddos'}
      #d_count > b_count
      #return {'Class': 'ddos'}
```

Figura 5.15: Continuarea procesului crearii arborelui binar de decizie pe baza setului de date redus

depth=2: #right = [[3, 2.4, ddos], [5,3.1,ddos], [7,4.8,benign], [8,5.0,ddos]]

#return {'Class': 'ddos'}

depth=2: #left = [[2, 1.5, benign]]

#return {'Class': 'benign'}

depth=1: #right = [[2, 1.5, benign], [3, 2.4, ddos], [5, 3.1, ddos], [7, 4.8, benign], [8, 5.0, ddos]]

```
return {
        'attribute': F2,
        'split_value': 1.95,
        'left': {'Class': 'benign'}
        'right':  {'Class': 'ddos'}
}
```

depth=1: #left = [[1, 1.2, benign]]

#return {'Class': 'benign'}

depth=0:

```
return {
        'attribute': F1,
        'split_value': 1.5,
        'left': {'Class': 'benign'},
        'right': { 'attribute': F2, 'split_value': 1.95, 'left': {'Class': 'benign'}, 'right':  {'Class': 'ddos'} }
}
```

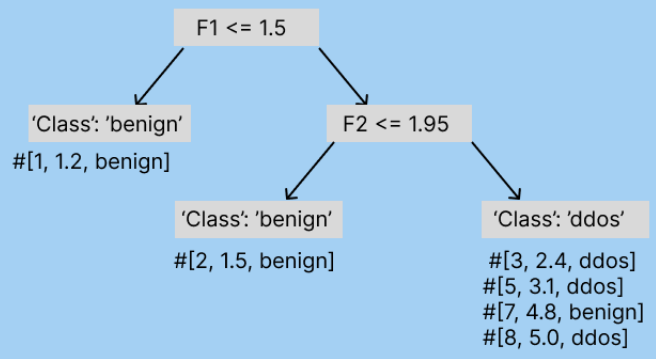

Figura 5.16: Arborele de decizie construit pe baza setului de date redus

5.3.9 Convertirea arborelui într-o structură vizualizabilă

Funcția binary_to_anytree(tree, parent=None)transformă arborele binar de decizie construit sub model de dicționar într-un format conform cu biblioteca anytree cerută pentru vizualizarea sa în consolă.

5.3.10 Evidențierea drumului parcurs în arbore pentru o instanță

Funcția highlight_tree_path_for_data_line primește un rând de date (o instanță) și lista referitoare la atribute, apoi:

Parsează valorile din linia de date.

Afișează mapping-ul dintre coloane și valorile corespunzătoare.

Execută o funcție pentru vizualizare (highlight_path_for_data_line) care accentuează drumul urmat de acel exemplu în arborele decizional.

Notă: funcția highlight_path_for_data_line este gasită în modulul de vizualizare și este importată în modulul construcției arborelui ID3.

5.3.11 Procesarea fișierelor de date noi

Funcția process_data_file(file_path, column_names) încarcă și parcurge un fișier .data. Log-urile au extensia data.

Se apelează funcția highlight_tree_path_for_data_line pentru fiecare rând conținut din log. Permite astfel testarea automată și expunerea în timp real a deciziilor arborelui privind instantele noi.

Mecanismul este folosit atât pentru resurse existente, cât și pentru loguri noi detectate în timp real.

5.3.12 Monitorizarea folderului de loguri

Funcția monitor_logs_folder configurează un sistem de monitorizare a fișierelor folosind libraria watchdog. Verifică dacă în folderul logs apar fișiere .data noi caz în care le prelucrează automat.

Crează un FileSystemEventHandler (DataFileHandler) care definește reacția la crearea sau editarea fișierelor.

Pornește un Observer care rulează în fundal și declanșează evenimentele definite.

Acest serviciu susține testarea în timp real a modelului pe noi date de rețea.

5.3.13 Optimizarea arborelui decizional

Această funcție optimize_tree_with_flag a luat naștere pentru a rezolva urmatorul caz: ori de câte ori split-ul ales separă în două grupuri care sunt deja omogene (numai ”benign” sau doar ”ddos”).

Consecințele sunt:

Algoritmul îl desemnează pentru că minimizează entropia.

Nu rămâne nicio rațiune ca să continue recursiv (condiția d_count == 0 sau b_count == 0 întrerupe crearea noilor noduri și transformă automat nodul într-o frunză).

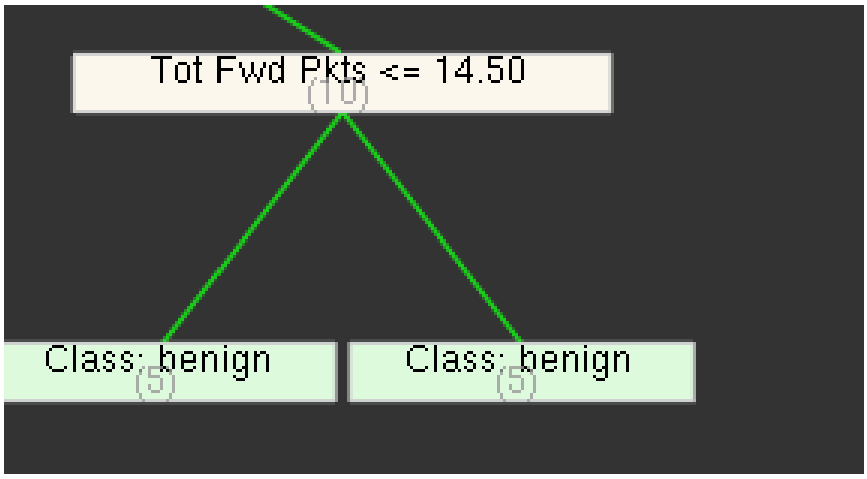

Figura 5.17: Caz special în ID3: când două frunze cu același părinte au aceeași clasă

Scopul principal

Obiectiv: Simplifică arborele de decizie prin înlocuirea întregului nod părinte (cu attribute și split_value) cu un nod simplu de forma:

’Class’: ’NumeleClasei’

Acest mecanism are loc până când nu rămâne nicio șansă pentru reducere.

Metoda: Se parcurge arborele de decizie folosind DFS post-order pentru a detecta nodurile interne cu ambii copii având aceeași clasă. Aceste noduri se înlocuiesc cu noduri frunză. Rutina are loc prin repetare iterativă până când face o parcurgere pe întreag ansamblu și nu mai găsește îmbunătățiri.

5.4 Modulul de vizualizare in OpenGL

Modulul de vizualizare folosește clasa TreeVisualizer, care pune laolaltă toate mecanismele grafice necesare pentru randarea interactivă a arborelui de decizie.

5.4.1 Inițializare init (self,screen_width=1200,screen_height=700, bg_color=(0.2, 0.2, 0.2, 1.0)

La crearea unui vizualizator, se configurează:

Dimensiuni și fundal: lățimea scenei și culoarea RGBA pentru background.

Layout noduri: raza, lațimea, înalțimea nivelelor și distanța minimă între noduri.

Navigare: variabile pentru panoramare (scroll_x/y) și zoom (factor, limite).

Culoare plus evidențiere: seturi de culori pentru noduri, muchii, frunze (”benign”/”ddos”) cât și arce.

Timere: contori având limite globale care permit nodurilor și muchiilor sublinierea cu control temporal.

Input timer: caseta textului în marginea scenei ca să modifice intervalul relevării.

5.4.2 Setarea arborelui și poziționarea nodurilor (set_tree)

Primește rădăcina ca anytree.Node, apoi:

Calculează coordonatele fiecărui nod pe orizontală și verticală.

Inițializează contorii pentru vizite și culorile de bază.

Opțional, stabilește distribuția claselor în subarbori spre a realiza o colorare suplimentară.

5.4.3 Mecanismul de Intensificare a Culorii

Modul acesta pentru vizualizare utilizează intensitatea culorii ca să reflecte numărul de repetări și valoarea evenimentului din structura de date vizualizată. Mecanismul descris se bazează pe principiul că elementele mai des vizitate sau mai importante primesc culori mai intense, creand o harta vizuala simpla ca sa utilizeze mediu.

Algoritm de Calcul al Intensității

Mecanismul utilizează o funcție de interpolare liniară pentru a decide intensitatea culorii pe baza numărului de vizitări:

progress = min(visit_count / self.max_visits, 1.0)

Formula aceasta normalizează număru de apariții la o valoare între 0 și 1, unde:
0 este gradul minim(deloc accesată)
1 echivalează cu tăria luminii maximă(cel mai des întâlnit)

Aplicarea Gradientului Cromatic

Mecanismul utilizează un gradient cromatic pentru fiecare element vizualizat între două culori extreme:

```
r = light_color[0] + (dark_color[0] - light_color[0]) * progress
g = light_color[1] + (dark_color[1] - light_color[1]) * progress
b = light_color[2] + (dark_color[2] - light_color[2]) * progress
```

Interpolația RGB permite tranziția fluidă de la culoarea deschisă(pentru elementele mai puțin vizitate) la culoarea închisă(elementele frecvent traversate), creând un contrast vizual clar între diferitele niveluri de activitate.

Sisteme de Colorare Diferențiate

Construcția prevede sisteme de colorare distincte pentru diferite tipuri de elemente:

Nodurile interne:
Paleta culorilor include grupuri coloristice după cum urmează albastru, galben și violet.
Fiecare tip conține gradiente de la nuanțe deschise la tonuri închise.
Asignarea coloristicii are loc aleatoriu pentru diversitate vizuală.

Frunze (clase):
Mecanismul utilizează un mod de codare prin culoare:
Verde este folosit pentru determinarea claselor ”benign”(activitate normală).

Rosu este desemnat ca să identifice etichetele ”attack” ori ”ddos”(comportament malitios).
Profunzimea demonstrează numărul de clasificari În grupa respectivă.

Aplicarea la Conexiuni (Arce)

Mecanismul folosește conceptul de intensitate și pentru legăturile elementelor, prin intermediul unei scheme de colorare bazată pe distribuția claselor în subarborele destinație:

```
ddos_ratio = distribution['ddos'] / total_leaves
benign_ratio = distribution['benign'] / total_leaves
```

Pe când nodurile, care utilizează un mecanism de colorare gradat bazat pe numărul de vizitări, muchiile din vizualizarea arborelui de decizie pun în folosință un mod de colorare binar și semantic, care reflectă distribuția claselor din subarborele destinație.

Muchii: Colorare Semantică Binară
Principiu: Culoarea este determinată de distribuția claselor din subarborele copil
Implementare: Atribuirea directă a culorii pe baza procentajelor de clasificare

Algoritmul calculează distribuția claselor:

```
distribution = self.subtree_class_distributions[child_id]
total_leaves = distribution['ddos'] + distribution['benign'] + \
               distribution['other']

ddos_ratio = distribution['ddos'] / total_leaves
benign_ratio = distribution['benign'] / total_leaves
```

Maparea Culorilor pe Baza Proporțiilor
Culoarea fiecarui nod este calculata prin interpolare liniara intre doua culori-baza, in functie de distributia pentru instante DDOS versus Benign:

1. Culori-baza

DDOS: (0.9, 0.1, 0.1) (roșu)

Benign: (0.1, 0.8, 0.1) (verde)

2. Interpolarea liniara

Pentru un nod cu proportii

```
ddos_ratio + benign_ratio = 1
```

Se calculeaza fiecare componenta RGB astfel:

```
r = ddos_ratio * ddos_color[0] + benign_ratio * benign_color[0]
g = ddos_ratio * ddos_color[1] + benign_ratio * benign_color[1]
b = ddos_ratio * ddos_color[2] + benign_ratio * benign_color[2]

r = max(0.0, min(1.0, r))
g = max(0.0, min(1.0, g))
b = max(0.0, min(1.0, b))
```

Fiecare canal(R,G,B) corespunde unei sume ponderate intre valorile sale pentru DDOS si BENIGN.

Prin urmare, pentru noduri pur DDOS(ddos_ratio = 1), culoarea va fi fix rosu, iar pur Benign(benign_ratio = 1) este verde.

5.4.4 Draw & Update Loop

display()

– Curată ecranul, realizează transformarile pentru zoom/pan și ca pas următor:

Desenează toate muchiile și nodurile (cu OpenGL).

Afișează tooltip-uri referitor la nodurile pe care s-a trecut cu mouse-ul.

Redă caseta de input pentru timer, indicatorul de zoom și text de ajutor.

Face buffer swap ca să se realizeze redarea scenei.

idle()

– În fiecare ciclu, calculează intervalul de timp (delta_time) și:

Update node timers:dacă un nod rămâne evidențiat mai mult decât durata setată, își resetează contorul și culoarea.

Update edge timers:analog pentru muchii.

Marchează impunerea unui redraw atunci când s-au resetat animate.

5.4.5 Evidențierea căii pentru o instanță (highlight_path_for_data)

Având în vedere o linie de date, se:

Găsește drumul nod cu nod pornind din radacină către frunză decizională.

Cu privire la fiecare nod și muchie din acest drum:

Contorul referitor ”vizitelor” creste și reseteaza timer-ul la zero (astfel punând în evidență elementul).

Adaugă nodul sau muchia la setul de elemente evidențiate, care vor fi desenate cu o culoare specială.

5.4.6 Caseta de input pentru timer (draw_timer_input_box & update_timer_values)

Desenează în colțul din dreapta-sus o cutie pentru text în care utilizatorul poate introduce un număr de secunde.

La fiecare apăsare de tastă, textul se actualizează; referitor la confirmare (de ex. Enter), update_timer_values parsează valoarea și reglează global_node_timer și global_edge_timer.

Structurile de date pentru timere

Pentru noduri:

```
self.node_timers = {}
self.node_timer_active = {}
self.global_node_timer = 2.0  #Global timer value in SECONDS
```

Pentru muchii:

```
self.edge_timers = {}
self.edge_visit_counts = {}
self.last_time = 0  # For tracking time in idle function
self.global_edge_timer = 2.0  #Global timer value in SECONDS
self.edge_timer_active = {}
```

Salvare individuală: Fiecare nod/muchie are propriul timer stocat în dicționare separate.

Recunoașterea unică: nodurile prin id(node), muchiile cu (parent_id, child_id). Revizuire continuă: Timer-ele sunt incrementate în funcția idle() cu delta_time.

Refacere automată: Când timer-ul atinge valoarea globală, visit_count-ul devine 0.
Reset la revenire: Timer-ul se resetează la 0 când elementul este vizitat din nou.
Stare activă: Flag timer_active controlează dacă timer-ul este activ sau nu.

Notă: În mod implicit, durata de stocare este setată la 2 secunde (valoarea implicită a timer-ului global). Cu toate acestea utilizatorul poate ajusta această perioadă conform cerințelor, direct din interfața pentru vizualizare.

Această implementare permite ca fiecare element vizualizat să-și păstreze starea temporar, revenind la culoarea inițială (albă) după expirarea timer-ului individual.

5.4.7 Lansarea vizualizării (visualize_binary_tree)

Creează o instanță TreeVisualizer și apelează set_tree(...) și lansează bucla OpenGL (run(window_title)), punând fereastra de vizualizare la dispoziția utilizatorului.

5.4.8 Caracteristici cheie ale modulului de vizualizare

TreeVisualizer oferă un mediu complet de explorare a arborelui de decizie:
Redare grafică cu pan și zoom.
Evidențiere dinamică pentru noduri și muchii, temporizată și reglabilă.
Tooltip-uri cât și text de ajutor.
Interacțiune directă printr-o casetă de input pentru perioada evidențierii.
Mecanismul cheie este modulul de vizualizare, ce asigură atât claritatea structurală a arborelui, cât și urmarireă deciziilor referitoare către date individuale.

Capitolul 6

Interfata si scenarii de utilizare

În acest capitol voi descrie platforma vizuală OpenGL și cum se desfasoară în timp real, cu capturi ale ecranului.

După executarea scriptului ddos_frequency_analysis.py se afisează scena OpenGL în care este redat arborele complet cu culoarea alb simbolizând că încă nu s-a porcesat nici un fișier log:

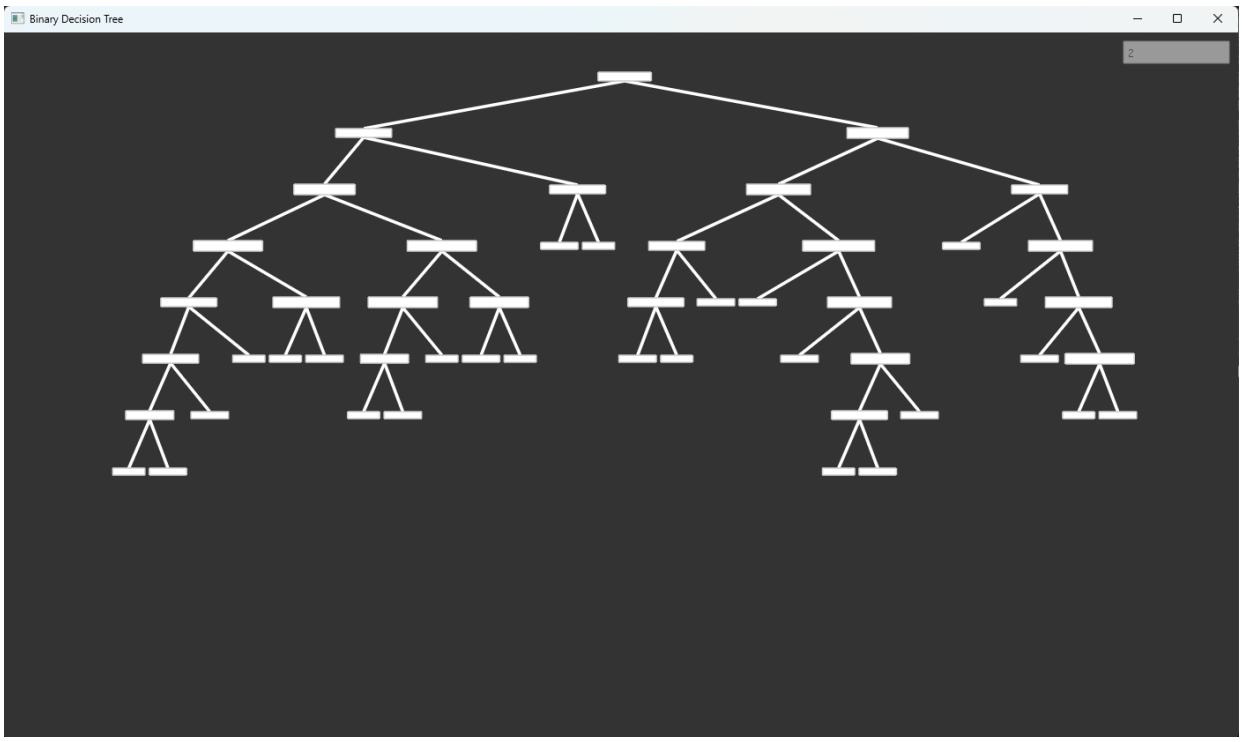

Figura 6.1: Arborele complet afișat în OpenGL cu culoarea albă - starea inițială înainte de procesarea fișierelor

De asemenea se observă faptul că în colțul dreapta sus al scenei avem căsuța de timer care by default are valoarea 2. Adică nodurile și muchiile nu vor sta colorate mai

mult de 2 secunde.

Atunci când începe să proceseze log-uri calea din arbore se ilumineză cu culoarea portocalie și nodurile și muchiile încep să capete opacitate sub aspectul coloristicii.

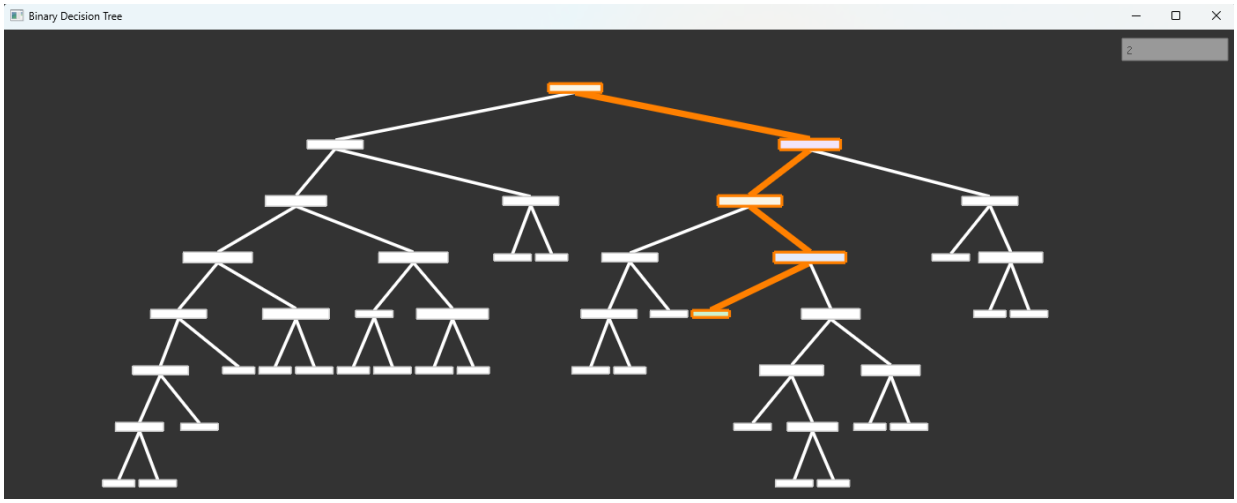

Figura 6.2: Arborele în timpul procesării - calea portocalie activă

Acum voi seta timer-ul la 20 de secunde pentru a ilustra culoarea mai intensă pe care o capătă nodurile atunci când au un grad de vizitare mai mare.

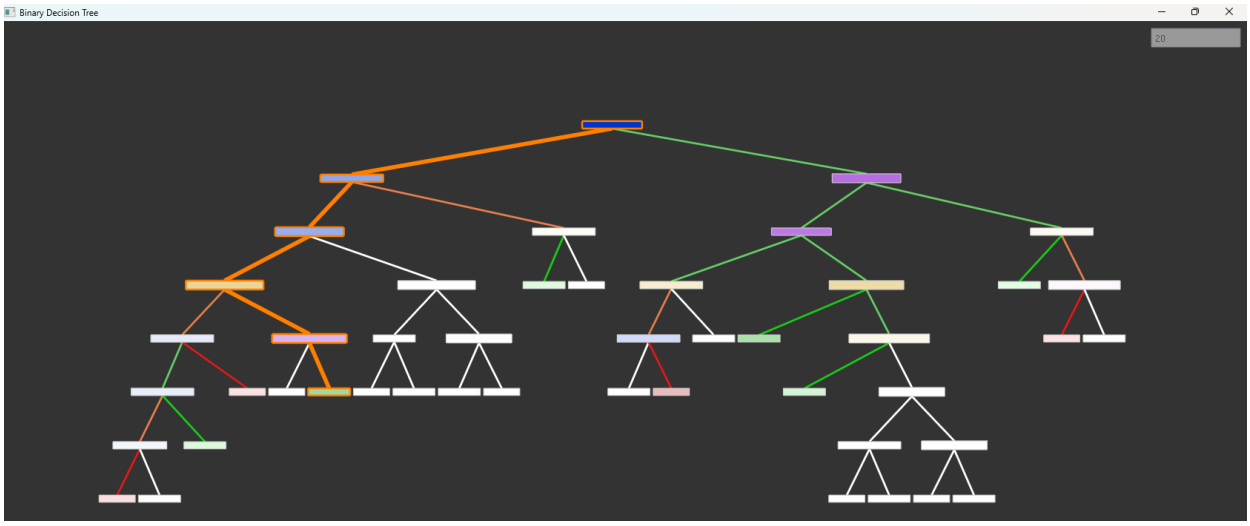

Figura 6.3: Nodurile cu vizitare frecventă capătă culori mai intense (timer: 20 secunde)

În colțul dreapta sus se observă ca timerul a fost setat la 20 și de asemena observăm ca nodul rădăcină are intensitatea celei mai inchise culori cum era așteptăt căci orice cale trece mai intâi prin rădăcină.

În aceste ilustrații a fost folosit zoom out dar este si posibil si zoom-in dacă doresc să văd atributul selectat pentru nod și pragul său.

Afișare cu zoom-in pentru subarborele stâng:

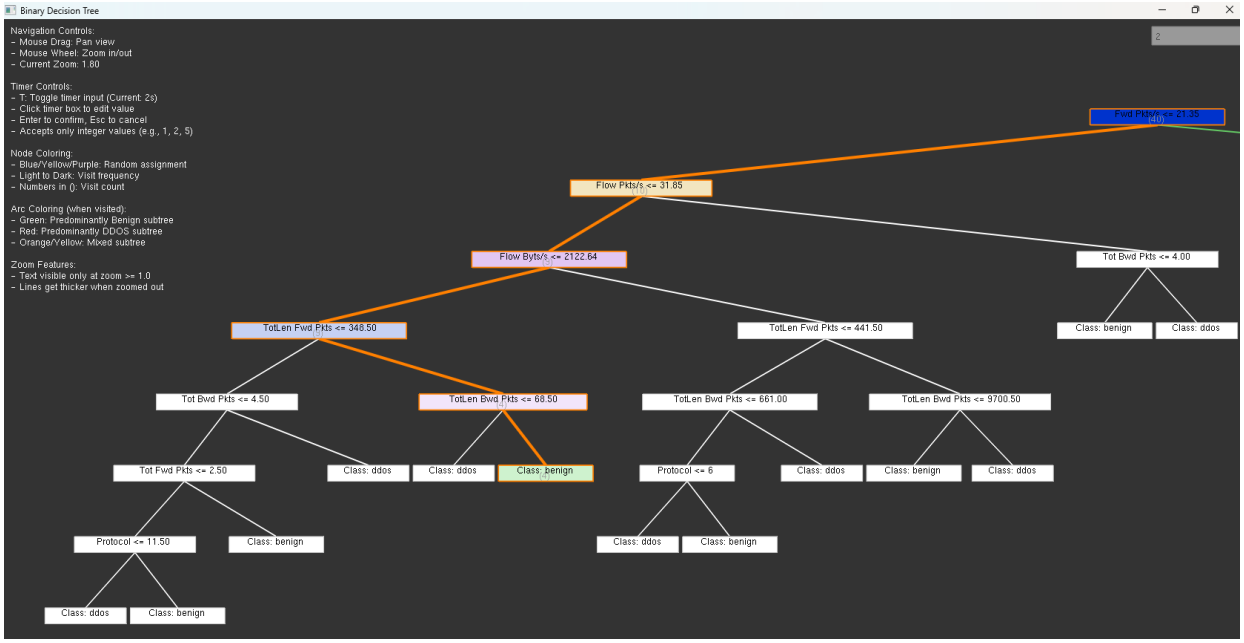

Figura 6.4: Zoom-in pe subarborele stâng

Concluzii

Lucrarea de față a prezentat un ansamblu funcțional de vizualizare și clasificare a traficului de rețea. Progresele cheie și contribuții sunt urmatoarele:

Model decizional bazat pe entropie

– Am adaptat algoritmul ID3 pentru trafic de rețea, calculând entropia locală și totală pe fiecare atribut (număr de pachete, lățime de bandă etc.) selectând split-uri care minimizează incertitudinea.

– Funcțiile build_H, process_columns și find_binary_split permit tratarea uniformă a atributelor continue și discrete, generând un arbore binar de decizie interpretabil.

Arbore recursiv și optimizare automată

Cu build_binary_decision_tree am construit arborele până la regulile pentru oprire (adâncime maximă, puritate, număr minim de instanțe).

Prin optimize_tree_with_flag am șters nodurile repetitive(cei doi copii frunză cu aceeași clasă), compressând structura și menținând acuratețea.

Modul interactiv de vizualizare a traficului

– Clasa TreeVisualizer oferă un mediu grafic OpenGL în care arborele decizional este afișat cu funcționalități pentru pan și zoom, tooltip-uri și un mecanism de evidențiere temporizată a nodurilor și muchiilor pe baza ”căilor” urmate de pachete.

– Fișierele log (format .data) sunt create prin intermediul generatorului de log-uri, iar fiecare linie de trafic este mapată în timp real pe arbore, evidențiind traseul de decizie și permițând analiza vizuală a motivelor pentru care un pachet este etichetat ”benign” sau ”ddos”.

– Timer-ul setabil (caseta de input) le oferă specialiștilor în securitate să regleze

durata pentru care nodurile rămân evidențiate, facilitând urmărirea fluxurilor pachetelor în vremuri mai lungi.

Aplicabilitate și utilitate practică

– Instrumentul permite operatorilor de rețea să inspecteze și să valideze deciziile modelului pe cazuri concrete de trafic, oferind transparență și fiabilitate în mecanismul de detecție.

– Prin monitorizarea automată a folderului de loguri, orice recent fișier apărut referitor la trafic este procesat instantaneu, iar rezultatele sunt redate în vizualizator fără intervenție manuală ulterioară.

Perspective de dezvoltare:

Extinderea la tipuri multiple de atacuri(port scanning, amplificare DNS) prin arbori multiclass.

Adăugarea în soluții de monitorizare în timp real (streaming cu Kafka/Flink) pentru detecția instantă a ddos.

Implementarea dashboard-urilor web interactive și export de rapoarte automatizate către echipele de securitate.

Concluzie generală

Per ansamblu, acest mecanism inteligent de vizualizare a atacurilor DDoS îmbină un model de decizie solid, bazat pe entropie, cu o interfață simplă și responsive, facilitând analiza, interpretarea cât și monitorizarea traficului rețelei în scopul detectării immediate a atacurilor ddos.

Bibliografie

[1] Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2005. https://www.researchgate.net/publication/251275816_Witten_IH_Frank_E_Data_Mining_Practical_Machine_Learning_Tools_and_Techniques

[2] Ian H. Witten, Eibe Frank, and Mark A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd Edition, Morgan Kaufmann, 2011. https://www.sciencedirect.com/book/9780123748560/data-mining-practical-machine-learning-tools-and-techniques

[3] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[4] Geoffrey Holmes, Andrew Donkin, and Ian H. Witten, *Weka: A Machine Learning Workbench*, Department of Computer Science, University of Waikato, Hamilton, New Zealand. https://www.researchgate.net/publication/3603890_WEKA_A_machine_learning_workbench

[5] Ian H. Witten, Eibe Frank, Mark A. Hall, Christopher J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th Edition, Morgan Kaufmann, 2017. https://www.sciencedirect.com/book/9780128042915/data-mining

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, *The WEKA Data Mining Software: An Update*, ACM SIGKDD Explorations Newsletter, vol. 11, no. 1, pp. 10–18, 2009. https://www.researchgate.net/publication/215990408_The_WEKA_data_mining_software_An_update

[7] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[8] Cezary Z. Janikow, *A Knowledge-Intensive Genetic Algorithm for Supervised Learning*, Machine Learning, vol. 13, pp. 189–228, 1993.

https://link.springer.com/article/10.1007/BF00993043

[9] Securing Machine Learning Algorithms *Apostolos Malatras, Ioannis Agrafiotis, Monika Adamczyk, ENISA*, December 2021.

https://www.enisa.europa.eu/publications/securing-machine-learning-algorithms

[10] Gopika Suresh, C. Chandrasekar, *A survey of applications of artificial intelligence and machine learning in future mobile networks-enabled systems*, Version of Record 12 June 2023

https://www.sciencedirect.com/science/article/pii/S2215098623001337

[11] Reva Schwartz, Apostol Vassilev, Kristen Greene, Lori Perine, Andrew Burt, Patrick Hall, *Towards a Standard for Identifying and Managing Bias in Artificial Intelligence*, NIST Special Publication 1270, National Institute of Standards and Technology, 2022.

https://doi.org/10.6028/NIST.SP.1270

[12] Machine Learning Quality Management Guideline, 2nd Edition, Technical Report, *National Institute of Advanced Industrial Science and Technology*, Japan, 2021.

https://www.digiarc.aist.go.jp/en/publication/aiqm/guideline-rev2.html

[13] Information and Communications in Japan, White Paper 2019, *Ministry of Internal Affairs and Communications (Japan)*

http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2019/2019-index.html

[14] Ramanpreet Kaur, Dušan Gabrijelčič, Tomaž Klobučar,
Artificial Intelligence for Cybersecurity: Literature Review and Future Directions, Volume 97, September 2023

https://www.sciencedirect.com/science/article/pii/S1566253523001136