Platforma de Laborator - Detectarea Intruziunilor în Rețele

# Curs de Cybersecurity - Ghid Practic pentru Studenți

# Cuprins

1. Introducere și Configurare (#introducere-și-configurare)
2. Modulul 1: Analiza Traficului de Rețea (#modulul-1-analiza-traficului-de-rețea)

* Laborator 1: Wireshark și Analiza Pachetelor (#laborator-1-wireshark-și-analiza-pachetelor)
* Laborator 2: Procesare PCAP și Introducere IDS (#laborator-2-procesare-pcap-și-introducere-ids)

1. Modulul 2: Machine Learning Clasic (#modulul-2-machine-learning-clasic)

* Laborator 3: Pregătirea Datelor (#laborator-3-pregătirea-datelor)
* Laborator 4: Algoritmi de Clasificare (#laborator-4-algoritmi-de-clasificare)

1. Modulul 3: Deep Learning și Evaluare (#modulul-3-deep-learning-și-evaluare)

* Laborator 5: Rețele Neurale (#laborator-5-rețele-neurale)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introducere și Configurare

## Despre acest laborator

Acest laborator vă va ghida prin procesul complet de construire a unui sistem de detectare a intruziunilor (IDS) bazat pe machine learning. Veți învăța să:

* Analizați traficul de rețea la nivel de pachete
* Extrageți caracteristici relevante din date brute
* Antrenați modele de machine learning pentru clasificare
* Evaluați și comparați performanța diferitelor abordări

## Logica laboratorului: trafic → analiză → detecție → automatizare → ML

## Descărcarea datelor

Dataset-uri necesare:

|  |  |  |
| --- | --- | --- |
| Dataset | Utilizare | Link |
| NSL-KDD | Lab 3-4 | [Kaggle](<https://www.kaggle.com/datasets/hassan06/nslkdd>) |

# Modulul 1: Analiza Traficului de Rețea

## Obiective ale modulului

* Înțelegerea structurii pachetelor de rețea
* Utilizarea instrumentelor de analiză (Wireshark)
* Automatizarea extracției de caracteristici cu Python

# Laborator 1: Wireshark și Analiza Pachetelor

## Obiective

* Capturarea și analiza traficului de rețea în timp real
* Identificarea protocoalelor și structurii pachetelor
* Recunoașterea pattern-urilor de atac

## Pregătire

1. Instalare Wireshark:

* Windows/Mac: wireshark.org/download (<https://www.wireshark.org/download.html>)
* Linux: sudo apt install wireshark

1. Descărcare fișiere PCAP de exercițiu:

* Accesați folderul data/pcap\_samples/
* Sau generati folosind solutii/generate\_pcap.py

## Exercițiul 1.1: Prima captură

Pași:

1. Deschideți Wireshark
2. Selectați interfața de rețea activă (de obicei Ethernet sau Wi-Fi)
3. View → Name Resolution → Resolve Network Addresses
4. Deschideți browser-ul și accesați câteva site-uri
5. Opriți captarea apăsând pe pătratul roșu
6. Adăugați filtrul următor pentru a vedea site-uri http(tcp.port = 80) și site-uri https(tcp.port = 443):

*dns or tcp.port == 443 or tcp.port == 80*

1. Adăugați filtrul următor pentru a vedea numele site-urilor pe care navigați:

*tls.handshake.extensions\_server\_name*

1. Statistics → Conversations. Accesați opțiunile pentru a vedea pachetele trimise prin IPv4, TCP, UDP

Întrebări de analiză:

* Câte pachete ați capturat?

*Depinde cât rulează captarea*

* Ce protocoale vedeți în coloana "Protocol"?

*TCP, http, DNS*

* Identificați pachetele DNS - ce întrebare a fost trimisă?

*Numele pentru domenii: teams.cloud.microsoft,* [*www.google.com*](http://www.google.com)

## Exercițiul 1.2: Filtrare și Analiză

Filtre utile de învățat:

|  |  |
| --- | --- |
| Filtru | Descriere |
| `tcp` | Doar pachetele TCP |
| `udp` | Doar pachetele UDP |
| `http` | Trafic HTTP |
| `dns` | Interogări DNS |
| `ip.addr == 192.168.1.1` | Pachetele de la/către IP |
| `tcp.port == 80` | Trafic pe portul 80 |
| `tcp.flags.syn == 1` | Pachete SYN |
| `tcp.flags.syn == 1 && tcp.flags.ack == 0` | SYN fără ACK (posibil scan) |

1. Deschideți în Visual Studio Code: solutii/generate\_pcap.py
2. Rulați codul, introducând în terminalul IDE-ului:

*python solutii/generate\_pcap.py --all --output-dir data/pcap\_samples*

Sarcină:

1. Încărcați fișierul normal\_traffic.pcap
2. Aplicați filtrul http
3. Găsiți o cerere GET și notați:

*http.request.method == "GET" && http.user\_agent*

Cerere GET = Browser-ul întreabă server-ul să trimită o resursă

* IP sursă și destinație
* Host accesat
* User-Agent

Pentru a nota user-agent-ul, dublu click pe un packet, **Hypertext Transfer Protocol**.

În cazul meu*: User-Agent: Mozilla/5.0\r\n*

## Exercițiul 1.3: Identificarea Atacurilor

Sarcină: Analizați fișierele PCAP și identificați tipul de atac:

Fișier 1: port\_scan.pcap

1. Încărcați fișierul
2. Observați: Statistics → Conversations → TCP
3. Întrebări:

* Câte porturi diferite au fost accesate?

*50 (fiecare rând = port diferit, 1 pachet per port)*

* Care este IP-ul atacatorului?

*IP-ul atacatorului este: 10.0.0.50*

*Ținta este: 192.168.1.10*

* Ce tip de scan este? (SYN/Connect/etc.)

***SYN scan = Start conexiune*** *(SYN=1, ACK=0, fără handshake complet)*

*În loc de: Salut! Ce faci? Salut! Bine, tu? Vorbim? Da, sigur!*

*Este : Salut! Salut! Salut! Salut! …. Nu răspunde nimeni*

Fișier 2: syn\_flood.pcap

1. Încărcați fișierul
2. Folosiți: Statistics → I/O Graphs
3. Observați rata pachetelor pe secundă
4. Întrebări:

* Care este rata de pachete SYN?

*Din grafic, 500 SYN/s*

* IP-urile sursă sunt reale sau spoofed?

**IP-urile sunt spoofed (falsificate)**  
*Nu sunt surse reale, sunt generate aleator pentru a ascunde atacatorul.*

Indicatori de SYN Flood:

* Multe pachete SYN fără răspuns ACK
* IP-uri sursă randomizate
* Toate către același port destinație

## Exercițiul 1.4: Urmărirea unui flux TCP

Fișier 3: http\_traffic.pcap

TCP Stream - vizualizarea conversației complete:

1. Click dreapta pe un pachet TCP
2. Selectați Follow → TCP Stream
3. Vedeți întreaga conversație client-server

*Host, post, content-type, content-length, username&password*

Găsiți:

* O cerere de login (căutați POST cu "login" sau "password")

*http.request.method == "POST"*

* Identificați credențialele transmise (dacă sunt în clar)

Dublu click pe un pachet, verifică username și parolă

*Admin: admin3; Parolă: 12343*

* Cât la sută din trafic este data și cât este form? Ce e data? Ce e form? Ce mai avem in afara de data?

*Data = informația trimisă*

*Form = datele transmise printr-un formular http*

*În rest, avem Ethernet headers, Ip headers, TCP headers, HTTP headers.*

## Exercițiul 1.5: Exportul statisticilor

1. Statistics → Protocol Hierarchy

* Exportați distribuția protocoalelor

1. Statistics → Endpoints

* Listați toate IP-urile și porturile

De ce avem 2 Ip-uri? De ce avem 6 TCP-uri?

*2 Ip-uri reprezintă server-ul și clientul, 6 TCP-uri pentru că fiecare combinație IP+port este considerată endpoint diferit. 5 porturi sursă diferite pentru fiecare conexiune+ 1 al server-ului*

1. Statistics → Conversations

* Exportați fluxurile de comunicație

De ce avem 5 conversatii?

*Fiecare conexiune HTTP folosește câte un port sursă și câte un handshake separat.*

*Deci avem: 5 conexiuni TCP diferite*

*5 porturi sursă diferite*

*5 handshake-uri*

*5 fluxuri separate*

## Întrebări de verificare - Lab 1

1. Ce informații conține header-ul unui pachet TCP?
2. Cum diferențiați un port scan de trafic normal?
3. De ce atacatorii folosesc IP spoofing în SYN flood?
4. Ce este un "three-way handshake" și cum arată în Wireshark?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Laborator 2: Procesare PCAP și Introducere IDS

## Partea 1: Procesare PCAP cu Python

## Obiective

* Parsarea fișierelor PCAP cu biblioteca Scapy
* Extragerea caracteristicilor relevante pentru ML
* Agregarea datelor la nivel de flux

Rulați din fișierul soluții:

python solutii/lab2\_pcap\_processing.py "data/pcap\_samples/http\_traffic.pcap"

## Exercițiul 2.1: Încărcarea și explorarea PCAP

Sarcini:

1. Rulați codul cu fișierul normal\_traffic.pcap (Pasii 1&2)
2. Căutați în output porturile sursă și destinație UDP pentru primele 3 pachete;

**Funcția** explore\_packets

*Pachet1: Sursă: 10609 Destinație: 53*

1. Numărați câte pachete sunt TCP vs UDP vs alte protocoale

*Adăugați următoarea secvență de cod în main între pasul 3 și pasul 4.*

*print("\nDistribuție protocoale (nivel PACHET):")*

*print(df\_packets['protocol'].value\_counts())*

## Exercițiul 2.2: Extragerea caracteristicilor

Caracteristici de extras pentru fiecare pachet:

|  |  |  |
| --- | --- | --- |
| Caracteristică | Descriere | Cod Scapy |
| IP sursă | Adresa IP sursă | `pkt[IP].src` |
| IP destinație | Adresa IP destinație | `pkt[IP].dst` |
| Port sursă | Portul sursă (TCP/UDP) | `pkt[TCP].sport` |
| Port destinație | Portul destinație | `pkt[TCP].dport` |
| Lungime | Dimensiunea pachetului | `len(pkt)` |
| TTL | Time To Live | `pkt[IP].ttl` |
| Flags TCP | Flagurile TCP | `pkt[TCP].flags` |
| Timestamp | Momentul capturii | `pkt.time` |

Sarcini:

1. Rulați codul și examinați DataFrame-ul rezultat
2. Adăugați o caracteristică nouă: payload\_size (dimensiunea datelor fără header)

În funcția extract\_packet\_features, adăugați înainte de *return features*:

*features['payload\_size'] = len(packet.payload)*

*|---- 20B ----|--- 8B ---|------ 30B ------|*

*IP header UDP hdr DNS data*

1. Calculați statistici: medie, min, max pentru lungimea tuturor pachetelor.

În main, între pasul 3 și 4, adăugați linia și modificați numărul de pachete(Pasul 2):

*print(df\_packets['ip\_len'].agg(['min', 'mean', 'max']))*

## Exercițiul 2.3: Agregarea la nivel de flux

Un flux = toate pachetele între aceeași pereche (IP\_src, Port\_src) ↔ (IP\_dst, Port\_dst).

Caracteristici la nivel de flux:

|  |  |
| --- | --- |
| Caracteristică | Formula |
| Durată | timestamp\_max - timestamp\_min |
| Total pachete | count |
| Total bytes | sum(length) |
| Bytes/secundă | total\_bytes / durată |
| Pachete/secundă | total\_pachete / durată |
| Lungime medie pachet | mean(length) |

1. Implementați și rulați agregarea
2. Există fluxuri cu durată 0? Ce înseamnă asta? Adăugați aceste linii după Pasul 4, după definirea df\_flows:

*zero\_dur = df\_flows[df\_flows['duration'] == 0]*

*print(f"\nFlows cu duration=0: {len(zero\_dur)} din {len(df\_flows)}")*

*Flow-uri cu durată 0 înseamnă că flow-ul conține un singur pachet. Flow-ul este „conversația”, pachetele sunt „replicile”.*

1. Ordonati dataframe-ul in ordine descrescatoare in functie de packet\_count.

Adăugați aceste linii de cod:

print("\*nTop 10 flows după packet\_count:"*)

print(*df\_flows*.sort\_values(*'packet\_count'*,ascending=False).head(10)[[*'ip\_src','src\_port','ip\_dst','dst\_port','protocol','packet\_count','total\_bytes','duration'*]])

*Acestea sunt flow-urile cu cele mai multe pachete(conversația cu cele mai multe replici).*

1. Decomentați tot main-ul și creați fișierele .csv ce conțin flow-urile și pachetele

## Exercițiul 2.4: Detectarea anomaliilor simple

Se aplică pe PCAP-uri malițioase, nu pe trafic normal, scopul este să validăm că atacurile pot fi detectate automat.

Pachetele analizate au flag-ul TCP SYN setat și ACK = 0, fără finalizarea three-way handshake-ului. Valorile SEQ sunt inițiale și nu sunt urmate de răspunsuri ACK, ceea ce indică un atac de tip **SYN Scan**, nu trafic TCP legitim.

Reguli bazate pe prag, rulati pe toate bazele de date:

port\_scan\_threshold = 10

connections\_per\_ip = df\_packets.groupby('ip\_src')['dst\_port'].nunique()

potential\_scanners = connections\_per\_ip[connections\_per\_ip > port\_scan\_threshold]

print("\nPotențiali scaneri (IP-uri cu > 10 porturi diferite accesate):")

print(potential\_scanners if len(potential\_scanners) > 0 else " Niciun scanner detectat")

flood\_threshold = 100

high\_rate\_flows = df\_flows[df\_flows['packets\_per\_second'] > flood\_threshold]

print(f"\nFluxuri cu rată mare (> {flood\_threshold} pachete/sec): {len(high\_rate\_flows)}")

if len(high\_rate\_flows) > 0:

print(high\_rate\_flows[['ip\_src', 'ip\_dst', 'protocol', 'packet\_count', 'packets\_per\_second']])

1. Rulați codul cu numele syn\_flood.pcap
2. Observați atacul

*500/500 flow-uri cu durată 0.*

1. Rulați codul pentru toate fișierele .pcap rămase, port\_scan.pcap și http\_traffic.pcap. Asigurati-va ca salvati csv (Pas 6).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Partea 2: Introducere în IDS/IPS

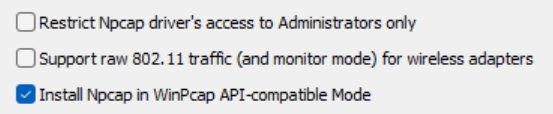
Notebook: lab2b\_ids\_intro.ipynb

În această parte veți învăța despre sistemele de detectare a intruziunilor și veți configura Snort.

## Exercițiul 2.5: Instalare Snort în Colab

Pe Windows, descărcați Npcap(driver de captură pachete, fara el, Snort nu vede trafic):

1. [*https://npcap.com/#download*](https://npcap.com/#download)După rularea executabilului, bifați:



1. [*https://www.snort.org/downloads*](https://www.snort.org/downloads)

Scroll down, până la Snort 2, click pe Snort\_2\_9\_20\_Installer.x64.exe

## Exercițiul 2.6: Înțelegerea regulilor Snortsnort

Structura unei reguli Snort:

ACTION PROTOCOL SRC\_IP SRC\_PORT -> DST\_IP DST\_PORT (OPTIONS)

Exemplu:

alert tcp any any -> any 80 (msg:"HTTP Traffic"; sid:1000001;)

Explicație: Dacă vezi trafic de tip TCP, de la oricine către portul 80, adică http, anunță-mă.

Componente:

|  |  |  |
| --- | --- | --- |
| Component | Descriere | Valori |
| ACTION | Ce face la match | alert, log, drop |
| PROTOCOL | Protocol | tcp, udp, icmp |
| SRC/DST | Adrese | any, IP, CIDR |
| OPTIONS | Detalii | msg, sid, content |

Sarcină: Studiați regulile din notebook și explicați ce face fiecare.

Rulați în terminalul VSCode:

C:\Snort\bin\snort.exe -r "port\_scan.pcap" -c "C:\Snort\etc\lab.conf" -A console -q -l "log"

Path “executabil Snort”-r “Path fișier pcap” -A console -q -l “Path log”

## Exercițiul 2.7: TODO - Scrierea regulilor custom

1. Detectare trafic HTTP. Ce detectează această regulă?

* Protocol: tcp
* Port destinație: 80

Adăugați regula de mai jos în fișierul: "C:\Snort\rules\local.rules" și salvați.

Regula: *alert tcp any any -> any 80 (msg:"HTTP Traffic"; sid:1000002; rev:1;)*

Rulați pentru http\_traffic.pcap

*Regula detectează orice pachet către portul 80.*

*Verificați în Wireshark adăugând filtrul: tcp.dstport == 80*

1. Detectare "password" în HTTP. Ce detectează această regulă?

* Folosiți: content:"password"; nocase; nu ține cont de literele mici/mari

Regula: *alert tcp any any -> any 80 (msg:"Possible Password in HTTP"; content:"password"; nocase; sid:1000004; rev:1;)*

*Regula detectează orice pachet TCP către portul 80 care conține textul “password” în payload.*

*Verificați în Wireshark parolele din pachetele cu 182 Post/ login HTTP*

1. Detectare Scan. Ce detectează această regulă?

* Protocol: tcp
* Port destinație: 23

Regula: *alert tcp any any -> any any (msg:"PORTSCAN? TCP SYN packet"; flags:S; sid:1000010; rev:1;)*

Rulați pentru port\_scan.pcap

*Regula detectează orice pachet TCP cu flag-ul SYN(pachet care încearcă să inițieze o conexiune TCP). Verificați IP sursă, IP destinație si porturile.*

1. Detectare Syn Flood. Ce detectează această regulă?

* Protocol: tcp
* Condiție: flag SYN setat
* Limitare: mai mult de 20 SYN în 1 secundă de la aceeași sursă

Regula: *alert tcp any any -> any any (msg:"SYN packet seen"; flags:S; sid:2000001; rev:1;)*

Rulați pentru syn\_flood.pcap

*Regula detectează orice pachet TCP cu flag-ul SYN setat, adică începe să inițializeze o conversație.*

## Exercițiul 2.8: Rulare Snort pe PCAP

După ce ați completat regulile:

Interpretați alertele:

* Ce reguli au fost declanșate?

**SID 1000001** → Syn

**SID 1000004** → Possible Password in HTTP

* Există false positives?

False positive = Avem alarmă, dar traficul este normal.

Regula 1000001 e foarte generică, apare de multe ori.

Regula 1000004 nu pare false positivă, pentru ca în fișierul pcap există parole pe http.

## Întrebări de verificare - Lab 2

Partea 1 - PCAP Processing:

1. Ce bibliotecă Python se folosește pentru parsarea PCAP?
2. Care e diferența dintre caracteristici la nivel de pachet și la nivel de flux?
3. De ce agregăm datele pe fluxuri pentru ML?
4. Ce caracteristici ar indica un atac de tip port scan?

Partea 2 - IDS/IPS:

1. Care este diferența principală între IDS și IPS?
2. Ce tip de IDS este Snort: NIDS sau HIDS?
3. Ce semnifică flags:S într-o regulă Snort?
4. De ce IDS signature-based nu poate detecta atacuri zero-day?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Modulul 2: Machine Learning Clasic

## Obiective ale modulului

* Pregătirea datelor pentru algoritmi ML
* Implementarea și compararea algoritmilor clasici
* Înțelegerea hiperparametrilor

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Laborator 3: Pregătirea Datelor

## Obiective

* Încărcarea și explorarea dataset-ului NSL-KDD
* Tratarea valorilor lipsă și encoding
* Normalizare și împărțirea datelor

## Pregătire

1. Descărcați NSL-KDD de pe Kaggle
2. Rulați: python solutii/lab3\_data\_preparation.py --input "KDDTrain+.txt" --outdir "solutii/lab3\_out\_train" --scale standard --test\_size 0.2

## Exercițiul 3.1: Încărcarea și explorarea datelor

Deschideți notebook-ul: solutii/lab3\_data\_preparation.ipynb

Notebook-ul conține cod complet pentru:

* Încărcarea NSL-KDD cu denumirea corectă a coloanelor
* Statistici descriptive
* Vizualizarea distribuției claselor

Sarcini de analiză:

1. Câte înregistrări sunt în setul de antrenare?

*100778, după split 80%-20%*

1. Câte features avem?

*41 features, din totalul de 43, deoarece au fost eliminate label și difficulty*

1. Care este distribuția Normal vs Attack?

*46.5% atacuri din total*

1. Există valori lipsă?

*0 valori lipsă*

## Exercițiul 3.2: Analiza exploratorie (EDA)

Sarcini:

1. Identificați cele mai frecvente 5 tipuri de atacuri

*neptune 41214*

*satan 3633*

*ipsweep 3599*

*portsweep 2931*

*smurf 2646*

1. Analizați distribuția pentru protocol\_type

*tcp 102689*

*udp 14993*

*icmp 8291*

*Adăugați în funcția basic\_eda, la final:*

print("\n[+] Distribuție protocol\_type:")

print(df['protocol\_type'].value\_counts())

1. Există features cu valori constante? (inutile pentru ML) Verificați în dataset.

*Adăugați în main, după basic\_eda(df) aceste linii de cod:*

*constant\_cols = [col for col in df.columns if df[col].nunique() <= 1]*

*print("\n*[+] Features cu valori constante:"*)*

*print(constant\_cols)*

*num\_outbound\_cmds are doar valori de 0, nu aduce informație utilă*

## Exercițiul 3.3: Encoding categorical

Encoding = procesul prin care transformăm **date categoriale (text)** în **valori numerice**, astfel încât să poată fi folosite de algoritmi de Machine Learning.

ML **nu înțelege text** (tcp, http, SF), ci doar **numere**.

Tipuri de encoding:

|  |  |  |
| --- | --- | --- |
| Metodă | Când se folosește | Exemplu |
| Label Encoding | Features ordinale sau target | tcp→0, udp→1 |
| One-Hot Encoding | Features nominale | tcp→[1,0,0], udp→[0,1,0], **matrice** |

Sarcini:

1. Aplicați encoding-ul conform notebook-ului

*Label Encoding este aplicat în funcția preprocess, identifică coloanele text protocol\_type, service, flag și le transoformă în numere.*

*Ordinea nu are semnificație, este doar o codificare.*

1. De ce doar cele 3 coloane sunt convertite în label encoding?

*Pentru ca doar cele 3 coloane sunt de tip string, restul sunt int, float, binary.*

***protocol\_type*** *→ cum comunică (TCP/UDP/ICMP)*

***service*** *→ ce aplicație/port e folosit*

***flag*** *→ cum s-a comportat conexiunea TCP*

1. Câte coloane avem după One-Hot Encoding?

Adăugați aceste linii de cod în main înainte de split:

*X\_tmp = df.drop(columns=['label', 'difficulty'])*

*X\_onehot = pd.get\_dummies(X\_tmp, columns=['protocol\_type', 'service', 'flag'])*

*print("\n[+] Nr coloane după One-Hot:",X\_onehot.shape[1])*

*print("protocol\_type:", len([c for c in X\_onehot.columns if c.startswith("protocol\_type\_")]))*

*print("service:", len([c for c in X\_onehot.columns if c.startswith("service\_")]))*

*print("flag:", len([c for c in X\_onehot.columns if c.startswith("flag\_")]))*

* protocol\_type →* ***3 coloane*** *(tcp, udp, icmp)*

* service →* ***~70 coloane*** *(http, ftp, smtp, private, etc.)*

* flag →* ***~11 coloane***

***În total, 84 de coloane.***

1. De ce nu folosim Label Encoding pentru protocol\_type?

*Pentru că tcp, udp, icmp sunt doar tipuri de protocol, label encoding sugerează o relație de ordine, dar de fapt ele sunt independent distrubuite, nu există o ordine între ele.*

Pentru normalizare/standardizare nu este nevoie de **One-Hot Encoding**, este necesar să avem date numerice, iar această operațiune este făcută de **Label Encoding**.

## Exercițiul 3.4: Normalizare/Standardizare

Diferența:

|  |  |  |  |
| --- | --- | --- | --- |
| Metodă | Formula | Range | Când se folosește |
| Normalizare |  | [0, 1] | Rețele neurale |
| Standardizare | x’ = | ~[-3, 3] | SVM, KNN |

Sarcină: Normalizați datele și verificați media/deviația standard.  
Adăugați în main, înainte de split, aceste linii de cod:

*print("\n[+] Verificare scalare:")*

*print(f"    Mean: {X\_scaled.mean():.2f}")*

*print(f"    Std:  {X\_scaled.std():.2f}")*

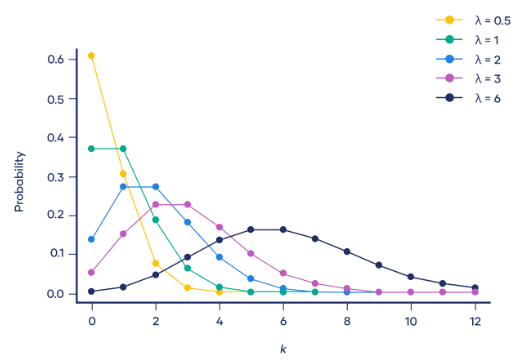
## Exercițiul 3.5: Evidențiere distribuția datelor

1. Selectați o variabilă numerică (de preferat count) și reprezentați grafic distribuția pe o scară logaritmică. Cu ce model de distribuție cunoscută seamănă? De ce am adăugat 10 ca offset în logaritmare?

Adăugați liniile de cod de mai jos între calculul pentru medie, abatere standard și split:



*Distribuția creată seamana cu o distribuție Poisson:*

**

## Exercițiul 3.6: Train-Test Split

În acest exercțiu se face split în 2 seturi de date distincte: **train(80%)**, pentru antrenare și **test(20%)**, pentru evaluarea finală, atât date, cât și etichete.

| **Date:duration** | **protocol** | **src\_bytes** | **…** | **→** | **Etichete** |
| --- | --- | --- | --- | --- | --- |
| 12.3 | tcp | 543 | … | → | 1 |
| 0.8 | udp | 45 | … | → | 0 |

## Întrebări de verificare - Lab 3

1. De ce este importantă normalizarea pentru KNN?
2. Care e diferența dintre Label Encoding și One-Hot Encoding?
3. Ce este stratify și de ce îl folosim?
4. Ce se întâmplă dacă aplicăm fit\_transform pe test set separat?

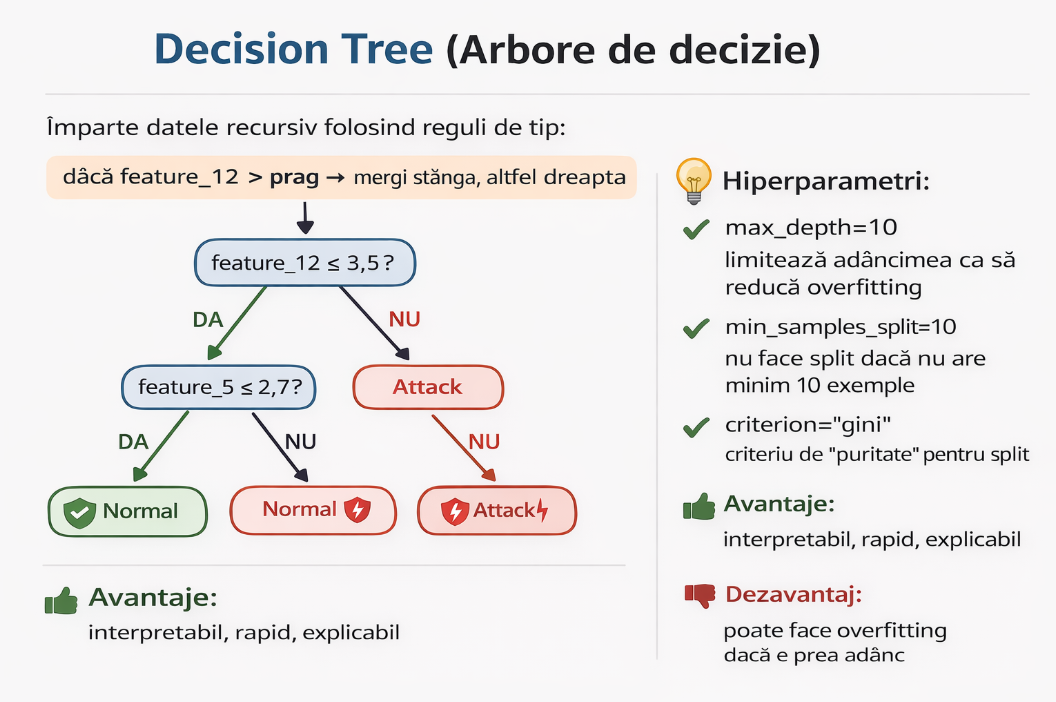
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

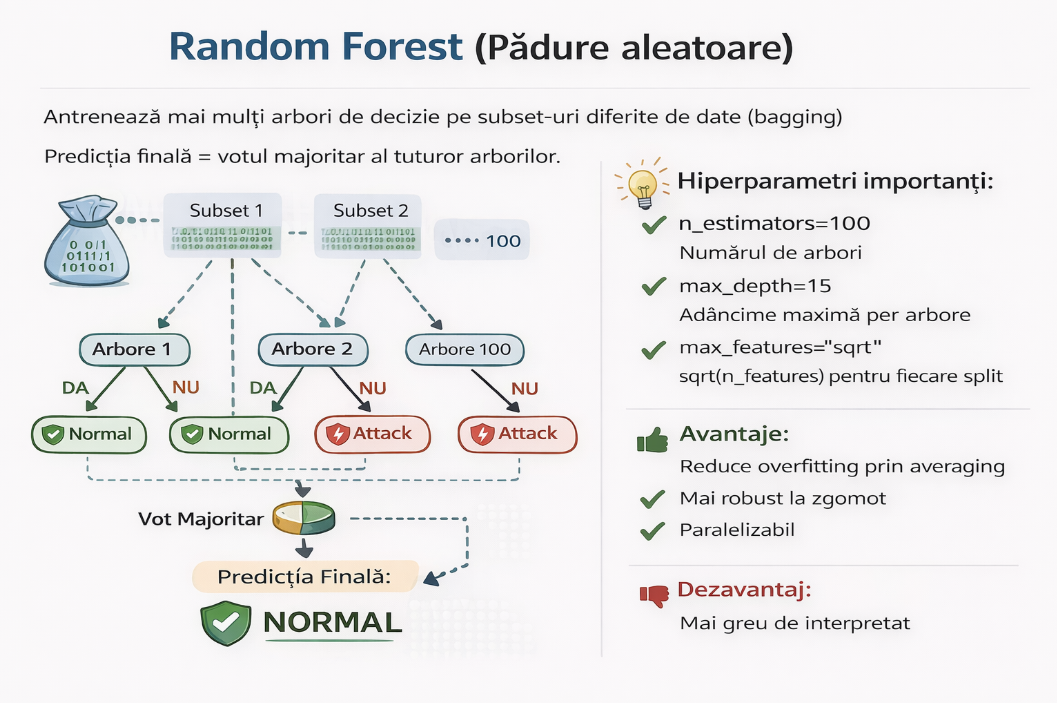
# Laborator 4: Algoritmi de Clasificare

## Obiective

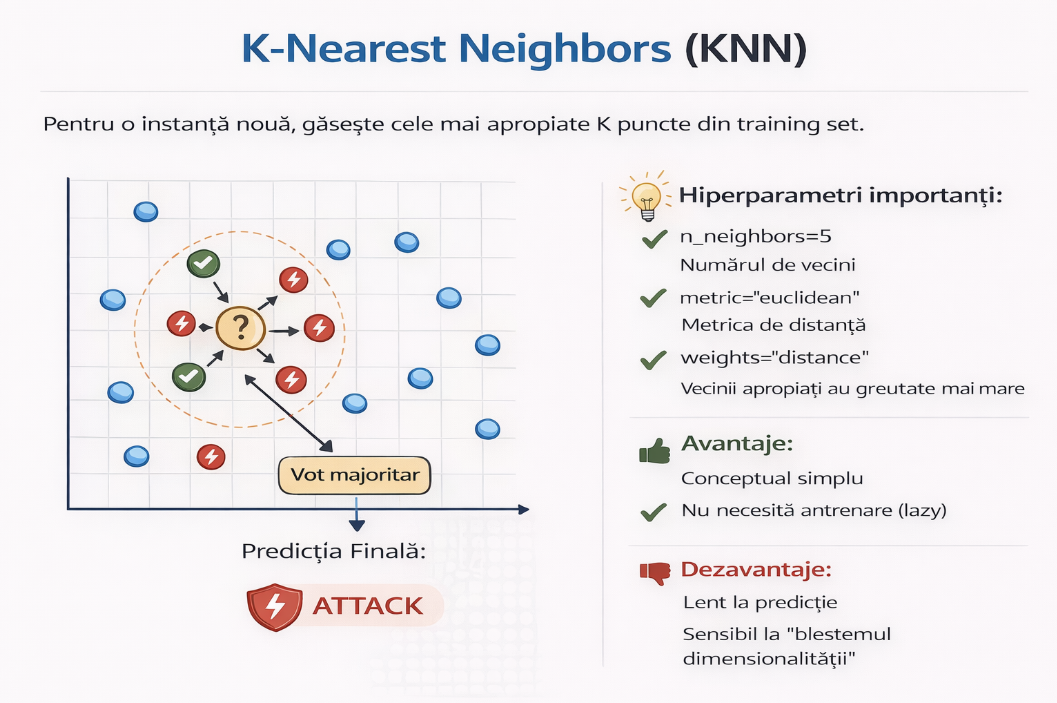
* Implementarea Decision Tree, Random Forest, KNN pentru a construi un clasificator de instruziuni(IDS).
* Înțelegerea hiperparametrilor
* Evaluarea și compararea modelelor

**Decision Tree:**

****

**Random Forest:**

**K-Nearest Neighbours**



The curse of dimensionality =   
more points lead to less capacity to prediction

Euclidian distance =

## Pregătire

1. Asigurați-vă că aveți datele salvate din Lab 3
2. Deschideti solutii/lab4\_classical\_ml.ipynb

Rulați: python solutii/lab4\_classical\_ml.py --data\_dir "solutii/lab3\_out\_train" --outdir "solutii/lab4\_out"

## Exercițiul 4.1: Decision Tree

Sarcini:

1. Rulați codul și notați accuracy-ul.

*Accuracy : 0.9976*

1. Ce se întâmplă dacă creșteți max\_depth la 20? 50?

*Arborele poate crea* ***reguli mai fine, rezultatele sunt puțin mai bune, dar creșterea adâncimii poate duce la*** *overfitting.*

1. Vizualizați arborele(poza este salvată în fișierul output-ului).

## Exercițiul 4.2: Random Forest

Pași:

1. Comparați accuracy cu Decision Tree

|  |  |
| --- | --- |
| ***Algoritm*** | ***Accuracy*** |
| *Decision Tree* | *0.9976* |
| *Random Forest* | *0.9987* |

1. Afișați feature importance (cod furnizat jos), adăugați înainte de KNN, în main:

*# -----------------------------*

*# Ex. 95: Feature importance*

*# -----------------------------*

*feature\_importance = pd.DataFrame({*

*"feature": [f"feature\_{i}" for i in range(X\_train.shape[1])],*

*"importance": rf\_model.feature\_importances\_*

*}).sort\_values(by="importance", ascending=False)*

*print("\nTop 10 Feature Importance - Random Forest")*

*print(feature\_importance.head(10))*

*plt.figure(figsize=(10, 5))*

*plt.bar(*

*feature\_importance["feature"].head(10),*

*feature\_importance["importance"].head(10)*

*)*

*plt.xticks(rotation=45)*

*plt.title("Top 10 Feature Importance - Random Forest")*

*plt.xlabel("Feature")*

*plt.ylabel("Importance")*

*plt.tight\_layout()*

*plt.savefig(os.path.join(args.outdir, "rf\_feature\_importance.png"), dpi=300)*

*plt.close()*

Ce semnifică aceste bare?

*Pe ce variabile s-a bazat cel mai mult Random Forest ca să decidă dacă o conexiune e Attack sau Normal: src\_bytes(numărul de bytes) și dst\_bytes(ratele de conexiune).*

## Exercițiul 4.3: K-Nearest Neighbors

Experimentare:

|  |  |  |
| --- | --- | --- |
| k | Accuracy | Observații |
| *1* | *0.9976* | *Risc de overfitting* |
| *3* | *0.9973* |  |
| *5* | *0.9967* | *Compromis* |
| *7* | *0.9963* |  |
| *11* | *0.9962* | *Poate fi prea smooth* |

*Valorile sunt apropiate, datorită dataset-ului, care este foarte bine separat și nu are zgomot.*

*Numărul de vecini = 5 reprezintă un compromis optim între overfitting și oversmoothing.*

## Exercițiul 4.4: Vizualizarea distanței euclidiene în KNN

1. Utilizați setul de date simplificat (20 puncte, 2 clase: *Normal* și *Atac*).
2. Alegeți un punct nou (query point) care nu aparține datasetului.
3. Calculați distanța euclidiană dintre punctul nou și toate punctele existente.
4. Identificați cei **K = 5** vecini cei mai apropiați.
5. Reprezentați grafic:
6. Punctele dataset-ului
7. Punctul nou
8. Liniile către cei 5 vecini
9. Valorile distanțelor pe fiecare segment
10. Indicați clasa prezisă pentru punctul nou

*Codul pentru acest exercițiu:*

**

## Exercițiul 4.5: Comparație și Vizualizare

1. Care algoritm este cel mai eficient? De ce?

*Random forest: acuratețe maximă, deviație standard minimă, precizie maximă (din toate alarmele de atac, câte au fost corecte?), rată de detecție maximă (din toate atacurile reale, câte au fost detectate?), scorul f1 maxim (metrică standard pentru IDS).*

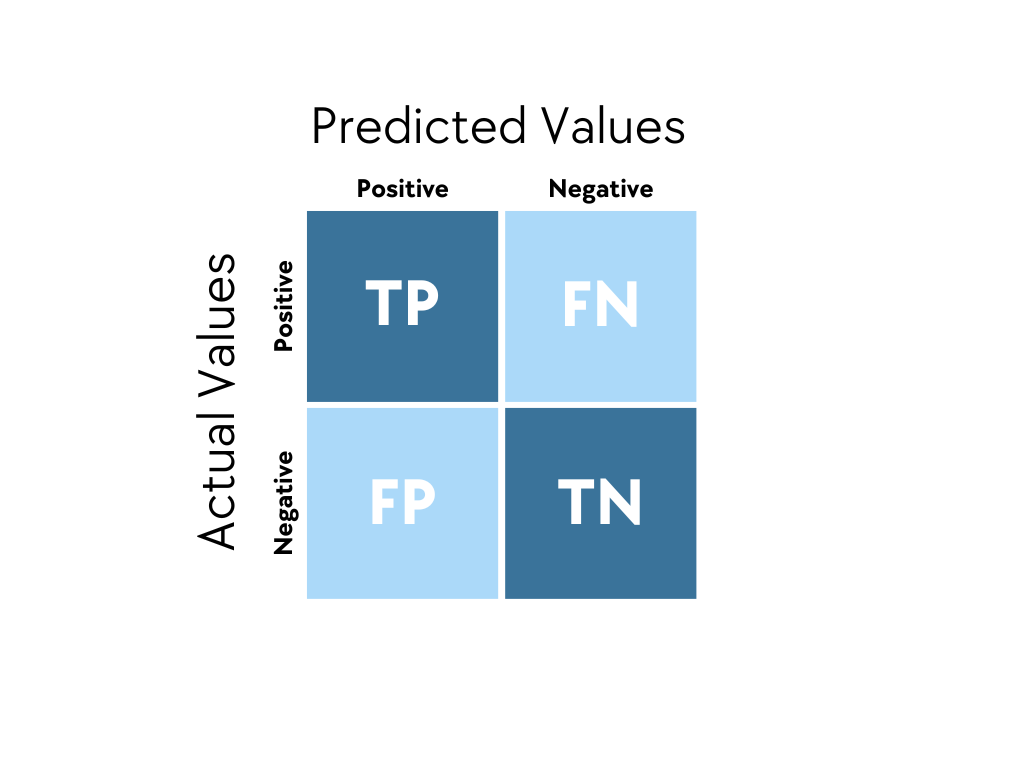
***TP*** *(True Positive): atac detectat corect*

***TN*** *(True Negative): trafic normal detectat corect*

***FP*** *(False Positive): alarmă falsă*

***FN*** *(False Negative): atac ratat*

1. Analizați matricele de confuzie furnizate de script.



*Random Forest are cele mai bune rezultate, cu cel mai mic număr de atacuri ratate și alarme false. KNN prezintă un număr mai mare de erori.*

## Întrebări de verificare - Lab 4

1. Ce este overfitting și cum îl prevenim la Decision Tree?
2. De ce Random Forest e de obicei mai bun decât un singur arbore?
3. Cum alegem valoarea optimă pentru K în KNN?
4. Ce avantaje are KNN? Ce dezavantaje?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Modulul 3: Deep Learning și Evaluare

## Obiective ale modulului

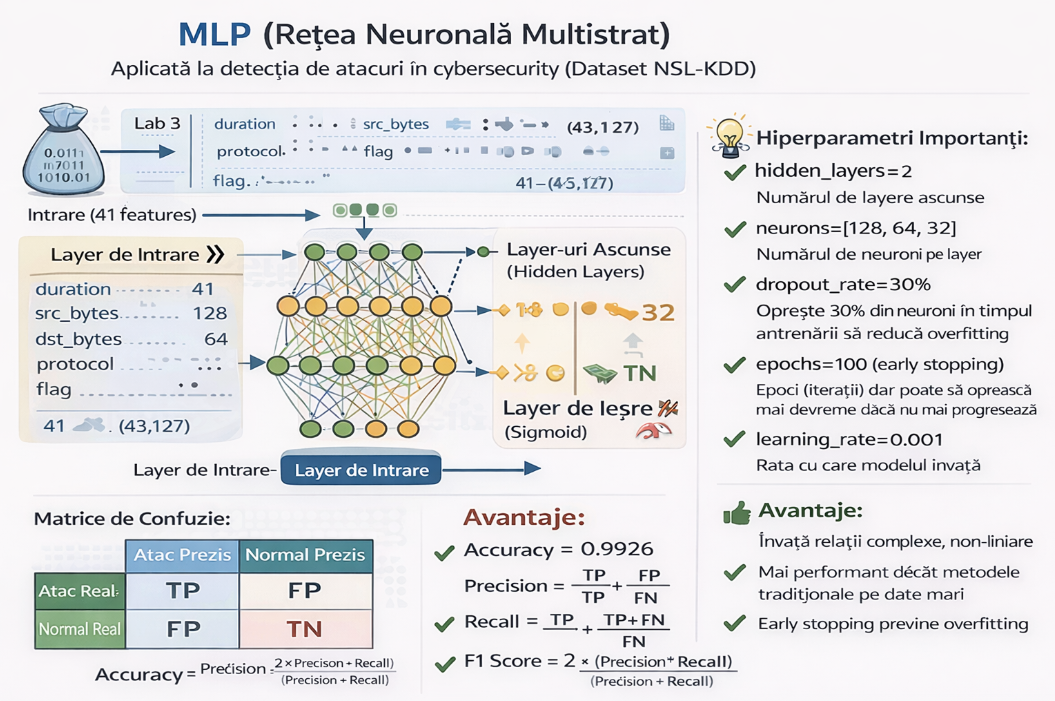
* Construirea rețelelor neurale cu TensorFlow/Keras
* Evaluarea comprehensivă a modelelor
* Interpretarea rezultatelor în context de securitate

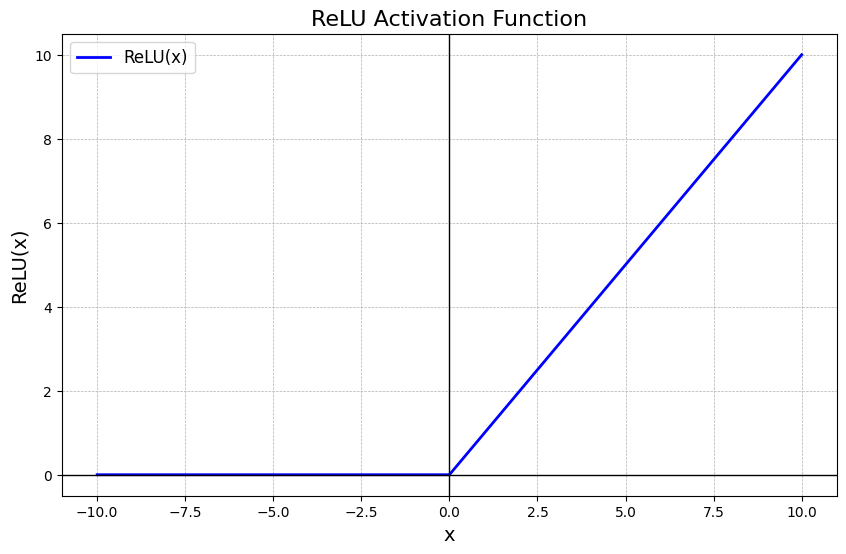
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Laborator 5: Rețele Neurale

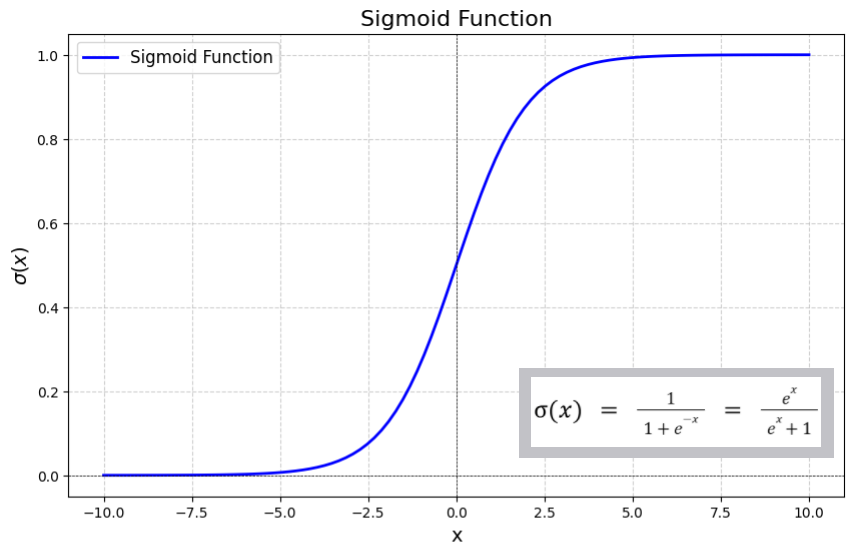
## Obiective

* Implementarea unui MLP pentru clasificare
* Înțelegerea arhitecturii și hiperparametrilor
* Vizualizarea procesului de antrenare



Funcții de activare pentru MLP:

Lasă valorile pozitive să treacă nemodificate și blochează valorile negative



Mapează input-urile în intervalul (0,1), astfel

se obțin probabilitățile.

Ambele funcții introduc non-linearitate în rețea pentru a permite acesteia să învețe relații complexe între feature-uri.

## Exercițiul 5.1: Arhitectura MLP – Antrenare și Observare

Deschideți: solutii/lab5\_deep\_learning.py

DinDin folderul proiectului, rulati: *.\tf\_env\Scripts\Activate*

Ne mutăm pe Python 3.11 deoarece tensorflow nu funcționează pe 3.13.

Pentru a rula codul: python solutii/lab5\_deep\_learning.py --data\_dir "solutii/lab3\_out\_train" --outdir "solutii/lab5\_out"

Sarcini:

1. Rulați codul și notați accuracy-ul MLP-ului, precision, recall și scorul f1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***MLP*** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
|  | *0.9924* | *0.9932* | *0.9905* | *0.9918* |

1. Ce se întâmplă dacă eliminați Dropout (Dropout(0.3) → Dropout(0.0))?

*dropout = 0*

*Accuracy: 0.9936; Precision: 0.9939; Recall: 0.9924; F1-Score: 0.9931*

*dropout = 0*

*Accuracy: 0.9919; Precision: 0.9962; Recall: 0.9863; F1-Score: 0.9912*

*Diferențele sunt minore deoarece dataset-ul este bine separat și foarte puțin zgomotos.*

1. **Ce rol are funcția ReLU în hidden layers? Dar Sigmoid?**
2. Ce este un epoch?

*Un* ***epoch*** *reprezintă o trecere completă a modelului prin toate datele de antrenare. Modelul nu învață totul într-o singură trecere, de fiecare dată când ajunge la un output, se întoarce înapoi în rețea pentru a ajusta greutățile și astfel eroarea scade cu fiecar epoch.*

*Analogie: Citești laboratorul de Machine Learning o singură dată -> nu rămâi cu prea multe.*

*Citești de mai multe ori, de fiecare dată mai înveți câte ceva.*

1. Ce sunt greutățile (weights) într-o rețea neuronală?

*Parametrii învățați de rețea care determină cât de importantă este fiecare caracteristică în luarea deciziei.*

***Să urmărim acest exercițiu:***

*Să luăm primul rând din dataset-ul train\_processed.csv (solutii/lab3\_out\_train) și considerăm primele 3 feature-uri:*



*Se consideră că fiecare neuron din arhitectura MLP calculează ieșirea după această formula:*

*z=*

*fie w1 = 0.8, w2 = -0.5, w3 = 0.3 și bias = 0.2; Aplicăm aceste valori pe formula de mai sus și obținem:*

*z = -0.30713, aplicăm ReLU, deci ReLU(z) = 0 Neuronul se dezactivează. Pentru al 2-lea epoch, greutățile se actualizează prin backpropagation si optimizer (Adam).*

## Exercițiul 5.2: Batch, Epoch și numărul de pași de antrenare

Pe baza dataset-ului prelucrat în laboratorul 3, avem în felul următor:

1. Câte exemple rămân efectiv pentru training și câte pentru validation?

*Completează:*

*Ntrain = 80k*

*Nvalidation = 20k*

1. Câte batch-uri există într-un epoch?

*batches\_per\_epoch = =*

1. Câte actualizări de greutăți face modelul într-un epoch?

*1 batch = 1 update de greutăți 1250 actualizări per epoch*

1. Câte actualizări totale de greutăți face modelul în 100 de epoci?

*Totalupdates = batches\_per\_epochs = 1250 100 = 125k updates*

## Exercițiul 5.3: Vizualizarea curbelor de învățare

1. Ce este training loss? Cum evoluează acesta?

*Training loss reprezintă eroarea modelului pe datele de antrenare (80%).*

, unde   
;

*loss(epoch) =*

*Loss-ul pe 1 epoch este egal cu loss-ul pe toate batch-urile din acel epoch.*

*Training loss scade constant pe măsură ce cresc epoch-urile. Modelul învață progresiv și își ajustează greutățile pentru a reduce eroarea pe datele de antrenare.*

1. Ce este validation loss? Cum evoluează acesta?

*Validation loss reprezintă eroarea modelului pe datele de validare (20%),* *adică pe date pe care modelul NU le folosește pentru actualizarea greutăților. Formulele sunt la fel ca exercițiul de mai sus.*

|  |  |
| --- | --- |
| ***Training loss*** | ***Validation loss*** |
| *Se calculează pe datele de antrenare* | Se calculează pe datele de validare |
| *Se actualizează greutățile* | *NU se actualizează greutățile* |
| *Scade în mod constant* | *Poate să crească* |

*Modelul învață corect daca ambele loss-uri scad. Până la epoch 3-4, modelul generalizează bine, după epoch 4, validation loss crește, deși training loss scade, concluzie că modelul începe să învețe pe de rost datele și performanța scade.*

*Best epoch = argmin(validation\_loss)*

1. Ce este training accuracy? Cum evoluează acesta?

*Accuracy = , Se face pe batch, iar apoi pe epoch este media aritmetică a accuracy-urilor pe batch.*

*Din grafic observăm:*

* + *Crește rapid la început → modelul învață relațiile de bază*
  + *Crește mai lent ulterior → ajustări fine ale greutăților*
  + *Tinde spre plafon (~0.995) → modelul aproape a învățat tot ce poate din datele de training*

*În porțiunea unde training accuracy crește și validation accuracy scade, modelul face overfitting, memorează datele și scade capacitatea de detecție.*

1. Ce este validation accuracy? Cum evoluează acesta?

*Validation accuracy reprezintă proporția exemplelor din* ***setul de validare*** *clasificate corect de model.* *Validation accuracy este indicatorul real al performanței modelului.*

## Exercițiul 5.4: Evaluare și matrice de confuzie

1. Notați parametrii TN, FP, FN, TP și explicați ce reprezintă.

***TN (Normal → Normal)*** *= 13389 Trafic normal clasificat corect*

***FP (Normal → Attack)*** *= 111 Trafic normal detectat ca fiind atac*

***FN (Attack → Normal)*** *= 80 Atac clasificat ca trafic normal*

***TP (Attack → Attack)*** *= 11615 Atac clasificat atac*

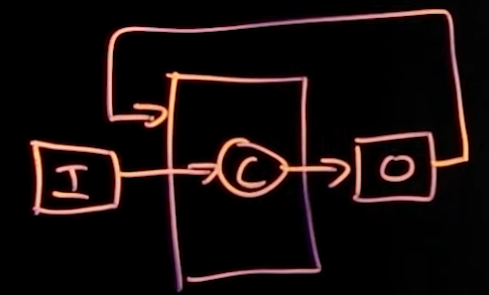
1. Calculați manual Accuracy:

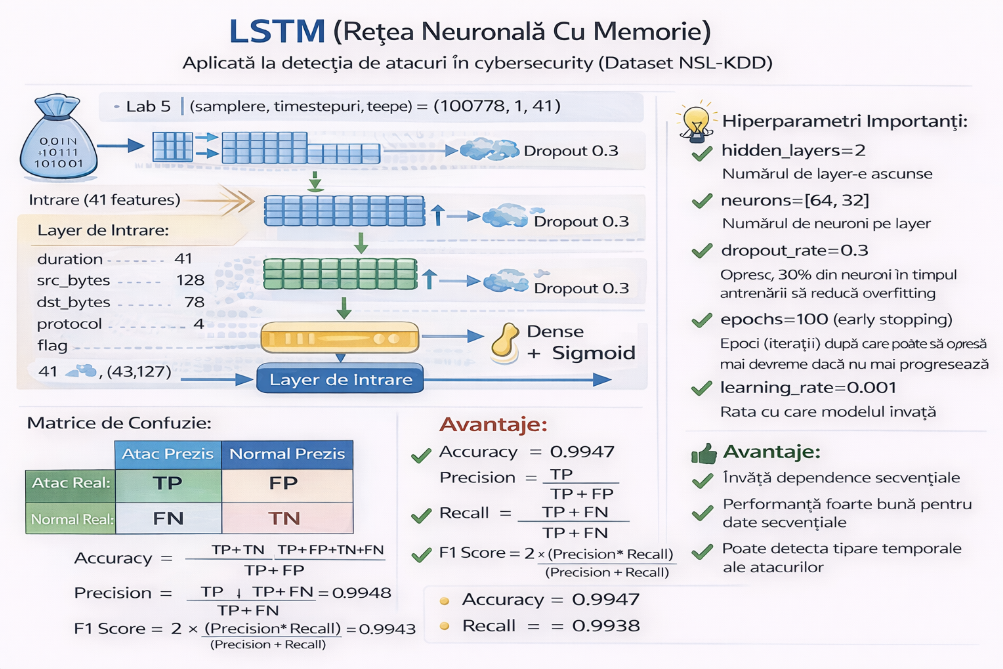
*Acurracy = =*

## Exercițiul 5.5: Arhitectura LSTM

LSTM = Celula Long-short term memory este un tip de celulă recurentă utilizată în arhitecturile de tip Recurrent Neural Network (RNN).

La fiecare pas, se schimbă input-ul deoarece între ieșire și intrare este o buclă:





1. Rulați codul și notați accuracy-ul LSTM-ului, precision, recall și scorul f1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***LSTM*** | ***Accuracy*** | ***Precision*** | ***Recall*** | ***F1-Score*** |
|  | *0.9958* | *0.9957* | *0.9953* | *0.9955* |

1. Explicați forma input-ului pentru LSTM

*În cod, avem: X\_train\_lstm = X\_train.reshape((X\_train.shape[0], 1, X\_train.shape[1]))*

*(samples, timesteps, features)=(100778, 1, 41), unde*

1. Vizualizați matricea de confuzie

***TN (Normal → Normal)*** = 13419

***FP (Normal → Attack)*** = 55

***FN (Attack → Normal)*** = 50

***TP (Attack → Attack)*** = 11671

1. Comparați modelele MLP și LSTM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Model* | *Accuracy* | *Precision* | *Recall* | *F1-Score* |
| *MLP* | 0.992419 | 0.993159 | 0.990534 | 0.991845 |
| *LSTM* | *0.995833* | *0.995734* | *0.995310* | *0.995522* |

## Întrebări de verificare - Lab 5

1. Ce rol are funcția de activare ReLU?
2. De ce folosim Dropout și cum funcționează?
3. Ce se întâmplă dacă learning rate e prea mare?
4. Cum interpretăm curbele de învățare pentru a detecta overfitting?