Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

Adozione di Tecniche di Machine Learning per l'Identificazione di Intrusioni nelle Reti di Telecomunicazioni

Relatore: Dott. Marco Savi

Correlatore: Jacopo Talpini

Relazione della prova finale di:
Urbani Nicolò
Matricola 856213

Anno Accademico 2021-2022

Abstract

Negli ultimi anni per proteggere le Reti di Telecomunicazioni da attacchi sempre più frequenti e sofisticati, sono stati progettati e sviluppati da parte della comunità scientifica diversi metodi per rilevare le intrusioni. Fra le soluzioni più diffuse ed efficaci vi sono diversi strumenti tra cui i Network Intrusion Detection System (NIDS) basati su tecniche di Machine Learning. L'utilizzo di tali sistemi NIDS si è reso necessario per fornire reti più sicure e mitigare i costi legati agli attacchi.

Questa relazione descrive l'attività di stage il cui scopo è stato l'analisi e il confronto di diverse tecniche di Machine Learning per l'identificazione e la classificazione degli attacchi, basata sull'analisi del traffico di rete, finalità particolarmente rilevante per la sicurezza nelle Reti di Telecomunicazioni, in particolare in ambito IoT(Internet of Things).

Sono stati presi in considerazione più dataset al fine di valutare le performance su attacchi e ambienti diversi. I dataset utilizzati sono NF-ToN-IoT-v2 e NF-CSE-CIC-IDS-2018-v2 (nuove versioni dei dataset ToN-IoT e CSE-CIC-IDS-2018), basati sull'insieme di feature collezionate per mezzo del protocollo NetFlow. Tali dataset, ampiamente utilizzati in letteratura, sono recenti e realistici, e sono stati estratti da ambienti di rete che simulano situazioni reali.

Sono state esplorate varie tecniche di Machine Learning fra cui Alberi di Decisione, Random Forest e Reti Neurali con lo scopo di confrontarle tra di loro. Tali modelli sono stati applicati al dataset NF-ToN-IoT-v2 al fine di valutare le performance e identificare il modello di classificazione più adeguato. Le performance (F1-Score) raggiunte dai vari modelli sono confrontabili con con quelle ottenute dai lavori dello stato dell'arte.

Dopo aver scelto il modello migliore, le Random Forest, si procede valutando le performance combinando i dataset NF-ToN-IoT-v2 e NF-CSE-CIC-IDS-2018-v2 al fine di valutare il comportamento del modello su dataset più complessi e con origine diversa. Inoltre, ulteriori analisi consentono di valutare il modello scelto in situazioni non ideali, per esempio quando il modello viene testato su una tipologia specifica attacco non presente durante l'addestramento. Combinando più dataset, nonostante l'addestramento del modello sia più complesso, nella maggior parte dei casi i flussi vengono classificati correttamente. Le ulteriori analisi svolte permettono di concludere che nel caso in cui il modello non è allenato su uno schema di attacco specifico, gli attacchi non vengono classificati correttamente.

Infine, si è utilizzato SHAP, una metodologia di explainable AI, per spiegare ed interpretare le decisioni di classificazione delle Random Forest sul dataset NF-ToN-IoT-v2. Considerata la complessità del modello, i valori Shapley delle feature vengono analizzati per determinare il contributo di ogni feature rispetto alla predizione finale. Dalle analisi svolte emerge che la feature che influisce maggiormente nella classificazione è *Longest Flow Packet*, cioè la lunghezza (in byte) del pacchetto più grande trasmesso nel flusso. L'utilizzo del metodo SHAP su una classe classe specifica ha permesso di identificare le feature più rilevanti nel processo di classificazione dei singoli attacchi.

Indice

1 Introduzione

Le Reti di Telcomunicazioni sono sempre più complesse, lo sviluppo tecnologico ha introdotto nuovi dispositivi connessi fra cui i device IoT(Internet of Things).

L'IoT è un ecosistema di dispositivi interconnessi, dotati di sensori per raccogliere, immagazzinare e scambiare dati attraverso la rete. E' stato stimato che il numero di dispositivi IoT raggiungerà i 41,6 miliardi nel 2025. La notevole diffusione di tali dispositivi rende ancora più complesso mantenere le reti sicure per proteggere i dati sensibili [11].

Sono state proposte diverse soluzioni per garantire la sicurezza nelle reti fra cui i NIDS (Network Intrusion Detection System), tali sistemi utilizzano il Machine Learning per identificare e classificare gli attacchi.

Diverse tecniche di Machine Learning (ML), descritte nel Capitolo 2, sono in grado di apprendere i pattern complessi che potrebbero essere malevoli. Tutte le intrusioni nelle reti generano un insieme di eventi che aiutano nel processo di classificazione, infatti, tali eventi possono essere raccolti in un dataset ed etichettati secondo il tipo di attacco. Le etichette (*Label*) consentono di distinguere il traffico benigno dal traffico malevolo. [13]

Sono disponibili molti dataset utili per costruire modelli di Machine Learning. Tali dataset si differenziano per tipologia di attacco, distribuzione delle classi e ambiente di rete in cui è stato costruito. I dataset utilizzati per le analisi vengono descritti nel Capitolo 3.

1.1 Intrusion Detection System e Tipologie di Attacco

Gli Intrusion Detection System sono sistemi (automatici e non) che analizzano i dati tracciati (auditing trails) per determinare/prevenire tentativi di compromissione o di uso non autorizzato di un sistema [3].

Lo sviluppo di sistemi NIDS in grado di riconoscere ed evitare attacchi informatici è sempre più complesso, nuovi attacchi vengono effettuati sempre più frequentemente. Pertanto, è fondamentale sviluppare IDS efficaci per mantenere le reti sicure e di conseguenza le informazioni riservate.

1.1.1 Network Intrusion Detection System

L'obiettivo dei Network Intrusion Detection System (NIDSs) è individuare gli attacchi e preservare i tre principi di sicurezza delle informazioni: confidenzialità, integrità e disponibilità [8].

Le reti sfruttano la commutazione di pacchetto, i dati da trasmettere vengono suddivisi in pacchetti e inviati sul mezzo trasmissivo. Nella trasmissione a pacchetto il flusso di dati viene suddiviso in unità più piccole, ognuna delle quali contiene metadati, l'origine della trasmissione, la destinazione e il contenuto. Ogni pacchetto viene trasmesso sulla rete tramite il livello Network e formattato in un protocollo adeguato tramite il livello di trasporto, l'informazione viene ricostruita dal livello applicativo [3].

I pacchetti utilizzati dai protocolli di comunicazione del modello ISO/OSI possono essere estratti dalla rete con strumenti di sniffing come *Wireshark*. Gli sniffer lavorano nel secondo livello del modello ISO/OSI, livello data-link, infatti, tramite schede di rete (NIC) impostate in modalità promiscua sono in grado di visualizzare tutto il traffico con i relativi header di livello 2, 3, 4 e livello applicativo con informazioni sui protocolli HTTP, TCP, etc. [6].

Lo *sniffing* ha diverse applicazioni pratiche fra cui individuare eventuali anomalie nelle reti, analizzare le performance, carpire informazioni e monitorare il traffico in rete. I sistemi NIDS monitorano la rete per raccogliere informazioni da analizzare per individuare eventuali attacchi. Il funzionamento di un IDS è mostrato in figura 1.1, l'interfaccia dello switch viene impostata in modalità mirroring, tutto il traffico che transita dallo switch viene inoltrato alla porta dello sniffer.

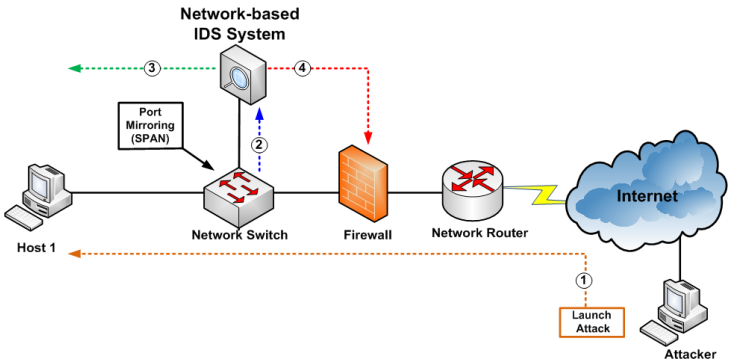


Figure 1.1: Rete con Intrusion Detection System [19]

Lo sniffer si occupa di raccogliere i dati in file, solitamente *pcaps* file. Tramite i flussi estratti dalla rete è possibile estrarre informazioni, tra cui l'IP sorgente, l'IP destinazione, le porte e la dimensione del pacchetto. Infatti, queste informazioni sono disponibili in chiaro negli header di ogni pacchetto, mentre il contenuto solitamente è criptato.

Le informazioni raccolte vengono utilizzate dall'IDS per rilevare se è in corso un attacco o meno. Inoltre, i dati raccolti, appositamente etichettati, possono essere utilizzati per costruire dataset su cui allenare l'IDS nel riconoscimento degli attacchi.

Raccogliere e memorizzare il traffico di rete è importante per un'organizzazione per monitorare, analizzare ed eventualmente rivedere la struttura della propria rete aziendale.

I Sistemi di Intrusione in combinazione con i firewall sono di fondamentale importanza per garantire la sicurezza in rete. I firewall lavorano attivamente bloccando i pacchetti che secondo le regole di filtraggio non sono consentiti, mentre gli IDS lavorano passivamente segnalando pacchetti sospetti che potrebbero determinare un attacco. I firewall tramite regole statiche, definite precedentemente cercano di prevenire gli attacchi facendo un'analisi meno approfondita basandosi solamente su IP e porte. Mentre, gli IDS analizzano approfonditamente le lunghezza della trasmissione, il numero di pacchetti nel flusso e altre informazioni sui flussi di rete. Tali informazioni consentono di identificare i flussi di dati sospetti che potrebbero determinare un attacco. Gli IDS sono efficaci anche nel caso di attacchi provenienti dall'interno della rete, mentre i firewall rilevano solamente pacchetti sospetti provenienti dall'esterno [3].

I sistemi di intrusione possono operare in tempo reale individuando gli attacchi e segnalandoli oppure tenere traccia degli attacchi e mostrarli all'utente in seguito.

I NIDS possono fare uso di tecniche di Supervised Machine Learning per identificare gli attacchi, tali tecniche vengono descritte in seguito.

1.1.2 Tipologie di Attacchi

Al fine di comprendere al meglio il funzionamento di un NIDS e capire come sono stati riconosciuti gli attacchi è fondamentale capire in cosa consiste ciascun attacco e quali sono le sue peculiarità.

Gli attacchi si suddividono in due categorie: passivi e attivi. Gli attacchi passivi non modificano nè generano flussi di dati. Gli attacchi passivi vengono sfruttati per ottenere informazioni oppure fare delle attività di ricognizione. Un esempio è il Port Scanning che l'attaccante utilizza per identificare quali porte sono aperte e i relativi servizi attivi. Mentre, gli attacchi attivi non si limitano ad intercettare i flussi di rete. Infatti, modificano il flussi di rete, generano traffico malevolo oppure rendono il servizio non disponibile [3]. Negli anni l'aumento di sistemi di difesa nelle Reti di Telecomunicazioni ha reso necessario sviluppare attacchi sempre più complessi e variegati.

Gli attacchi di tipo passivo violano il principio di confidenzialità, le informazioni vengono intercettate da utenti non autorizzati. Nei dataset descritti in seguito gli attacchi di questo tipo sono [13]:

- **MITM** Man In The Middle è un metodo che posiziona l'attaccante fra due host, l'obiettivo è di intercettare il traffico nel canale trasmissivo. Esistono diverse tecniche per effettuare questo attacco fra cui: ARP Cache Poisoning, ICMP redirect e Port-Stealing. L'ARP Cache Poisoning consiste nel modificare le ARP Table per inoltrare tutto il traffico all'attaccante, ad ogni IP nella tabella viene associato il MAC dell'attaccante. L'ICMP redirect è un attacco che sfrutta i messaggi ICMP,

particolari messaggi del livello di rete, che consentono a un router di informare un host che esiste un percorso più efficiente verso una destinazione. Viene suggerito all'host di modificare di conseguenza la sua tabella di routing, tramite questo meccanismo i messaggi vengono reindirizzati all'attaccante [4]. L'attacco di Port-Stealing spesso viene effettuato quando l'ARP Cache Poisoning non è possibile, l'attacco "ruba"la porta dello switch di ogni vittima, per sottrarre i dati.

- **Scanning** Consiste in un insieme di tecniche che hanno l'obiettivo di scoprire informazioni riguardo la rete e i dispositivi connessi. Tale attacco, conosciuto anche come probing, viene spesso utilizzato come fase preliminare all'attacco vero e proprio. Lo scanning consente di identificare le debolezze della rete da sfruttare per l'attacco.

- **Backdoor** Tale tecnica consente all'attaccante di ottenere un accesso remoto non autorizzato ai dispositivi IoT infettati da un malware Backdoor. L'attaccante utilizza la backdoor per controllare i dispositivi IoT infetti e li rende parte di botnet per tentare attacchi DDoS.

Gli attacchi di tipo attivo violano oltre alla confidenzialità, anche l'integrità e la disponibilità. L'integrità garantisce l'attendibilità e la non modificabilità delle informazioni trasmesse. Mentre, la disponibilità garantisce l'accesso alle risorse in ogni momento in cui l'utente autorizzato ne necessiti.

Nei dataset descritti in seguito gli attacchi di questo tipo sono [13]:

- **Dos** Denial Of Service è un tentativo di sovraccaricare le risorse di un sistema con l'obbiettivo di negare l'accesso al sistema o alle risorse stesse. Questo attacco può essere effettuato inondando i dispositivi IoT di destinazione con un grande numero numero di connessioni in modo da esaurire le risorse del dispositivo (ad esempio, CPU e memoria). I dispositivi IoT hanno una potenza di calcolo e una capacità di memorizzazione limitata, il che li rende facilmente vulnerabili agli attacchi DoS [1]. Per effettuare questa tipologia di attacco sono disponibili diversi tool fra cui: *DoS attacks-GoldenEye, DoS attacks-Hulk, DoS attacks-SlowHTTPTest, DoS attacks-Slowloris*. Gli ultimi due tool lasciano che una singola macchina mantenga aperte connessioni con larghezza di banda minima, le risorse del server attaccato vengono consumate rapidamente finchè il server non sarà più raggiungibile [14].

- **DDos** Distibuted Denial of Service è una attacco simile a Dos, ma l'attacco proviene da più sorgenti. Un attacco di questo tipo è spesso il risultato di più sistemi di compromessi (per esempio, una Botnet) che inondano il sistema bersaglio generando un enorme traffico di rete. Fra i tool disponibili si hanno: *DDOS attack-HOIC, DDOS attack-LOIC-UDP, DdoS attacks-LOIC-HTTP*. Tali tool inviano molte richieste UDP, TCP, o HTTP al server vittima.

- **Injection** Un insieme di attacchi che forniscono input malevoli per alterare la normale esecuzione. Fra gli attacchi di tipo injection abbiamo: SQL Injection e XSS. L'attacco di tipo SQL Injection consiste nell'alterare lo stato di un database SQL tramite query per ottenere maggiori privilegi o modificare il database. L'attacco

XSS (Cross Site Scripting) consiste nell'utilizzare le applicazioni web per eseguire script malevoli. Tali script, sono in grado di alterare le risorse del server oppure inviare all'attaccante informazioni riservate [1].

- **Password** Comprende un insieme di attacchi per ottenere password tramite brute force o sniffing.

- **Ransomware** Un attacco che codifica i file memorizzati su un host e richiede una somma, spesso in cryptovalute, per decriptare i dati. Nel caso in analisi per effettuare l'attacco viene utilizzato il tool *Metasploitable3*.

- **Bot** Un attacco che consente all'attaccante di controllare da remoto dispositivi compromessi al fine di effettuare attività malevole.

- **Brute Force** E' uno degli attacchi più popolari che può essere usato non solo per violare le password, ma anche per scoprire pagine e contenuti nascosti in un'applicazione web. Si tratta fondamentalmente di un attacco "hit and try ", nel quale si fanno più tentativi finchè non si ottiene quanto voluto.

- **Infilteration** Un attacco interno che invia un file malevolo tramite mail per sfruttare la vulnerabilità di un'applicazione, tale attacco è seguito da una backdoor che analizza la rete per individuare ulteriori debolezze.

1.2 Supervised Machine Learning Classification

Il termine Machine Learning (ML) si riferisce ad algoritmi e processi che "imparano", cioè sono in grado di generalizzare dati ed esperienze passate per predire risultati futuri [3]. Gli umani utilizzano un insieme di strategie cognitive per dare un senso al mondo che li circonda. Una delle capacità più utili è assegnare ad oggetti delle categorie specifiche, basandoci su caratteristiche.

Per esempio, trovandoci di fronte a un oggetto ovale di colore giallo, morbido all'interno, e con un odore dolce e pungente, dovremmo fare riferimento alla nostra conoscenza passata per predire che appartiene alla categoria "frutta". Possiamo testare l'accuratezza di quanto predetto portando l'oggetto in un supermercato, se fra la frutta troviamo frutti simili etichettati come "mango"possiamo concludere che la nostra predizione è corretta. Inoltre, possiamo generalizzare la nostra conoscenza della frutta stabilendo che il mango ha un buon sapore e buoni valori nutrizionali. Il processo di assegnare una categoria è chiamato classificazione [17].

Le tecniche di Supervised ML utilizzano la probabilità di un evento precedente per inferire la probabilità di un nuovo evento. Questo metodo può essere utilizzato per i problemi di classificazione oppure di regressione (predire una proprietà numerica di un campione) [3].

Per classificare utilizzando un approccio supervisionato si parte da un insieme di dati

etichettati, di cui è già stata stabilita la classe di appartenenza. Il compito del classificatore è quello di stabilire come le *feature* di ogni classe possono essere utilizzate per prevedere la classe di nuovi campioni non etichettati [17]. Per ogni categoria vengono identificate delle caratteristiche particolari, questa fase è denominata training del modello. Presentando al modello nuovi campioni l'algoritmo fa riferimento ai dati visti in precedenza e stabilisce una classe per il campione in input.

Per costruire un modello accurato, è necessaria una buona quantità di dati etichettati con la classe corretta. Questi dati solitamente vengono suddivisi in tre set distinti: training, validation e testing. Il training è il set più ampio, più dati contiene più il classificatore produrrà un modello accurato [17]. Le fasi per costruire un classificatore sono le seguenti [17]:

1. Nella fase di **training** il classificatore viene applicato all'insieme di dati per costruire il modello. I modelli a disposizione forniscono un insieme di iperparametri che possono essere modificati per controllare come costruire il modello.

2. Nella fase di **validation** il modello viene valutato sul validation set, se l'accuratezza non è sufficiente gli iperparametri vengono modificati. Si ripete la procedura finchè non si raggiunge il grado di accuratezza desiderato.

3. Nella fase di **testing** per valutare l'accuratezza, il set di test viene applicato al modello. Si confrontano le classi predette con quelle effettive. Se il risultato rispetta le performance attese, si procede con la fase di *deployment*. In caso contrario si ritorna alla fase di training per migliorare il modello.

4. Nella fase di **deployment** il modello viene applicato per predire le classi di dati non etichettati, su cui il modello non è allenato.

Gli algoritmi di Machine Learning sono basati su matematica e statistica, tali algoritmi sono in grado di identificare pattern, correlazioni, e anomalie nei dati. Infatti, tali algoritmi sono ampiamente diffusi nell'ambito della sicurezza informatica, per esempio sono utilizzati per identificare le mail contenti SPAM [3]. Nel nostro caso utilizzeremo queste tecniche di Machine Learning per identificare e classificare possibili attacchi nelle Reti di Telecomunicazione.

1.3 Machine Learning per NIDS

I Network Intrusion Detection System (NIDS) usano ampiamente tecniche di Machine Learning per identificare e classificare gli attacchi. Il Machine Learning (ML), un campo dell'Intelligenza Artificiale, è in grado di imparare ed estrarre pattern di attacco, tali attacchi se non identificati potrebbero compromettere la sicurezza della rete [8].

Con il termine pattern si intendono caratteristiche specifiche presenti nei flussi di rete di un determinato tipo di attacco. Per esempio, se nei dati si nota che un singolo indirizzo IP sta effettuando oltre 20 richieste al secondo al web server in un periodo di 5 minuti, probabilmente è un attacco DoS. Questa particolarità genera un pattern, che viene utilizzato dal modello per allenare l'algoritmo di ML [3].

Il modello viene allenato su un insieme di dati etichettato con la classe di tale flusso. Il modello apprende quali sono i flussi malevoli e a quale attacco appartengono, il comportamento di un attacco viene associato ad un pattern specifico. Maggiore è la quantità di dati su cui il modello è allenato, maggiore è la probabilità che il modello identifichi correttamente la classe su dati mai visti prima. Il modello allenato verrà utilizzato per predire la classe di nuovi flussi di rete [2].

Gli Intrusion Detection System possono utilizzare diverse tecniche di Machine Learning, le più rilevanti nell'ambito della sicurezza vengono descritte nel prossimo capitolo 2.

2 Modelli di Classificazione per NIDS

Sono disponibili diverse tecniche di classificazione supervisionata utili nell'ambito dei sistemi di intrusione. Le principali vengono introdotte nel capitolo seguente.

2.1 Alberi Di Decisione

Gli Alberi di Decisione (Decision Tree) sono modelli di apprendimento supervisionato molto versatili con l'importante caratteristica di essere semplici da interpretare [3]. Tali modelli hanno un buon grado di *explanation,* ossia le scelte di classificazione fatte dal modello sono facilmente spiegabili all'utente finale. Un Decision Tree, come suggerisce il nome, è basato su una successione di decisioni binarie utilizzate per prendere una decisione, cioè scegliere come classificare un elemento dato in input.

La capacità di predire valori categorici (Classification Trees) e valori reali (Regression Trees) è spesso utilizzata nel Machine Learning, infatti, l'algoritmo CART (Classification and Regression Trees), è in grado di generare sia alberi di regressione che di classificazione. La libreria Python scikit-learn, una delle più importanti librerie di Machine Learning, implementa l'algoritmo CART. [17]

L'algoritmo per stabilire se un elemento, rappresentato da un insieme di coppie feature-valore, appartiene a una classe oppure ad un'altra è basato sulla definizione di una sequenza "if-then-else" [17].

Utilizzando tale principio, un albero di decisione viene costruito seguendo i seguenti passi [2]:

1. Partendo dalla radice dell'albero, l'intero dataset viene suddiviso in due sottoinsiemi in base a una condizione binaria. Tale condizione viene stabilita esaminando tutti i campioni disponibili e seleziona la feature migliore come variabile di split. Il valore di split viene stabilito partendo dal range dei valori associati alla feature, scelta precedentemente.

2. L'algoritmo per stabilire quale è la migliore feature e il relativo valore di soglia, su cui effettuare lo split, valuta la riduzione di impurezza, misura di quanto lo split consente di ottenere elementi appartenenti alla stessa classe. Ci sono diverse misure per valutare la qualità dello split fra cui *Entropia*. L'entropia assume valore zero quando un ramo dell'albero contiene solamente istanze delle stessa classe. Un'altra

misura di purezza è rappresentata dall'*indice di Gini*, entrambi i criteri di split spesso producono alberi simili. L'entropia consente di produrre alberi bilanciati, ma in tempi maggiori rispetto all'*indice di Gini*.

La coppia k (feature) e t (threshold) che permette di ottenere sottoinsiemi più puri viene scelta per suddividere il dataset [2].

3. Stabilita la condizione di split partendo dalla coppia (k, t), gli elementi del dataset per cui la condizione è vera vengono posizionati nel lato sinistro. Mentre, i campioni che non verificano la condizione vengono posizionati nel lato destro.

   Per esempio se viene scelta come feature (k) l'età e, utilizzando i valori del range di età disponibili, si stabilisce come soglia di split il valore (t) 18, le persone che avranno come caratteristica un'età minore o uguale a 18 verranno collocate nel lato sinistro dell'albero, gli individui con un'età superiore al valore soglia, nel lato destro dell'albero.

4. I sottoinsiemi ottenuti, rappresentati dai due rami dell'albero, possono essere ricorsivamente partizionati utilizzando altre condizioni, selezionate automaticamente, al fine di ottenere lo split migliore.

5. L'algoritmo ripete lo stesso processo più volte ad ogni foglia ottenuta finchè l'albero è "saturo", ogni nodo è perfettamente puro o un ulteriore split non riduce l'impurità, oppure se l'albero raggiunge una condizione che rispecchia quanto specificato nel settaggio degli iperparametri (es numero massimo di foglie o massima profondità).

6. Il risultato ottenuto è un albero binario dove ogni nodo rappresenta una decisione binaria, i figli di ogni nodo rappresentano i due possibili risultati di tale decisione. Le foglie rappresentano la classificazione dei samples di partenza, seguendo il percorso dalla radice alla foglia è possibile ricostruire le caratteristiche dell'elemento classificato. Considerando l'esempio precedente, per raggiungere il nodo foglia "Maschio Maggiorenne" è necessario, partendo dalla radice, attraversare due nodi, il primo nodo dove viene stabilito se il vettore fornito in input è di sesso maschile o femminile e il secondo per stabilire se è maggiorenne o minorenne, percorrendo i rami dell'albero l'algoritmo termina in un nodo foglia. Tale foglia rappresenta una classe di elementi, le cui caratteristiche sono simili [17] [3].

Si ottengono buone performance anche con grandi quantità di dati, gli alberi di decisione sono efficienti nell'allenare e fare previsioni poichè ogni sample al massimo attraversa la massima altezza del albero di decisione con complessità temporale è $O(n \log n)$ [17].

Nonostante ciò, gli Alberi Decisionali hanno delle limitazioni, spesso soffrono di overfitting: l'albero è troppo complesso e non generalizza bene con i dati di test. Per evitare tale problema è necessario ottimizzare i punti di split, per produrre un modello in grado di generalizzare e classificare correttamente i nuovi input su cui il modello non è stato allenato [3].

L'ottimizzazione degli iperparametri consente di restringere la libertà durante l'allenamento

dell'albero, per esempio restringendo la massima profondità dell'albero [2]. Alcuni iperparametri utili per regolarizzare il modello sono [17]:

- max-depth: determina il numero massimo di livelli permessi prima che la costruzione dell'albero termini
- min-samples-split determina il numero minimo di samples che un nodo deve avere per effettuare lo split
- min-samples-leaf determina il numero minimo di elementi per creare un nodo foglia
- max-leaf-nodes determina quanti nodi possono essere creati in totale

Per esempio, ridurre max-depth consente di regolarizzare il modello e quindi ridurre il rischio di overfitting.

Un altro iperparametro utile per migliorare il modello è *criterion*. Tale parametro può assumere valore *Gini Impurity* oppure *Entropy*. La libreria Python *scikit-learn* di default utilizza l'indice di Gini.

L'entropia assume valore zero quando il nodo contiene istanze di una sola classe. Tale misura di purezza con k classi a profondità i è definita come [2]:

$$\mathrm{H}_i = -\sum_{k=1}^{n} \mathrm{p}_{i,k} \log(\mathrm{p}_{i,k})$$

$\mathrm{p}_{i,k}$: è il rapporto tra le istanze della classe k e le istanze di addestramento nell'iesimo nodo

Mentre la *Gini Impurity* è definita come [2]:

$$\mathrm{G}_i = 1 - \sum_{k=1}^{n} \mathrm{p}_{i,k}^2$$

Entrambi i criteri spesso portano a risultati simili, si ottengono alberi comparabili. Testando il modello con entrambi è possibile valutare quale criterio porta a risultati migliori [2].

Gli alberi di decisione hanno un'importante caratteristica, ossia avere una buona interpretabilità, infatti, ogni predizione può essere ricondotta in una serie di condizioni booleane, costruite partendo dalla radice dell'albero fino alla relativa foglia.

La semplicità del modello permette di rappresentare graficamente gli alberi di decisione, dalla rappresentazione è possibile capire perchè un'istanza è stata classificata in una

specifica classe. Come mostrato in figura 2.1 vengono mostrate le feature di split con le relative soglie e le misure di impurezza. La rappresentazioni migliora la spiegabilità del modello costruito.

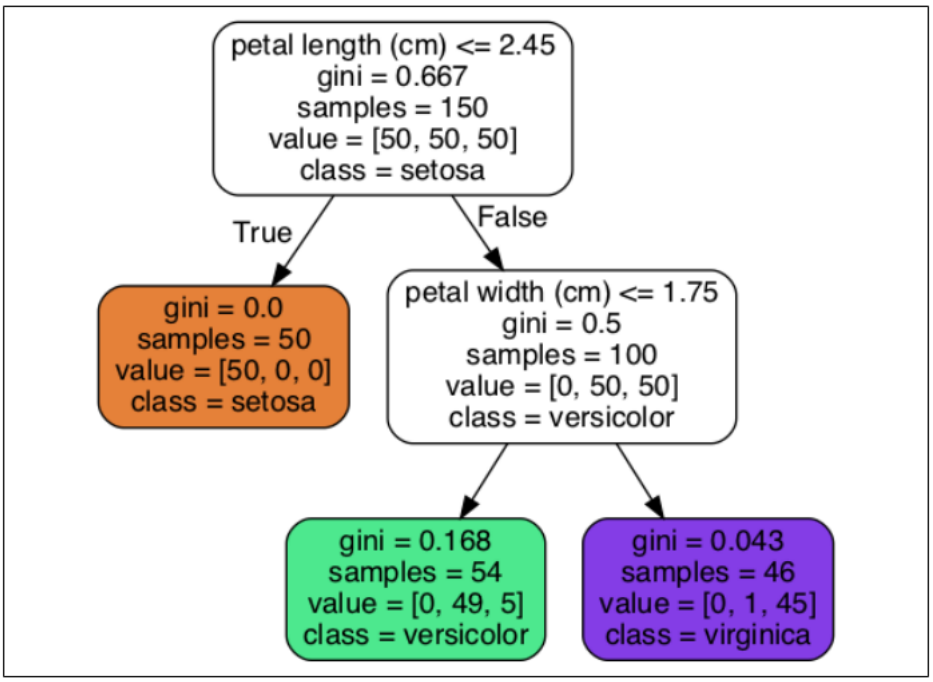


Figure 2.1: Decision Tree IRIS dataset [2]

Gli alberi di decisione potrebbero essere instabili. Piccole variazioni nei samples possono produrre alberi completamente differenti [17]. Considerando le limitazioni e i problemi dei Decision Tree, sono stati sviluppati nuovi modelli chiamati *Ensemble of Trees*.

2.2 Ensemble Learning

Il termine "ensemble" si riferisce alla combinazione di più classificatori che genera modelli più complessi e più performanti, questa tecnica è anche chiamata Ensemble Learning [3].

Combinando più alberi di decisione si possono ad esempio ottenere le Random Forest, uno fra i più potenti algoritmi di Machine Learning disponibili [2].

Le Random Forest sono basate sul principio di "wisdom of the crowd", la risposta di molti è meglio della risposta di un solo esperto, allo stesso modo, aggregando le predizione di un gruppo di predittori (come classificatori o regressori), si ottengono predizioni migliori rispetto al singolo predittore. Spesso, gli alberi di decisione singoli producono overfitting rispetto al training set, le Random Forest cercano di mitigare questo effetto, combinando i singoli alberi costruiti.

Inoltre, ogni albero può essere allenato indipendentemente dal resto degli alberi, pertanto l'algoritmo può essere parallelizzato, sfruttando tutta la potenza computazionale

a disposizione, utilizzando diversi CPU cores oppure server differenti, il training e la predizione effettuati in maniera non sequenziale aumentano l'efficienza dell'algoritmo. L'aumento di complessità delle RF (Random Forest) rende le predizioni difficili da spiegare, l'interpretabilità è nettamente minore rispetto agli Alberi di Decisione [3].

Esistono molti approcci per sfruttare le potenzialità delle Random Forest, per esempio utilizzare lo stesso training set con classificatori di tipo diverso e combinare i risultati della predizione considerando il risultato più frequente oppure allenare il modello più volte correggendo gli errori del modello precedente. Questi approcci vengono approfonditi nelle sezioni successive.

2.2.1 Random Forests

Un approccio consiste nell'utilizzare lo stesso algoritmo di training per ogni classificatore, ma allenarlo con differenti sottoinsiemi del training set. Per costruire i sottoinsiemi di training è possibile usare due metodi bagging, con sostituzione (le istanze vengono riutilizzate più volte nei diversi sottoinsiemi) oppure pasting, senza sostituzione [2].

Il funzionamento viene mostrato nell'immagine che segue:

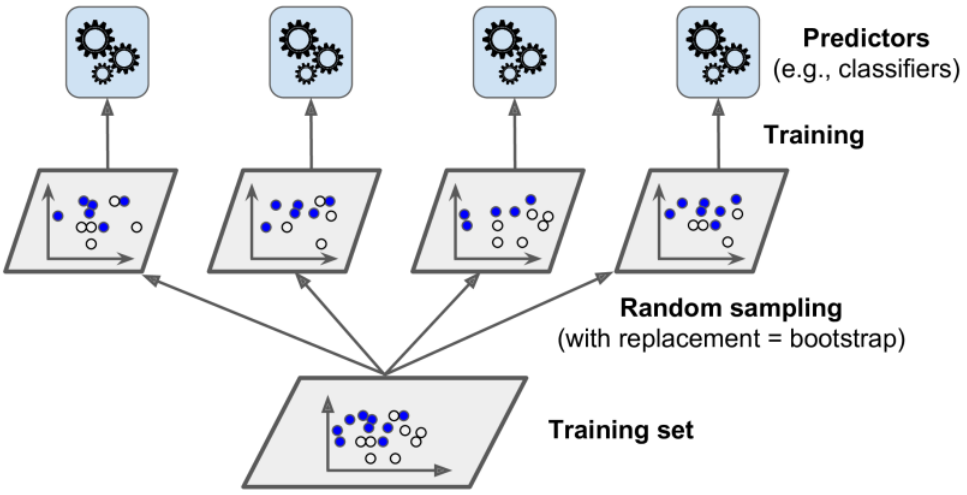


Figure 2.2: Bagging/Pasting training set sampling and training [2]

Quando tutti i predittori sono allenati, la RF può predire una nuova istanza semplicemente aggregando le predizioni dei singoli predittori. Il risultato è dato dalla classe ottenuta del maggior numero di predittori.

Il bagging (bootstrap aggregating) introduce maggiore diversità nel subset dove ogni predittore viene allenato, spesso il bagging costruisce modelli migliori rispetto al pasting. Nonostante ciò, per verificare quale modello è migliore nel caso specifico è possibile effettuare la cross-validation con entrambi bagging e pasting e scegliere quale metodo ottiene risultati migliori.

Le Random Forest, solitamente allenate utilizzando il bagging method, introducono un'ulteriore componente random, non viene scelta la feature migliore quando viene effettuato lo splitting su un nodo, ma viene selezionata la migliore da un set di feature random. Questo porta a una maggiore diversità nei singoli alberi, ogni classificatore costruirà un modello indipendente che potrebbe migliorare l'accuratezza del risultato finale [2]. Gli alberi decisionali tendono all'overfitting, costruendo un modello poco generalizzabile, le Random Forest costruiscono alberi indipendenti che combinati, considerando la classe più votata dal singolo albero, generalizzano meglio migliorando le performance [2].

2.2.2 Extra Trees

Quando viene costruito un albero in una Random Forest, per ogni nodo viene considerato solamente un subset di features (come discusso precedentemente). E' possibile costruire alberi più casuali utilizzando un thresholds (t) random per ogni feature al posto di cercare la soglia migliore (come negli Alberi di Decisione).

Un insieme di alberi "extremely" random è chiamato *Extremely Randomized Trees* (Extra-Trees). Gli Extra-Tree sono molto più rapidi da allenare rispetto alle Random Forest, infatti trovare il migliore threshold per ogni feature ad ogni nodo aumenta notevolmente i tempi computazionali nella costruzione di un albero.

Gli Extra Tree sono un insieme di alberi, come le Random Forest, cambia solo il metodo di costruzione dei singoli alberi. Non è possibile stabilire a priori se una Random Forest ottiene risultati migliori rispetto ad un Extra Tree, generalmente l'unico modo per stabilirlo è effettuare la cross-validation (tuning degli iperparametri tramite grid search) [2].

2.3 Boosted Decision Tree - AdaBoost

Il boosting è un altro esempio di*Ensemble Method* che combina più classificatori "deboli" in uno più potente. L'idea alla base del boosting è allenare i classificatori sequenzialmente, ognuno di essi corregge il precedente. Il modello viene costruito gradualmente imparando dagli errori e cercando di correggerli gradualmente.

AdaBoost, abbreviazione di "Adaptive Boosting," è un algoritmo di *boosting ensemble learning*, è stato uno dei primi approcci di boosting di successo.

Il nuovo predittore corregge il predecessore facendo più attenzione alle istanze di training che soffrono di "underfitting". Il predecessore non è stato allenato per riconoscere e classificare alcuni samples, il nuovo predittore cercherà di individuare tali istanze e allenare al meglio il modello. Il risultato sarà un modello che si concentrerà sempre di più sui casi più complessi. Questa tecnica viene utilizzata da AdaBoost [2].

Per esempio, per costruire classificatore AdaBoost, un classificatore base (come un Decision Tree) viene allenato e usato per fare predizioni sul training set. Il peso delle istanze

di training non correttamente classificate viene incrementato (boosted). Il classificatore successivo viene allenato usando i nuovi si ottiene il classificatore desiderato oppure, predizioni sul training set vengono nuovamente effettuate, i peso vengono aggiornati. L'algoritmo viene ripetuto più volte fino a quando viene raggiunto il numero di modelli desiderato oppure si ottiene il modello perfetto [2].

Quando ciascun predittore è stato allenato, l'algoritmo effettua le predizioni come le Random Forest con bagging o pasting, ma ciascun predittore avrà un peso diverso in base ad un peso, determinato in fase di training. Il numero di alberi del modello e il loro contributo è bilanciato. Molti alberi potrebbero richiedere un learning-rate minore, con pochi alberi maggiore. Solitamente vengono usati valori intermedi fra 0 e 1. [1]

Un'importante iperparametro nell'algoritmo AdaBoost è il learning-rate. Tale parametro consente di controllare il contributo di ogni modello nell' *ensemble* complessivo. Il learning-rate di default è pari a 1, *full contribution*.

Questa tecnica di apprendimento sequenziale non può essere parallelizzata, perchè ogni predittore può essere allenato solamente dopo che il precedente è stato allenato e valutato, come mostrato nell'immagine 2.3.

Pertanto, utilizzando AdaBoost con grandi quantità di dati le performance calano drasticamente.

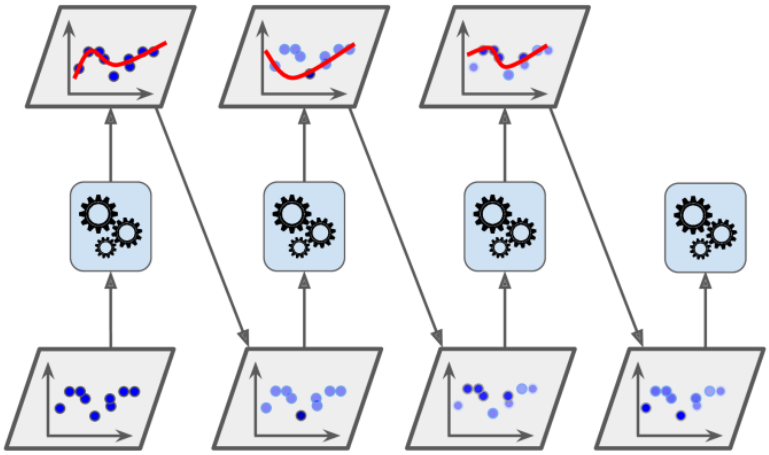


Figure 2.3: AdaBoost sequential training con aggiornamento del peso delle istanze [2]

[1] https://machinelearningmastery.com/adaboost-ensemble-in-python/

2.4 Implementazione Modelli con Scikit-learn

I modelli presentati possono essere implementati in linguaggio Python tramite la libreria scikit-learn. Scikit-learn (Sklearn) è una delle librerie più diffuse e complete per il Machine Learning in Python. Fornisce una selezione di strumenti efficienti per l'apprendimento automatico e la modellazione statistica, tra cui classificazione, regressione e clustering [15].

L'implementazione dei modelli presentati prevede l'utilizzo di alcuni iperparametri, utili per definire il modello [15]:

- `criterion`

  La funzione per misurare la qualità dello split. I criteri supportati sono "gini" per la Gini Impurity e "entropy". La definizione e l'utilizzo di tali misure per costruire gli alberi sono definite nella sezione 2.1. Il valore di default è "gini".

- `class_weight`

  Definisce i pesi associati a ciascuna classe nella forma {class_label: weight}. Se non definiti, tutte le classi hanno peso uno. Se class_weight assume il valore "balanced"i valori di y vengono utilizzati per bilanciare i pesi, in maniera inversamente proporzionale rispetto alla frequenza dei dati di input come *n_samples / (n_classes * np.bincount(y))*

- `random_state`

  Controlla la casualità degli alberi. Di default è impostato a *None*.

In particolare per i metodi Ensemble è possibile definire i seguenti iperparametri [15]:

- `n_estimators`

  Definisce tramite un numero intero il numero di alberi nella foresta. Il valore di default è 100.

- `n_jobs`

  Il Numero di elaborazioni da eseguire in parallelo. *None* significa 1 job per volta. -1 significa utilizzare tutti i processori disponibili al fine di svolgere l'elaborazione.

Nel caso dei Boosted Decison Tree si definisono due parametri particolari [15]:

- `algorithm`

  Se il parametro assume valore "SAMME.R" allora viene utilizzato l'algoritmo di boosting SAMME.R Se assume valore "SAMME" allora viene utilizzato SAMME, un algoritmo di boosting di tipo discreto. L'algoritmo SAMME.R tipicamente con-

verge più rapidamente ripetto all'algoritmo SAMME, che raggiunge un errore di test minore con meno iterazioni di boosting. Il valore di default è "SAMME.R".

- `learning_rate` Tale parametro esprime il peso applicato ad ogni classificatore a ogni iterazione. Un' *learning_rate* più alto aumenta il contributo di ogni classificatore. Ci deve essere un compromesso fra i parametri *n_estimators* e il *lerning_rate*. Il valore di default è 1.0

2.5 Reti Neurali

Nelle sezioni precedenti sono stati mostrati vari modelli di Machine Learning utili per risolvere un problema di classificazione, tali approcci sono relativamente semplici.

Le Reti Neurali utilizzano algoritmi più complessi, basati su più livelli, ogni livello esegue calcoli diversi. I campioni vengono processati di livello in livello, l'output di un livello è l'input del livello successivo. Almeno uno di questi livelli è nascosto, questo distingue il Deep Learning da tutti gli altri metodi [17]. Il termine *Deep Learning* comprende un'ampia gamma di metodi di apprendimento supervisionato e non supervisionato basati sull'uso delle Reti Neurali.

Le Reti Neurali sono una classe di algoritmi così chiamati perchè simulano le modalità di interazione tra reti di neuroni densamente interconnesse nel cervello. L'aumento della potenza computazionale e la diffusione di database di grandi dimensioni hanno portato a una sempre maggiore diffusione delle tecniche di Deep Learning. Infatti, le Reti Neurali sono in grado di risolvere problemi complessi, come categorizzare le immagini oppure il riconoscimento vocale. Inoltre, è stato dimostrato che il Deep Learning è molto efficace per affrontare un ampio spettro di problemi di sicurezza in rete [17].

Architettura di una Rete Neurale

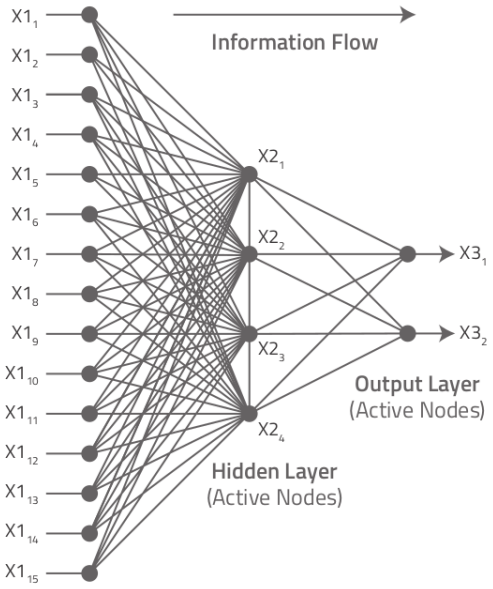


Figure 2.4: Rete Neurale Generica [17].

Come illustrato in figura 2.4, una rete neurale è composta da nodi contenuti nei livelli di input, nascosti e di output. Ogni livello ha un ruolo determinante nella classificazione:

- I nodi nel **Livello di Input** sono passivi. Essi ricevono il valore di un attributo per un particolare campione, i valori ricevuti vengono passati a tutti i nodi del primo livello nascosto per essere elaborati. Il livello di input deve contenere un nodo per ogni feature nel set. Per esempio, se le feature sono 10 nel primo livello avremo 10 nodi.

- I **Livelli Nascosti** sono composti da nodi rappresentati in figura 2.5, tali nodi eseguono il lavoro più importante nel processo di apprendimento.

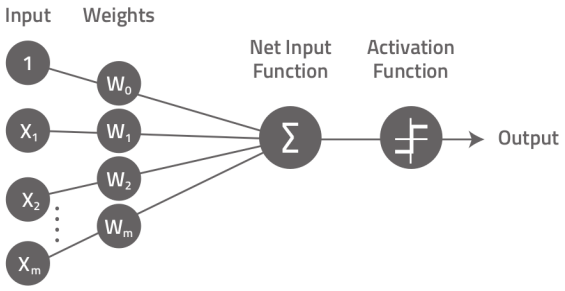


Figure 2.5: Nodo #1 nel Livello nascosto #1. [17]

Si procede come segue [17]:

1. Ogni nodo del livello nascosto riceve i valori di tutti gli attributi del primo campione

2. Ogni attributo viene moltiplicato per il peso corrispondente (es l'attributo nell'input x1 è moltiplicato per il peso w1), se il valore di w è maggiore di 1 il contributo delle feature nella classificazione sarà maggiore, se il peso è minore di 1 il contributo diminuisce. Il peso w viene stabilito ad ogni *epochs* di training, minimizzando la funzione di costo e massimizzando l'accuratezza.

3. I prodotti degli input x e w vengono sommati e passati alla funzione di attivazione. La funzione di attivazione esegue una particolare computazione specificata per quel livello. E' possibile scegliere tra un'ampia gamma di funzioni di attivazione in base alla natura del problema.

4. Viene applicata la funzione di attivazione, il risultato è un valore numerico che riflette gli effetti delle combinazioni peso e valore degli attributi. Processando tutte le combinazioni e passando il risultato a un ulteriore livello nascosto le reti neurali sono in grado di determinare passo dopo passo quale combinazione di features e pesi sono in grado di predire la classe di un campione.

Nel livello nascosto successivo ogni nodo riceve l'output da ogni nodo nel livello nascosto 1. Nuovamente, ognuno di questi valori viene moltiplicato per un peso, i prodotti vengono sommati, i risultati inviati ad una funzione di attivazione per produrre un nuovo output per il livello successivo dove il processo si ripete.

- Il **livello di Output** è il livello finale di una Rete Neurale. Ogni nodo di tale livello rappresenta una possibile classe di classificazione, ognuno di questi nodi incorpora una funzione di attivazione. Per esempio, la funzione di attivazione *softmax* determina un insieme di probabilità di classificazione. Il nodo con una probabilità più alta, determina la classe a cui il campione in input verrà assegnato. Un valore maggiore indica una probabilità più alta di essere classificato in tale classe.

  Il training viene ripetuto più volte, una funzione confronta le classi predette con le etichette presenti nel dataset e determina come modificare i pesi nei livelli nascosti per produrre risultati più accurati. Il processo viene ripetuto quante volte è richiesto, l'iperparmetro *epochs* specifica quante volte deve essere ripetuto il processo di training. Terminata la fase di allenamento il modello può essere valutato sul set di validation e di test.

L'elaborazione nelle Reti Neurali è estremamente granulare, passando da segnali di basso livello a decisioni complesse attraverso una sequenza ordinata di calcoli su più livelli gerarchici. Ogni livello nascosto ha una funzione di attivazione specifica che si occupa di un compito specifico nella classificazione. Per esempio, nella classificazione delle immagini un livello potrebbe riconoscere le forme, un altro associare a una forma una categoria, come un viso.

Maggiore è il numero di livelli, maggiore è la capacità della rete neurale di risolvere problemi complessi. La capacità di risolvere problemi deve essere calibrata in fase di allenamento, un eccesso genera overfitting mentre la carenza genera underfitting. Regolando la densità dei nodi e dei livelli è possibile regolare la capacità. Come negli Alberi di Decisione tramite pruning vengono rimossi i rami non necessari, così nelle reti neurali

è importante scegliere il numero di neuroni e prevenire l'overfitting [17].

Le Reti Neurali sono estremamente flessibili, possono essere utilizzate per risolvere un'ampia gamma di problemi in modi diversi. Rispetto ad altri algoritmi, come le Random Forest, possono avere milioni di parametri per definire il modello.

I modelli descritti in questo capitolo (cap 2), hanno caratteristiche differenti. Pertanto, per valutare quale modello scegliere è opportuno testare i modelli costruiti su nuovi dati, dove il modello non è allenato. In questo modo è possibile valutare overfitting, underfitting e le performance ottenute.

3 Dataset IOT per NIDS

Nel corso del tempo i ricercatori hanno raccolto flussi di rete in dataset. I dati raccolti ci consentono di costruire modelli di classificazione per rilevare intrusioni nelle reti. Ottenere dataset contenenti traffico estratto da reti reali è complesso per ragioni di sicurezza e di privacy. Pertanto, i ricercatori hanno progettato reti in grado di generare traffico sintetico, costituito da flussi di rete etichettati, simulati in laboratorio.

3.1 Analisi Dataset Disponibili

I pacchetti di rete vengono raccolti in unità native (pcap), tramite le modalità spiegate nel capitolo 1.1. Un insieme di features viene estratta dai *pcap files*, utilizzando strumenti appositi, in questo modo vengono formati i flussi di dati. Il risultato è un dataset di flussi etichettati, dove è possibile distinguere gli attacchi dal traffico benigno [13].

Esistono due tipologie di dataset: reali e sintetici. Nei primi i dati vengono raccolti all'interno di reti utilizzate tutti i giorni da utenti, gli attacchi sono reali. Nei secondi vengono costruite delle reti in laboratorio dove vengono simulati degli attacchi. In entrambe le tipologie di dataset i pacchetti vengono raccolti in flussi che vengono etichettati indicando il tipo di attacco.

I dati raccolti hanno permesso di costruire molti dataset non uniformi, ogni dataset ha un insieme di feature diverse (Sezione 3.1.1), tali dataset, poi, sono stati convertiti in dataset con un insieme di feature standard (Sezione 3.1.2).

3.1.1 Dataset Originali

Sono disponibili diversi dataset, resi disponibili pubblicamente per costruire modelli di classificazione, in questa sezione vengono discussi quattro NIDS dataset rilasciati negli ultimi anni, in grado di rappresentare i moderni attacchi nelle reti: [13]

- **ToN-IoT** è un dataset eterogeneo, costituito da flussi di rete di dispositivi IoT, log di sistemi operativi e traffico di rete. Viene considerata solo la parte di flussi di rete. Il dataset è stato raccolto da una rete reale e di grande scala progettata dal *Cyber Range and IoT Labs, the School of Engineering and Information technology (SEIT), UNSW Canberra @ the Australian Defence Force Academy (ADFA)*. L'ambiente di test è formato da più macchine virtuali e host Window e Linux. Il dataset contiene

un grande numero di attacchi. Tramite Bro-IDS sono state estratte 44 features. Il dataset è costituito da 796,380 (3.56%) flussi benigni e 21,542,641 (96.44%) attacchi, in totale 22,339,021 flussi.

Alcune features estratte sono le seguenti: `ts` (Timestamp of connection between flow identifiers), `src-ip` (Source IP addresses which originate endpoints' IP addresses), `src-port` (Source ports which Originate endpoint's TCP/UDP ports), `dst-ip` (Destination IP addresses which respond to endpoint's IP), `dst-port` (Destination ports which respond to endpoint's TCP/UDP ports), `proto` (Transport layer protocols of flow connections), `service` (Dynamically detected protocols, such as DNS, HTTP and SSL), `duration` (The time of the packet connections)

- **CSE-CIC-IDS2018** ottenuto dalla collaborazione di Communications Security Establishment (CSE) & Canadian Institute for Cybersecurity (CIC) nel 2018. La rete è formata da 5 dipartimenti e una stanza server. I pacchetti benigni sono stati generati da utenti umani usando la rete. Gli attacchi sono stati eseguiti da molte macchine esterne alla rete. Tramite CIC-FlowMeter sono state estratte 75 features. L'intero dataset è formato da 13,484,708 (83.07%) flussi benigni e 2,748,235 (16.93%) attacchi, in totale 16,232,943 flussi.

  Alcune delle features estratte sono le seguenti: `fl-dur` (Total packets in the forward direction), `tot-fw-pk` (Total packets in the backward direction), `tot-bw-pk`(Total size of packet in forward direction), `tot-l-fw-pkt` (Maximum size of packet in forward direction), `fw-pkt-l-max` (Minimum size of packet in forward direction), `fw-pkt-l-min` (Average size of packet in forward direction), `fw-pkt-l-avg` (Standard deviation size of packet in forward direction).

- **UNSW-NB15** Questo dataset è stato rilasciato dal Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) nel 2015, è ampiamente utilizzato. Contiene sia traffico reale che sintetico. Sono stati estratti 100 GB di pacap files e tramite un tool apposito sono state estratte 49 features. Il dataset contiene 2,218,761 (87.35%) flussi benigni e 321,283 (12.65%) di attacchi, in totale 2,540,044 flussi.

  Alcune delle feature estratte sono le seguenti: `id, dur, proto, service, state, spkts, dpkts`

- **BoT-IoT** Il Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) nel 2018 ha progettato un un ambiente di rete costituito da traffico normale e botnet. Sono stati raccolti 69.3 GB di pcap files e tramite Argus Tool sono state estratte 42 features. Il datset contiene 477 (0.01%) flussi benigni and 3,668,045 (99.99%)di attacco, in totale 3,668,522 flussi.

  Alcune delle feature estratte sono le seguenti: `pkSeqID`(Row Identifier), `Stime` (Record start time), `Flgs` (Flow state flags seen in transactions), `flgs-number` (Numerical representation of feature flags), `Proto`(Textual representation of transaction protocols present in network flow), `proto-number`(Numerical representation of feature proto), `Saddr` (Source IP address)

I dataset UNSW-NB15 e CSE-CIC-IDS2018 rappresentano una situazione reale, molto traffico benigno e pochi attacchi, viceversa BoT-IoT e ToN-IoT sono formati da molti attacchi e poco traffico benigno. Alcune feature presenti nei dataset sono state calcolate statisticamente, partendo dalle feature già presenti. [13]

Inoltre, sono disponibili molti altri dataset, come KDD-99, N-BaIoT e IoTID20. Tutti i dataset citati sono formati da flussi prelevati da reti differenti ed ognuno ha feature differenti.

Come mostrato sopra le feature dei vari dataset sono diverse, alcune rappresentano lo stesso concetto, ma sono espresse in maniera diversa. Pertanto, è complesso far corrispondere le feature di ogni singolo dataset.

L' assenza di compatibilità fra i veri dataset disponibili ha portato a sviluppare un insieme di feature standard per i Network Intrusion Detection System (NIDS) dataset.

3.1.2 Dataset Standard Feature Set

I dataset introdotti hanno set di feature differenti gli uni dagli altri, parte delle feature sono comuni a più dataset. Al fine di valutare e confrontare i modelli sui diversi dataset è necessario utilizzare un insieme di feature standard. Le principali problematiche dovute all'assenza di un set di feature standard sono [13]:

- L'estrazione di alcune caratteristiche dal traffico di rete è complessa, alcune delle quali sono irrilevanti a causa della mancanza di eventi di sicurezza
- Capacità limitata di valutare la generalizzazione di un modello di Machine Learning su dataset distinti
- Assenza di un dataset universale contenente flussi di dati raccolti in diversi ambienti di rete.

Considerate le problematiche dovute all'assenza di standardizzazione, è stato scelto un insieme di feature significative in grado di costruire modelli con buone performance, il set di feature viene proposto nel paper [12].

I flussi di rete possono essere rappresentati in diversi modi, lo standard de-facto è NetFlow, sviluppato e proposto da Cisco nel 1996. Un flusso di rete è una sequenza di pacchetti, fra due *endpoint* condividendo alcuni attributi come l'IP sorgente e destinazione. Raccogliere il traffico in flussi consente di ridurre lo spazio occupato. Inoltre, i flussi contengono informazioni riguardo agli eventi di sicurezza, essenziali per analizzare la rete in caso di attacco.

La maggior parte dei device di rete, come router e switches, sono in grado di estrarre i flussi in formato NetFlow. Inoltre, tale standard è ampiamente utilizzato in ambito aziendale per esportare e analizzare informazioni di rete. Pertanto, utilizzare le feature

NetFlow per standardizzare i dataset NIDS riduce le risorse necessarie per raccogliere i dati riducendo la complessità. [8]

Le fatures NetFlow Versione 9 sono flow-based, cioè estratte dall'header del pacchetto, non dipendono dal play-load del pacchetto, spesso criptato per mantenere la confidenzialità delle informazioni trasmesse. Tutte le feature sono di tipo numerico, ciò facilita la costruzione di modelli di Machine Learning.

Il tool nProbe viene utilizzato per estrarre le feature NetFlow dai file pcap disponibili pubblicamente, come mostrato nella figura 3.1. L'output è un file CSV, ogni feature è separata da una virgola. Vengono aggiunte due etichette: L̈abelë Ättack̈: La prima indica con un valore binario 0 (Benigno) e 1 (Attacco). La seconda la tipologia di attacco o B̈enignse si tratta di traffico non etichettato come malevolo. [8]

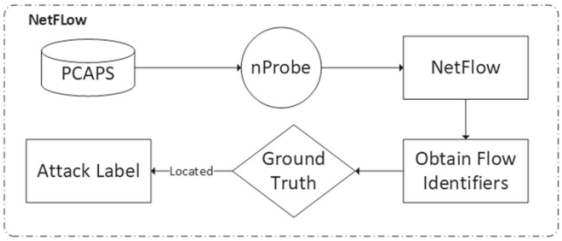


Figure 3.1: Estrazione delle feature da PCAP file [8]

La scelta delle feature altera significativamente le performance del NIDS, infatti è necessario estrarre le feature più significative che consentono di sviluppare modelli di classificazione con buone performance.

I dataset presentati nella sezione precedente 3.1.1, dataset in formato generale sono stati convertiti in dataset con standard NetFlow versione 9 ottenendo: dataset con 12 features NetFlow con il suffisso v́1disponibili nel paper [12] e dataset con 43 features NetFlow con il suffisso v́2disponibili nel paper [13].

Alcune delle feature dei dataset NetFlow V2 sono le seguenti:

Feature	**Descrizione**
IPV4-SRC-ADDR	IPv4 source address
IPV4-DST-ADDR	IPv4 destination address
L4-SRC-PORT IPv4	source port number
L4-DST-PORT IPv4	destination port number
PROTOCOL	IP protocol identifier byte
L7-PROTO	Layer 7 protocol (numeric)
IN-BYTES	Incoming number of bytes
OUT-BYTES	Outgoing number of bytes
IN-PKTS	Incoming number of packets
OUT-PKTS	Outgoing number of packets
FLOW-DURATION-MILLISECONDS	Flow duration in milliseconds
DURATION-IN	Client to Server stream duration (msec)
DURATION-OUT	Client to Server stream duration (msec)
LONGEST-FLOW-PKT	Longest packet (bytes) of the flow

Table 3.1: Feature NetFlow V2 Datasets [9]

I dataset considerati nelle successive analisi sono i seguenti [13]:

3.1.3 NF-ToN-IoT-v2

I pcaps file del dataset ToN-IoT, disponibili pubblicamente, vengono utilizzati per generare i record NetFlow. Si ottiene un set di dati di rete IoT basato su NetFlow, denominata NF-ToN-IoT. Il numero totale di flussi di dati è 16,940,496 di cui 10,841,027 (63.99%) sono campioni di attacco e 6,099,469 (36.01%) sono benigni.

L'istogramma 3.2 mostra la distribuzione degli attacchi nel dataset, il numero di attacchi è presente nella tabella 3.2.

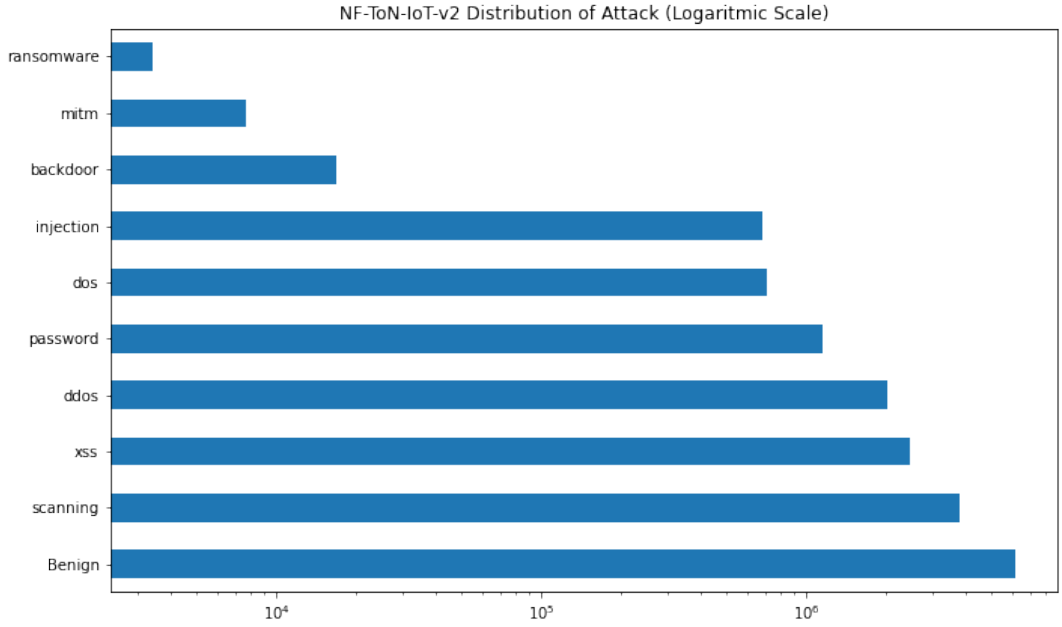


Figure 3.2: Distribuzione Attacchi NF-ToN-IoT-v2

Classe	Count	Descrizione
Benign	6099469	Normal unmalicious flows
Backdoor	16809	A technique that aims to attack remote-access computers by replying to specific constructed client applications
DoS	712609	An attempt to overload a computer system's resources with the aim of preventing access to or availability of its data.
DDoS	2026234	An attempt similar to DoS but has multiple different distributed sources.
Injection	684465	A variety of attacks that supply untrusted inputs that aim to alter the course of execution, with SQL and Code injections two of the main ones.
MITM	7723	Man In The Middle is a method that places an attacker between a victim and host with which the victim is trying to communicate, with the aim of intercepting traffic and communications.
Password	1153323	covers a variety of attacks aimed at retrieving passwords by either brute force or sniffing.
Ransomware	3425	An attack that encrypts the files stored on a host and asks for compensation in exchange for the decryption technique-key.
Scanning	3781419	A group that consists of a variety of techniques that aim to discover information about networks and hosts, and is also known as probing.
XSS	2455020	Cross-site Scripting is a type of injection in which an attacker uses web applications to send malicious scripts to end-users

Table 3.2: Distribuzione NF-ToN-IoT-v2 [8]

3.1.4 NF-CSE-CIC-IDS2018-v2

I pcap files originali del dataset CSE-CIC-IDS2018 dataset sono utilizzati per generare un set di dati basato su NetFlow chiamato NF-CSE-CIC-IDS2018-v2. Il numero totale di flussi è 18,893,708 di cui 2,258,141 (11.95%) sono campioni di attacco e 6,635,567 (88.05%) sono campioni benigni.

L'istogramma 3.3 mostra la distribuzione degli attacchi nel dataset, il numero di attacchi è presente nella tabella 3.3.

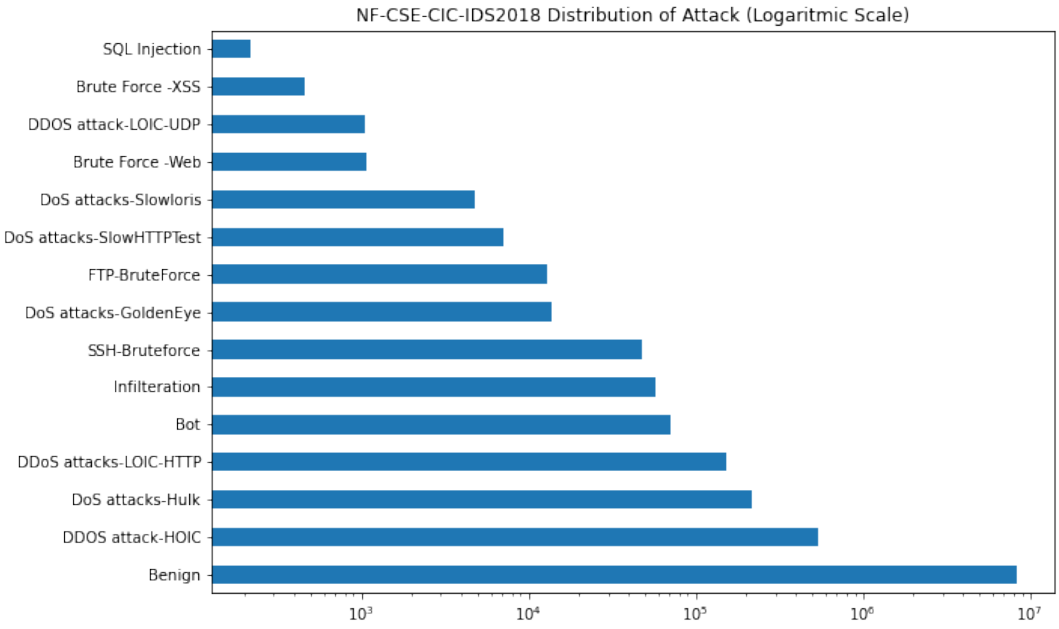


Figure 3.3: Distribuzione Attacchi NF-CSE-CIC-IDS2018-v2

Classe	Count	Descrizione
BruteForce	120912	A technique that aims to obtain usernames and password credentials by accessing a list of predefined possibilities
Bot	143097	An attack that enables an attacker to remotely control several hijacked computers to perform malicious activities.
DoS	483999	An attempt to overload a computer system's resources with the aim of preventing access to or availability of its data.
DDoS	1390270	An attempt similar to DoS but has multiple different distributed sources.
Infiltration	116361	An inside attack that sends a malicious file via an email to exploit an application and is followed by a backdoor that scans the network for other vulnerabilities
Web Attacks	3502	A group that includes SQL injections, command injections and unrestricted file uploads

Table 3.3: Distribuzione NF-CSE-CIC-IDS2018-v2 [8]

Un'alternativa alle feature NetFlow sono le feature estratte dello strumento CIC-FlowMeter.

3.1.5 CIC-ToN-IoT

Un dataset generato partendo dai pcaps files di Ton-IoT. Lo strumento CIC-FlowMeter versione 4 viene utilizzato per estrarre 83 features. Sono presenti 5,351,760 campioni di cui 2,836,524 (53.00%) sono attacchi e 2,515,236 (47.00%) sono campioni benigni [11].

L'istogramma 3.4 mostra la distribuzione degli attacchi nel dataset, il numero di attacchi è presente nella tabella 3.4.

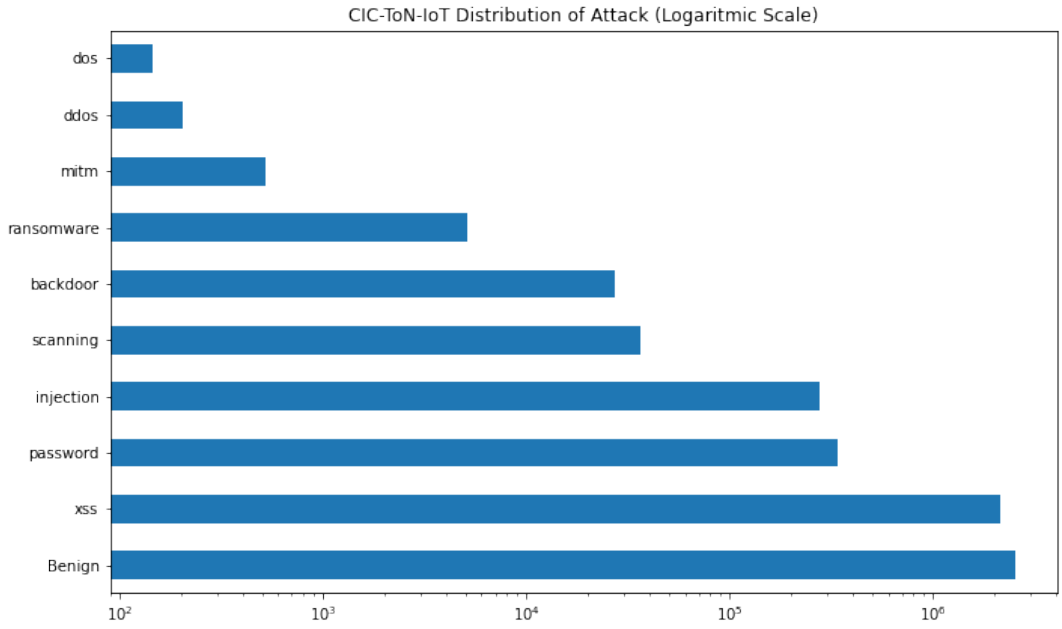


Figure 3.4: Distribuzione Attacchi CIC-ToN-IoT

Classe	Count	Descrizione
Benign	2515236	Normal unmalicious flows
Backdoor	27145	A technique that aims to attack remote-access computers by replying to specific constructed client applications.
DoS	145	An attempt to overload a computer system's resources with the aim of preventing access to or availability of its data.
DDoS	202	An attempt similar to DoS but has multiple different distributed sources.
Injection	277696	A variety of attacks that supply untrusted inputs that aim to alter the course of execution, with SQL and Code injections two of the main ones.
MITM	517	Man In The Middle is a method that places an attacker between a victim and host with which the victim is trying to communicate, with the aim of intercepting traffic and communications.
Password	340208	covers a variety of attacks aimed at retrieving passwords by either brute force or sniffing.
Ransomware	5098	An attack that encrypts the files stored on a host and asks for compensation in exchange for the decryption technique/key.
Scanning	36205	A group that consists of a variety of techniques that aim to discover information about networks and hosts, and is also known as probing.
XSS	2149308	Cross-site Scripting is a type of injection in which an attacker uses web applications to send malicious scripts to end-users.

Table 3.4: Distribuzione CIC-ToN-IoT [11]

Alcune feature dei dataset CIC sono le seguenti [9]: fl-dur (Flow duration), tot-fw-pk (Total packets in the forward direction), tot-bw-pk (Total packets in the backward direction), tot-l-fw-pkt (Total size of packet in forward direction), fw-pkt-l-max (Maximum size of packet in forward direction), fw-pkt-l-min (Minimum size of packet in forward

direction), fw-pkt-l-avg (Average size of packet in forward direction)

I dataset standard, con un insieme di feature predefinito, consente una valutazione affidabile dei modelli di NIDS su più dataset, ambienti di rete e schemi di attacco. Inoltre, tali dataset hanno migliorato significativamente le performance della classificazione multiclasse [8].

I dataset citati nella sezione 3.1.2 sono resi disponibili online dal centro di ricerca dell'Università del Queensland [9].

3.2 Preprocessing Dataset

I dataset citati nelle sezioni precedenti per essere utilizzati nella costruzione di modelli di Machine Learning necessitano di una fase iniziale di pre-elaborazione. I passi necessari sono:

Data Cleaning

E' necessario rimuovere i valori nulli e infiniti, è possibile rimuoverli in diversi modi per esempio eliminando l'intera riga con un valore nullo o infinito, oppure sostituire il valore nullo con un valore medio.

In seguito, è opportuno rimuovere feature non utili per la classificazione. In particolare è necessario rimuovere gli identificatori dei flussi, IP sorgente e destinazione e timestamps. Gli IP vengono rimossi poichè potrebbero portare a *learning bias*, cioè un insieme di ipotesi che l'algoritmo utilizza per prevedere le uscite di dati input che non ha incontrato [2]. In questo caso gli IP sono informazioni che altererebbero il modello, l'utilizzo di un IP specifico non caratterizza un attacco. Inoltre, gli ID dei flussi e i timestamps sono univoci nel dataset, di conseguenza non sono utili per classificare i samples.

Inoltre, è necessario rimuovere tutti i valori categorici presenti nel dataset, nel caso in analisi la tipologia di attacco (es DoS). Utilizzando *LabelEncoder*[7] presente nella libreria di preprocessing di scikit-learn è possibile convertire i valori categorici in valori interi, per esempio se l'attacco è di tipo DoS verrà attribuito il valore numerico intero 3.

Nel caso di Reti Neurali è necessario convertire i valori categorici in *one hot encoding* cioè il valore categorico verrà convertito in un valore binario dove il bit corrispondente alla classe a cui appartiene sarà a 1. Per esempio nel caso siano possibili 3 tipi di attacco diversi, e il flusso nel dataset corrisponde ad un attacco DoS otterremo 001 (il terzo bit rappresenta la classe DoS, gli altri bit le altre classi di attacco). Tale funzionalità viene messa a disposizione dalla libreria *keras.utils*

Standardizzazione

La standardizzazione dei dati è il processo di ridimensionamento di uno o più attributi in modo che abbiano un valore medio pari a 0 e una deviazione standard pari a 1. [3]. Questo processo è necessario quando la distribuzione di ogni feature è molto variabile, questo influenzerebbe la classificazione. La standardizzazione è particolarmente importante nel caso in cui si utilizzano le Reti Neurali.

Per esempio, se media e la deviazione standard della feature1 sono molto più grandi della media e della deviazione standard della feature2. Senza applicare la standardizzazione/normalizzazione la feature1 dominerebbe, il modello non considererà le informazione fornite dalla feature2 [2]. La libreria scikit-learn include *sklearn.preprocessing.StandardScaler* [1 che fornisce questa funzionalità. Un'alternativa alla standardizzazione è la normalizzazione.

La normalizzazione dei dati è il processo di ridimensionamento di uno o più attributi in un intervallo compreso tra 0 e 1. Ciò significa che il valore più grande per ciascun attributo è 1 e il valore più piccolo è 0. La normalizzazione è possibile in scikit-learn tramite *MinMaxScaler* [10]. E' preferibile utilizzare la normalizzazione rispetto alla standardizzazione nel caso in cui è necessario preservare valori bassi di deviazione standard [2]. Lo stesso *Scaler* deve essere applicato sia al training-set e al test-set.

Terminata la fase di preprocessing i dati sono pronti per costruire i modelli di classificazione, valutare le performance e stabilire quale modello utilizzare.

4 Performance Evaluation

Nel capitolo vengono presentate le analisi svolte sui dataset proposti nel capitolo 3. In particolare, vengono descritte le analisi svolte sul dataset NF-TON-IoT-v2, viene scelto un modello di classificazione e utilizzato per ulteriori analisi sul dataset NF-CSE-CIC-IDS-2018-v2. In seguito, i due dataset considerati vengono combinati per ulteriori analisi.

4.1 Metodologia Applicata

I dataset presi in considerazione sono: NF-TON-IoT-v2 (Sezione 3.1.3) e NF-CSE-CIC-IDS-2018-v2 (Sezione 3.1.4). Tali dataset vengono predisposti per la classificazione, tramite le fase di preprocessing e in seguito vengono addestrati diversi modelli di classificazione per valutare il modello migliore.

Le simulazioni vengono svolte utilizzando il seguente ambiente: RAM 16 GB, Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz e linguaggio di programmazione Python (Librerie: scikit-learn e Keras).

La metodologia applicata per effettuare il preprocessing e costruire i modelli di classificazione è la seguente:

1. **Import Dataset**

   I dataset presi in considerazione vengono importati. Il set di dati viene ridotto a metà, mantenendo le proporzioni di attacchi presenti nel dataset. In particolare si ottiene:

   - **TON-IoT-v2** formato da circa 17 milioni (16940496) di campioni si ottiene un dataset di circa 8.5 milioni (8470247) di campioni.
   - **CSE-CIC-IDS2018-v2** formato da circa 19 milioni (18893708) di campioni si ottiene un dataset di circa 9.5 milioni (9446855) di campioni.

   Tale riduzione è resa necessaria dalla memoria limitata della macchina in uso per le simulazioni.

2. **Analisi Descrittiva**

I dati importati vengono analizzati per capire la distribuzione dei vari attacchi e individuare eventuali anomalie nei dati, come valori nulli.

3. **Pulizia dei dati**

   I valori nulli o infiniti identificati precedentemente vengono rimossi. I casi presi in analisi non presentano valori infiniti o NaN. Si rimuovono gli attributi non necessari alla classificazione, per le ragioni spiegate nelle sezione 3.2, in particolare gli IP sorgente e destinazione.

4. **Data Split**

   Il dataset viene suddiviso inizialmente in due parti Y contenenti le etichette che classificano l'attacco (Label e Attack), mentre in X le restanti colonne. Le due parti vengono nuovamente suddivise in tre parti: train set 60%, validation set 20% e test set 20%. Le proporzioni fra le classi di attacco vengono mantenute (viene utilizzato il parametro stratify=Y). I set vengono costruiti in maniera random, i dati provengono da più punti nel dataset (shuffle=True).

5. **Standardizzazione**

   Ogni porzione di X (Train, Validation, Test) di X viene standardizzata secondo le modalità descritte nella sezione 3.2.

6. **Rimozione valori categorici**

   I valori categorici presenti nella porzione Y del dataset vengono rimossi convertendoli in valori numerici utilizzando un apposito LabelEncoder come descritto nella sezione 3.2.

7. **Creazione Modelli**

   Si definisce un modello di classificazione (es. Random forest, Rete Neurale) utilizzando iperparametri standard e si allena il modello sui dati di addestramento.

8. **Validazione e Tuning Iperparametri**

   Il modello definito precedentemente viene testato su dati mai visti prima, il set di validazione. Si valutano le prestazioni ottenute ed eventualmente si modificano gli iperparametri fino ad ottenere performance migliori. Il tuning degli iperparametri viene effettuato automaticamente tramite la libreria *GridSearchCV* [5]. In questo modo è possibile provare tutte le combinazioni di iperaparametri e restituire il modello con risultati migliori.

   In tutti i modelli considerati viene specificato l'iperparametro class-weight, il modello assegna automaticamente i pesi delle classi in modo inversamente proporzionale alle rispettive frequenze [15]. Ciò aiuta, durante la fase di addestramento, a dare pesi maggiori alla classe con meno campioni e pesi minori alla classe con più cam-

pioni.

9. **Valutazione Modello**

   Il modello ottenuto viene utilizzato per predire le classi di un nuovo set di dati, le classi ottenute vengono confrontate con quelle effettive. Si calcolano le metriche di performance.

10. **Scelta Modello Migliore**

    I passi precedenti vengono ripetuti su più tecniche di classificazione, si confrontano le metriche di performance ottenute e si sceglie il modello migliore.

4.2 Metriche di Performance

Le metriche di classificazione raccolte durante le analisi sono: F1 Score, Precision, Recall, Matrice di Confusione, tempo di Training, tempo di predizione di un singolo campione.

- *Precision*. Indica quanti dei casi correttamente predetti in realtà sono positivi

$$\text{Precision} = \frac{TP}{TP + FP}$$

- *Recall*. Indica quanti dei casi positivi il modello ha predetto accuratamente

$$\text{Recall} = \frac{TP}{TP + FN}$$

- *F1 Score*. La combinazione di precision e recall in un'unica metrica calcolando la media armonica fra le metriche

$$\text{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- *Matrice di Confusione*. Riassume il numero di predizioni corrette e scorrette suddiviso per ogni classe. La matrice di confusione mostra i modi in cui il modello di classificazione è confuso quando prevede le classi dei campioni in input, cioè mostra in che modo il modello classifica i dati di test rispetto alla classe di appartenenza.

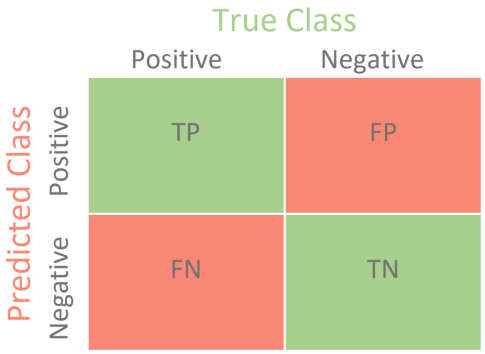


Figure 4.1: Matrice di Confusione
[18]

Per ottenere dei risultati riassuntivi che considerano tutte le classi, si valuta:

- **Macro AVG** La media aritmetica delle metriche calcolate per ogni singola classe. Per esempio, gli F1-Score, calcolati per ogni singola classe, vengono sommati e divisi per il numero di classi. Pertanto, si parla di media non pesata, ogni classe ha lo stesso peso nel risultato finale.

- **Weighted AVG** Le media aritmetica calcolata considerando il numero di campioni per ogni classe, tale media viene pesata in base alla rilevanza di ogni risultato ottenuto. Per esempio, gli F1-Score di ogni classe vengono moltiplicati per il numero di campioni di tale classe e divisi per il numero totale di campioni.

Inoltre si considerano il numero di campioni predetti come traffico benigno:

- **Correctly Classified as Benign** samples etichettati come benigni e classificati come traffico benigno (True Positive)

- **Wrongly Classified as Benign** samples etichettati come attacco ma predetti come traffico benigno (False Positive). Questa metrica consente di valutare la gravità dell'errore commesso, classificare un campione maligno come benigno è più grave di classificarlo come attacco sbagliando la classe.

- **Percentage False Benign** esprime in percentuale quanti dei campioni classificati come benigni non sono effettivamente benigni. Si calcola come rapporto fra i campioni maligni classificati come benigni (*Wrongly Classified as Benign*) e tutti i campioni classificati come benigni (la somma fra *Wrongly Classified as Benign* e *Correctly Classified as Benign*)

Al fine i valutare i tempi di ogni modello si rilevano i seguenti tempi:

- **Model Fitting Time (s)** Tempo necessario per allenare il modello sul set di training.

- **Prediction Time (microsec)** Tempo necessario al modello allenato per predire la classe di un singolo campione presente nel set di test.

Tali metriche vengono utilizzate per valutare i vari modelli e scegliere quale modello è meglio utilizzare per rilevare e classificare gli attacchi che potrebbero presentarsi.

4.3 Confronto Modelli di Classificazione

I modelli presentati nel capitolo 2 vengono applicati ai dataset descritti nel capitolo 3. Le metriche di performance ottenute permettono di valutare quale modello di classificazione supervisionata è più efficace nel rilevare gli attacchi. Per valutare i modelli vengono presi in analisi due dataset molto diversi, NF-TON-IoT-v2 e NF-CSE-CIC-IDS-2018-v2, il primo con più attacchi il secondo con più traffico benigno.

4.3.1 NF-TON-IoT-v2

Il dataset NF-TON-IoT-v2, descritto nella sezione 3.1.3, viene utilizzato come riferimento per scegliere il modello di classificazione migliore. Tale dataset contiene più traffico maligno rispetto al benigno, il 36% del traffico è benigno. La distribuzione dei samples per classe non è uniforme, vi sono classi con molti campioni e classi con pochi campioni. Viene scelto tale dataset per valutare quale modello classifica meglio le diverse tipologie di attacco.

La distribuzione degli attacchi nel dataset, ridotto a 8.5 milioni di campioni mantenendo la proporzione di attacchi, è la seguente:

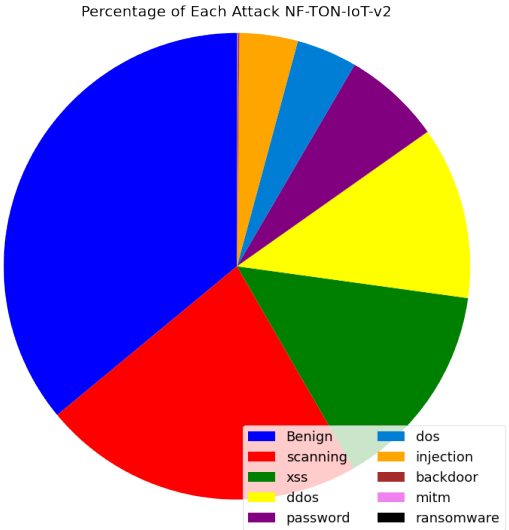


Figure 4.2: Distribution Pie Chart NF-TON-IoT-v2

Il traffico benigno è il 36%, i campioni restanti sono traffico malevolo.

I modelli presi in analisi per stabilire quale tecnica di Machine Learning utilizzare sono:

- Decision Tree
- Random Forest
- Extra Trees
- Boosted Decision Tree - AdaBoost
- Reti Neurali

Per ogni modello viene mostrato il report di classificazione, non si riportano le matrici di confusione per sintesi.

Alberi Di Decisione

La prima tecnica di Machine Learning presa in considerazione è l'utilizzo degli Alberi di Decisione, descritti nella sezione 2.1.

Dopo aver effettuato il tuning degli iperparametri, si ottiene il seguente modello:

```
DecisionTreeClassifier(criterion= 'entropy', class-weight="balanced")
```

Per ulteriori dettagli riguardo agli iperparametri utilizzati fare riferimento alla Sezione 2.4.

Si ottengono i seguenti risultati sul test set:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9932	0.9935	0.9934	609947
backdoor	0.9976	0.9964	0.9970	1681
ddos	0.9850	0.9858	0.9854	202624
dos	0.8893	0.8868	0.8881	71261
injection	0.8937	0.8941	0.8939	68446
mitm	0.5754	0.5389	0.5565	772
password	0.9649	0.9653	0.9651	115333
ransomware	0.9736	0.9708	0.9722	342
scanning	0.9962	0.9961	0.9961	378142
xss	0.9434	0.9431	0.9433	245502
accuracy			0.9752	1694050
macro avg	0.9212	0.9171	0.9191	1694050
weighted avg	0.9752	0.9752	0.9752	1694050

Table 4.1: NF-TON-IoT-v2 Decision Tree Classification Report.

Dal report di classificazione è possibile notare che i campioni vengono classificati con buoni risultati, la classe di attacco MITM (Man In The Middle) ottiene risultati nettamente minori rispetto alle altre classi. I valori di precision e recall sono simili fra loro.

Random Forest

Un altro modello di classificazione promettente sono le Random Forest, descritte nella sezione 2.2.1, una combinazione di Decision Tree che cooperano al fine di ottenere classificazioni migliori.

Dopo aver effettuato il tuning degli iperparametri, al fine di massimizzare l'F1-Score, si definisce il seguente modello:

```
RandomForestClassifier(n-estimators=100, criterion= entropy,
random_state=None, n-jobs=-1, class-weight="balanced")
```

Per ulteriori dettagli riguardo agli iperparametri utilizzati fare riferimento alla Sezione 2.4.

Il modello allenato produce i seguenti risultati sul set di test:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9970	0.9938	0.9954	609947
backdoor	0.9994	0.9952	0.9973	1681
ddos	0.9925	0.9872	0.9898	202624
dos	0.9269	0.9081	0.9174	71261
injection	0.9576	0.8918	0.9235	68446
mitm	0.9420	0.3368	0.4962	772
password	0.9746	0.9680	0.9713	115333
ransomware	0.9706	0.9649	0.9677	342
scanning	0.9971	0.9960	0.9966	378142
xss	0.9404	0.9821	0.9608	245502
accuracy			0.9820	1694050
macro avg	0.9698	0.9024	0.9216	1694050
weighted avg	0.9822	0.9820	0.9819	1694050

Table 4.2: NF-TON-IoT-v2 Random Forest Classification Report.

Le performance migliorano rispetto all'utilizzo degli Alberi Decisionali e ottenendo un F1-Score leggermente migliore, sia nel caso aritmetico che pesato. La classificazione di attacchi Dos, Injection e XSS migliora. La classificazione di attacchi MITM peggiora in questo caso.

Extra Trees

Gli Extra Trees, descritti nella sezione 2.2.2, combinano più alberi di decisione costruiti utilizzando soglie random, non cercano il *threshold (t)* migliore come le Random Forest.

Il modello utilizzato è il seguente:

```
ExtraTreesClassifier(n_estimators=100, criterion='entropy',
random_state=None, n_jobs=-1,
class_weight="balanced")
```

Per ulteriori dettagli riguardo agli iperparametri utilizzati fare riferimento alla Sezione 2.4

I risultati ottenuti sono i seguenti:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9968	0.9932	0.9950	609947
backdoor	0.9994	0.9958	0.9976	1681
ddos	0.9916	0.9870	0.9893	202624
dos	0.9254	0.9038	0.9145	71261
injection	0.9505	0.8890	0.9187	68446
mitm	0.7550	0.3394	0.4683	772
password	0.9727	0.9670	0.9698	115333
ransomware	0.9705	0.9620	0.9662	342
scanning	0.9959	0.9961	0.9960	378142
xss	0.9402	0.9797	0.9595	245502
accuracy			0.9811	1694050
macro avg	0.9498	0.9013	0.9175	1694050
weighted avg	0.9812	0.9811	0.9810	1694050

Table 4.3: NF-TON-IoT-v2 Extra Trees Classification Report.

Le performance sono molto simili alle Random Forest, ma i tempi di costruzione del modello sono maggiori (come mostrato nella tabella 4.13).

Boosted Decison Tree - AdaBoost

AdaBoost una tecnica di *Boosting Ensemble Learning*, descritta nella sezione 2.3, allena sequenzialmente più alberi di decisione.

Per ulteriori dettagli riguardo agli iperparametri utilizzati fare riferimento alla Sezione 2.4.

Per definire il modello si riutilizzano i parametri definiti per l'albero singolo, e lo stesso numero di stimatori utilizzato per gli altri metodi di Ensemble Learning:

```
AdaBoostClassifier(
DecisionTreeClassifier(criterion= 'entropy', class_weight="balanced"),
n_estimators=100, algorithm="SAMME.R", learning_rate=0.5)
```

Il test del modello costruito produce questi risultati:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9933	0.9934	0.9934	609947
backdoor	0.9941	0.9964	0.9952	1681
ddos	0.9851	0.9857	0.9854	202624
dos	0.8880	0.8867	0.8874	71261
injection	0.8936	0.8931	0.8933	68446
mitm	0.5917	0.5389	0.5641	772
password	0.9651	0.9656	0.9653	115333
ransomware	0.9737	0.9737	0.9737	342
scanning	0.9961	0.9961	0.9961	378142
xss	0.9434	0.9433	0.9434	245502
accuracy			0.9752	1694050
macro avg	0.9224	0.9173	0.9197	1694050
weighted avg	0.9752	0.9752	0.9752	1694050

Table 4.4: NF-TON-IoT-v2 Boosted Decision Tree Classification Report.

I risultati ottenuti sono molto simili agli Alberi di Decisione, quindi inferiori rispetto alle Random Forest.

Reti Neurali

Al fine di avere una panoramica completa delle tecniche di classificazione, si considera una semplice Rete Neurale con un solo livello nascosto.

Le Reti Neurali vengono descritte nella sezione 2.5.

La Rete Neurale utilizzata viene definita come segue:

```
#Create the Model
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=(41,)))
model.add(keras.layers.Dense(32, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

#Compile the Model
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=['acc'])

#Fit the model
```

```
model.fit(x_train, y_train, epochs=15, batch_size=512,
          validation_data=(x_validate, y_validate),
         class_weight = class_weights,)
```

Per implementare le Reti Neurali in linguaggio Python è stata utilizzata la libreria Keras. Keras permette di sviluppare e sperimentare velocemente nell'ambito del Deep Learning e Machine Learning. Per definire il modello si forniscono i seguenti comandi [2]:

- `Sequential`

  Specifica che si sta creando un modello sequenziale, il modello Keras più semplice. La Rete Neurale è composta da un insieme di livelli connessi sequenzialmente, l'output di ogni livello è l'input per lo strato successivo.

- `model.add`

  Tale costrutto viene utilizzato per aggiungere un livello alla Rete Neurale. E' necessario specificare come parametro il tipo di livello. Il primo livello è di tipo `Flatten` il suo ruolo è convertire ogni campione in un array 1D, tale livello svolge solo un ruolo di pre-processing. Essendo il primo livello del modello si specifica l' `input_shape`, la struttura delle istanze, in questo caso 41 feature per ogni campione in input. Il secondo livello di tipo nascosto `Dense` con 32 neuroni, utilizza la funzione di attivazione ReLU. Infine, si aggiunge un livello di output di tipo `Dense` con 10 neuroni (Uno per classe), si utilizza la funzione di attivazione Softmax, i cui valori possono essere interpretati come le probabilità che un dato input appartenga a ciascuna classe.

- `model.compile`

  Dopo aver creato il modello è necessario chiamare il metodo `compile()` per specificare la funzione di loss e l'optimizer da utilizzare. Se `loss` assume valore categorical_crossentropy, specifica che abbiamo più classi. Viene specificato l'`optimizer` con cui allenare il modello, il più utilizzato è *Adam*. Il parametro `metrics` è usato per specificare il modo in cui vogliamo misurare le performance durante l'allenamento e la validazione. Nel caso in analisi viene scelta l' accuracy.

- `model.fit`

  Il modello è pronto per essere allenato, per fare ciò è necessario specificare i dati in input (x_train), le lables (y_train), numero di iterazioni (n_ephochs) e la dimensione del batch. Spesso il dataset è molto ampio quindi viene utilizzato il batch_size. Questo suddivide i dati in batch ognuno della dimensione del `batch_size`. Quando un batch viene processato viene eliminato dalla memoria e caricato un nuovo batch. Il parametro facoltativo `validation_data` specifica i dati di validation da utilizzare, ciò è utile per valutare l'overfitting. Il parametro `class_weight` considera i pesi da attibuire ad ogni classe, calcolati precedentemente, e specificati come dizionario. Tale paramentro viene utilizzato per bilanciare la loss function. La funzione di fitting restituisce a ogni iterazione l'accuratezza del modello, sia per il train set che

per il validation set.

Terminati questi passaggi la Rete Neurale è allenata.

Per ulteriori dettagli riguardo alle Reti Neurali fare riferimento alla sezione 2.

Non sono state considerate Reti Neurali più profonde, con più livelli, che potrebbero portare a risultati migliori.

I risultati ottenuti sono i seguenti:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9828	0.9310	0.9562	609947
backdoor	0.9258	0.9875	0.9557	1681
ddos	0.9657	0.9239	0.9443	202624
dos	0.9210	0.8270	0.8715	71261
injection	0.6929	0.7818	0.7347	68446
mitm	0.0133	0.5000	0.0259	772
password	0.8854	0.9015	0.8934	115333
ransomware	0.1186	0.9678	0.2113	342
scanning	0.9529	0.9644	0.9586	378142
xss	0.9297	0.9369	0.9333	245502
accuracy			0.9259	1694050
macro avg	0.7388	0.8722	0.7485	1694050
weighted avg	0.9448	0.9259	0.9346	1694050

Table 4.5: NF-TON-IoT-v2 Neural Network Classification Report.

I risultati peggiori si ottengono su attacchi con il minor numero di campioni, fra cui MITM e Ransomware. Per quanto riguarda le altre tipologie di attacco, confrontate con i modelli precedenti, i risultati sono inferiori. La rete neurale presa in considerazione conduce a risultati peggiori rispetto agli altri modelli analizzati.

Scelta Modello di Classificazione

Le tecniche di Machine Learning utilizzate per classificare gli attacchi del dataset NF-TON-IoT-v2 hanno permesso di ottenere buoni risultati, l'F1-Score è sempre maggiore di 0.93 La tabella riassuntiva che segue, fornisce una visione complessiva sui modelli analizzati:

		Decision Tree	Random Forest	Extra Trees	Boosted DT	Neural Network
Precision	Macro AVG	0,9212	0.9698	0.9498	0.9224	0.7388
	Weighted AVG	0,9752	0.9822	0.9812	0.9752	0.9448
Recall	Macro AVG	0,9171	0.9024	0.9013	0.9173	0.8722
	Weighted AVG	0,9751	0.9820	0.9811	0.9752	0.9259
F1-Score	Macro AVG	0,9191	0.9216	0.9175	0.9197	0.7485
	Weighted AVG	0,9752	0.9819	0.9810	0.9752	0.9346
Percentage False Benign		0,68%	0,30%	0,32%	0,67%	1,72%
Model Fitting Time (s)		89,13	234,9	305,3	89,74	145,44
Prediction Time(μs)		0,21	6,66	6,03	0,47	1,38

Table 4.6: Comparison of ML Model on NF-TON-IoT-v2

Un compromesso fra F1-Score, tempi e *Percentage False Benign* porta a scegliere le Random Forest come modello migliore da utilizzare nelle successive analisi.

Le Random Forest, rispetto ai modelli presi in analisi, con F1-Score simili, hanno tempi di allenamento migliori. La metrica *Percentage False Benign*, la percentuale di campioni erroneamente classificati come benigni rispetto a tutti i campioni classificati come benigni, è minore nel modello scelto. Tale metrica è significativa per costruire un Intrusion Detection System efficace, infatti, è più importante rilevare gli attacchi pur non identificando la classe corretta rispetto a non rilevarli, classificandoli come benigni.

La matrice di confusione, del modello scelto, mostra graficamente i risultati di classificazione, in particolare si nota che in alcuni casi gli attacchi MITM non vengono classificati correttamente.

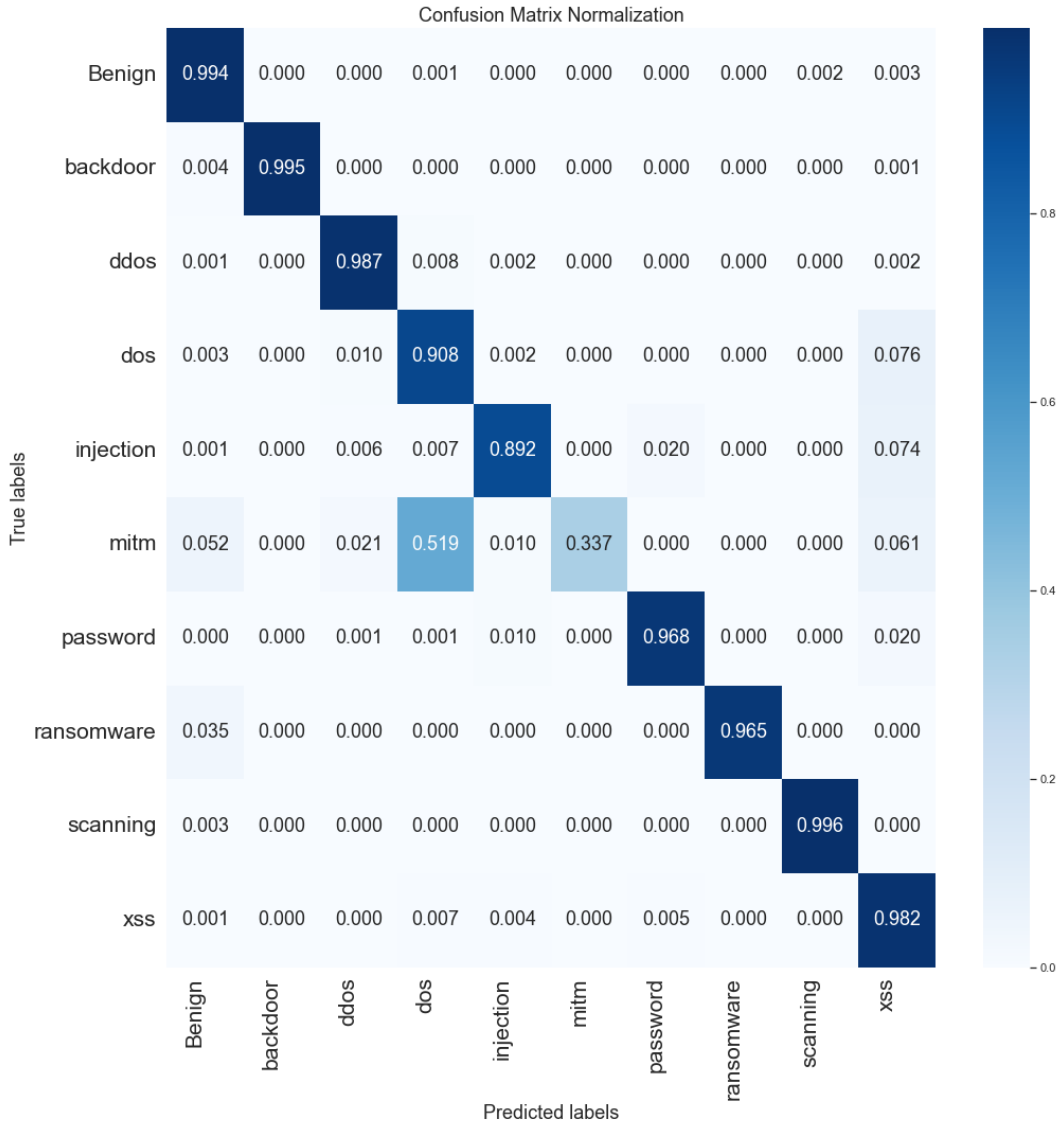


Figure 4.3: Confusion Matrix Random Forest NF-TON-IoT-v2.

Il numero di attacchi MITM presenti nel dataset è molto minore rispetto agli altri attacchi (3862 campioni nell'intero dataset). Inoltre, la classificazione di attacchi MITM in maniera errata potrebbe essere ricondotta all'assenza di pattern simili fra i vari attacchi di questo tipo. Infatti, gli attacchi di tipo MITM presenti nel dataset sono stati simulati utilizzando una macchina virtuale Kali con IP 192.168.1.31-34 tramite il tool Etter-cap lanciando tre scenari di attacco diversi: *ARP Cache Poisoning, ICMP Redirection and Port-Stealing Attacks* [1]. Tali schemi di attacco cercano di carpire i flussi di rete seguendo metodologie diverse, attacchi diversi producono pattern diversi nel dataset.

La categoria MITM include tutte le tipologie di attacco, la costruzione dei set di train, validation e test avviene in maniera randoom, viene mantenuta la propozione di attacchi MITM, ma non delle sottocategorie. Il modello potrebbe essere allenato nel riconoscere una sola categoria specifica, per esempio ARP Cache Poisoning. ertanto, considerando il numero limitato di campioni di attacco MITM e i differenti metodi di attacco utilizzati, i modelli allenati faticano a riconoscere questa tipologia di attacchi.

Le altre classi vengono riconosciute in maniera corretta nella maggior parte dei casi, gli F1-score sono sempre maggiori di 0.91

Interpretazione Modello - SHAP

Al fine di comprendere perchè un campione è stato classificato in una classe, cioè spiegare le scelte del modello, si utilizza SHAP (SHapley Additive exPlanations). L'obiettivo di SHAP è spiegare la predizione di un campione computando il contributo di ogni feature nella predizione. SHAP è un approccio usato nella teoria dei giochi per spiegare l'output di qualsiasi modello di apprendimento automatico. Collega l'allocazione ottimale del credito con la spiegazione locale utilizzando i valori di Shapley[1].

I valori SHAP sono basati su Shapley Values, un concetto della teoria dei giochi. La teoria dei giochi ha bisogno di due elementi: un gioco e alcuni giocatori. Considerando un modello predittivo, il gioco è l'output del modello, mentre i giocatori sono le *features* incluse nel modello. Shapley quantifica il contributo di ogni giocatore nel gioco. Mentre, SHAP quantifica il contributo di ogni feature nella predizione datta dal modello. I valori SHAP esprimono quanto equamente è distribuito il payoutfra le features.

Di seguito vengono illustrati i risultati ottenuti con le Random Forest sul dataset NF-TON-IoT-v2.

[1] https://github.com/slundberg/shap

Nell'istogramma che segue viene mostrata la SHAP Explanation su tutte le classi del dataset. Si considerano 2 milioni di dati con classi distribuite proporzionalmente, tale scelta è resa necessaria della complessità computazionale di SHAP e le risorse a disposizione.

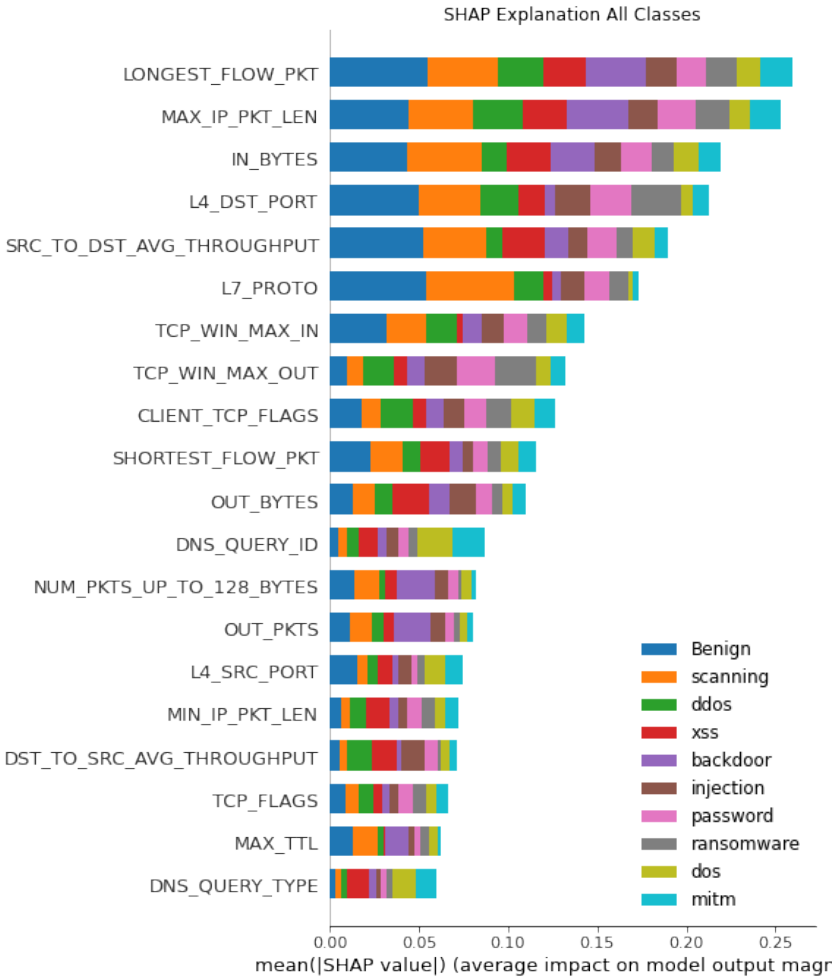


Figure 4.4: SHAP Explanation All Classes NF-TON-IoT-v2 (2 million sample)

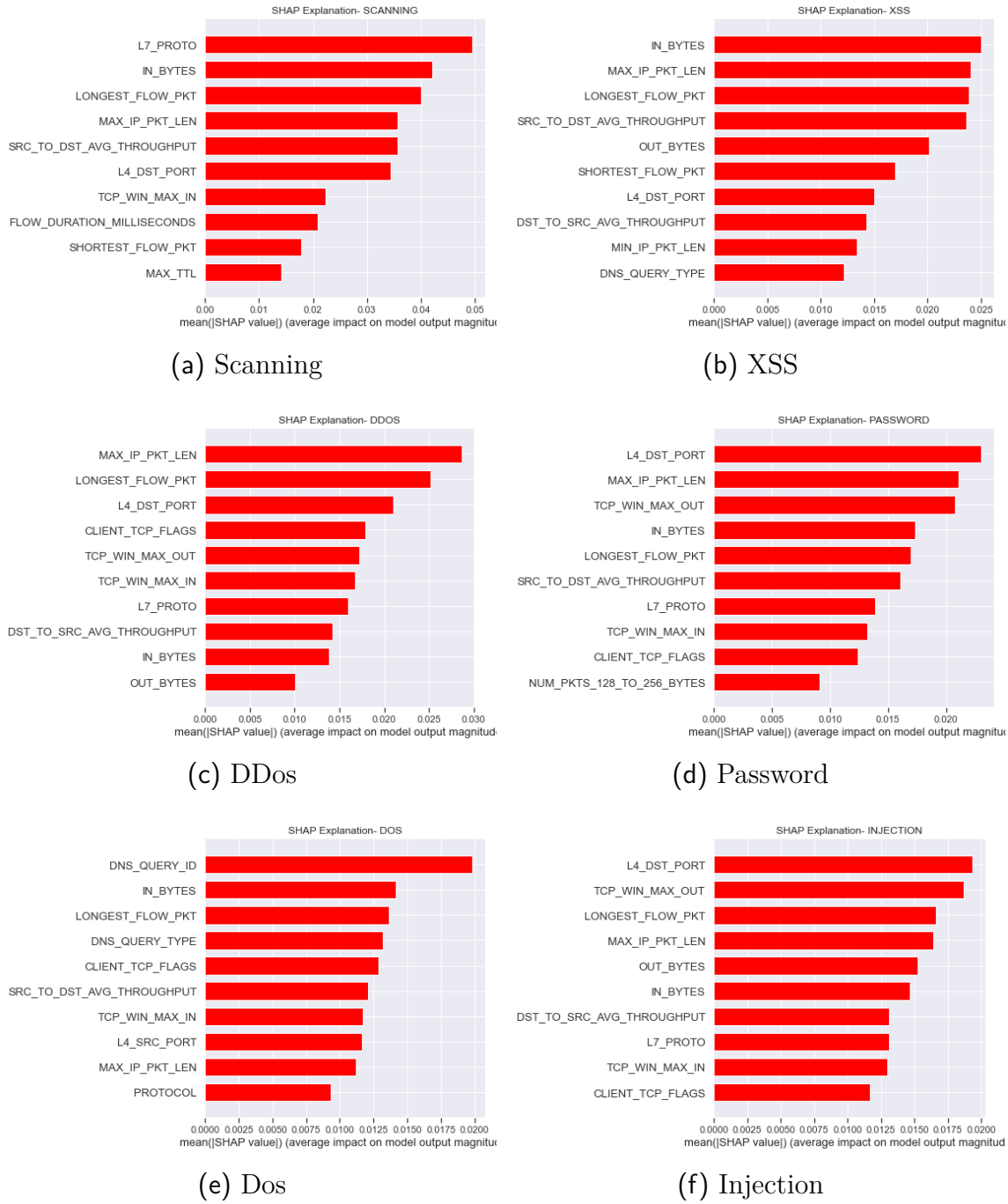
Dal grafico emerge che la feature che influisce maggiormente nella classificazione è *Longest Flow Packet*, cioè la lunghezza (in byte) del pacchetto più grande trasmesso nel flusso. Nella porzione di dataset che segue è possibile vedere alcuni esempi di valori assunti dalle feature.

L4 SRC PORT	L4 DST PORT	L7 PROTO	IN BYTES	IN PKTS	OUT BYTES	LONGEST FLOW PKT	...	Label	Attack
49228	1880	0	1600	40	35741	1286		0	Benign
0	0	0	212	2	0	106		0	Benign
65317	1900	0	165	1	0	165		0	Benign
54023	53	0	108	2	108	54		1	dos
54023	53	0	108	2	108	54		1	dos
59901	53	0	100	2	50	50		1	dos
34689	443	91	84	2	44	44		1	scanning
62873	3389	88	44	1	40	44		1	scanning
41754	3766	0	48	1	0	48		1	scanning

Table 4.7: Sample of NF-TON-IoT-v2 Dataset

Individuare manualmente i pattern che l'algoritmo di ML ha utilizzato per classificare un determinato campione è molto complesso. Pertanto, il contributo di SHAP è importante per capire quali feature sono più importanti nella classificazione di un determinato attacco.

Di seguito viene mostrata la SHAP Explanation per classe, su 5 milioni di dati, considerando le 6 classi con più campioni e le 10 feature più significative.



(a) Scanning (b) XSS

(c) DDos (d) Password

(e) Dos (f) Injection

Figure 4.5: SHAP Explanation Classes NF-TON-IoT-v2

Nell'explanation per classe la feature *Longest Flow Packet* ricopre sempre le prime po-

sizioni. Le feature che hanno influito nella classificazione sono spesso comuni alle varie classi, fra cui *L4_DST_PORT*, cioè la porta di destinazione a livello di Trasporto e *IN_BYTES*, in numero di bytes in ingresso. Le feature non mostrate sono poco significative nel processo di classificazione.

L'utilizzo del metodo SHAP ha permesso di capire meglio le scelte del modello costruito, identificando per ogni classe quale feature ha influito maggiormente per determinare se si tratta di traffico benigno o meno ed eventualmente a quale classe di attacco appartiene.

4.3.2 NF-CSE-CIC-IDS-2018-v2

Nel dataset NF-CSE-CIC-IDS-2018-v2 descritto nella sezione 3.1.4, il traffico benigno è presente in quantità maggiore. Pertanto, si considera tale dataset per confrontare le performance ottenute rispetto al dataset analizzato nella sezione precedente. La distribuzione degli attacchi e del traffico benigno è la seguente:

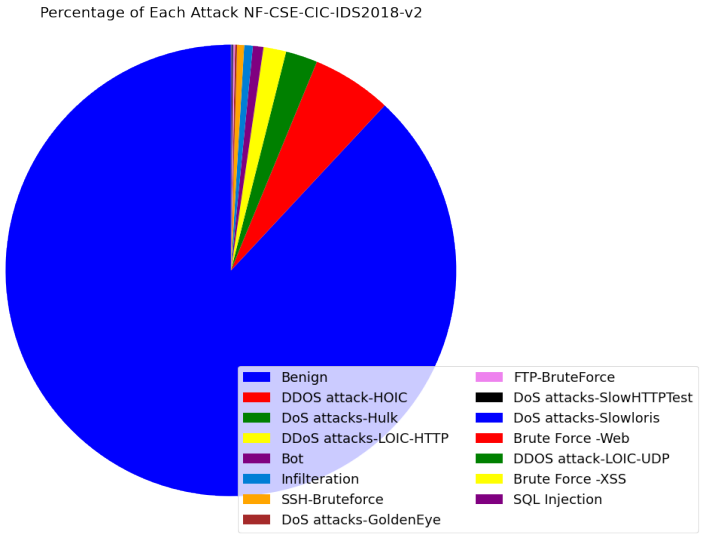


Figure 4.6: Distribution PIE Chart NF-CSE-CIC-IDS-2018-v2

I modelli applicati nella sezione precedente vengono applicati anche a questo dataset, le performance sono migliori rispetto al primo datset come mostrato nella tabella:

	Decision Tree	Random Forest	Extra Trees	Boosted DT	Neural Network
Precision Weighted AVG	0,9919	0,9955	0,9945	0,9919	0,9900
Recall Weighted AVG	0,9921	0,9956	0,9951	0,9921	0,9900
F1-Score Weighted AVG	0,9920	0,9947	0,9943	0,9920	0,9933
Percentage False Benign	0,45%	0,45%	0,45%	0,44%	0,54%
Model Fitting Time (s)	180,43	394,21	353,29	2000	173,8

Table 4.8: Comparison of ML Model on NF-CSE-CIC-IDS-2018-v2

Rispetto al dataset considerato precedentemente, l'F1-Score è simile per ogni modello,

non vi sono tecniche di Machine Learning tali per cui i risultati sono nettamente migliori. In questo caso l'allenamento sequenziale dei Boosteed Decision Tree necessita di tempi particolarmente lunghi, circa 30 min. In questo caso gli Extra Trees ottengono risultati comparabili alle Random Forest. Infatti, i due modelli sono molto simili in alcuni casi gli Extra Trees performano meglio, ma non è possibile stabilirlo a priori [2].

Non si riportano i risultati dettagliati, considerate le analisi della sezione precedente si decide di proseguire focalizzandosi sul modello più promettente, le Random Forest.

Applicando tale modello i risultati sul test suddivisi per classe sono i seguenti:

	Precision	**Recall**	**F1-score**	**Support**
Bot	1.0000	1.0000	1.0000	14310
Brute Force -Web	0.1575	0.1075	0.1278	214
Brute Force -XSS	0.1897	0.1183	0.1457	93
DDOS attack-HOIC	0.9973	0.9984	0.9978	108086
DDOS-attack-LOIC-UDP	0.9952	0.9905	0.9929	211
DdoS-attacks-LOIC-HTTP	0.9999	1.0000	1.0000	30730
DoS-attacks-GoldenEye	1.0000	0.9996	0.9998	2772
DoS attacks-Hulk	1.0000	1.0000	1.0000	43265
DoS-attacks-SlowHTTPTest	1.0000	1.0000	1.0000	1412
DoS attacks-Slowloris	0.9989	0.9979	0.9984	951
FTP-BruteForce	1.0000	1.0000	1.0000	2593
Infilteration	0.9684	0.3525	0.5169	11636
SQL Injection	0.2083	0.1163	0.1493	43
SSH-Bruteforce	1.0000	1.0000	1.0000	9498
accuracy			0.9957	1889371
macro avg	0.8341	0.7787	0.7951	1889371
weighted avg	0.9955	0.9957	0.9947	1889371

Table 4.9: NF-CSE-CIC-IDS-2018-v2 Random Forest Classification Report.

I risultati ottenuti sono migliori rispetto a NF-TON-IoT-v2, il modello è in grado di riconoscere meglio i pattern presenti nel dataset e rilevare gli attacchi. I dataset sono diversi, sia per la distribuzione di attacchi e traffico benigno sia per la tipologia di attacchi presenti nel dataset. Alcuni attacchi sono comuni ad entrambi i dataset come Dos, DDos e injection. Nel dataset NF-CSE-CIC-IDS-2018-v2 gli attacchi sono etichettati in maniera più specifica, nell'altro dataset in maniera più generica. Un attacco rappresenta una famiglia con schemi di attacco diversi e di conseguenza pattern differenti nei flussi di rete. Infatti, gli attacchi DDos e Dos portano a risultati migliori nel dataset in analisi, rispetto al dataset analizzato precedentemente, il modello riesce ad allenarsi su dati più specifici caratteristici di un particolare attacco.

Alcuni classi nel set di test hanno un numero molto esiguo di campioni, infatti i risultati di alcuni attacchi come XSS (93 campioni) e SQL Injection (43 campioni) sono molto bassi, ciò è giustificato dalla scarsità di dati di training. Considerato il numero ridotto di campioni in alcune classi è importante calcolare la media pesata e non pesata. Considerato lo sbilanciamento del set di test le metriche di performance calcolate tramite media aritmetica sono molto minori rispetto al caso precedente. Considerando la media pesata,

i risultati sono migliori, tale metrica considera anche la quantità di dati disponibili.

La matrice di confusione normalizzata, calcolata sul test-set, mostra chiaramente quanto discusso:

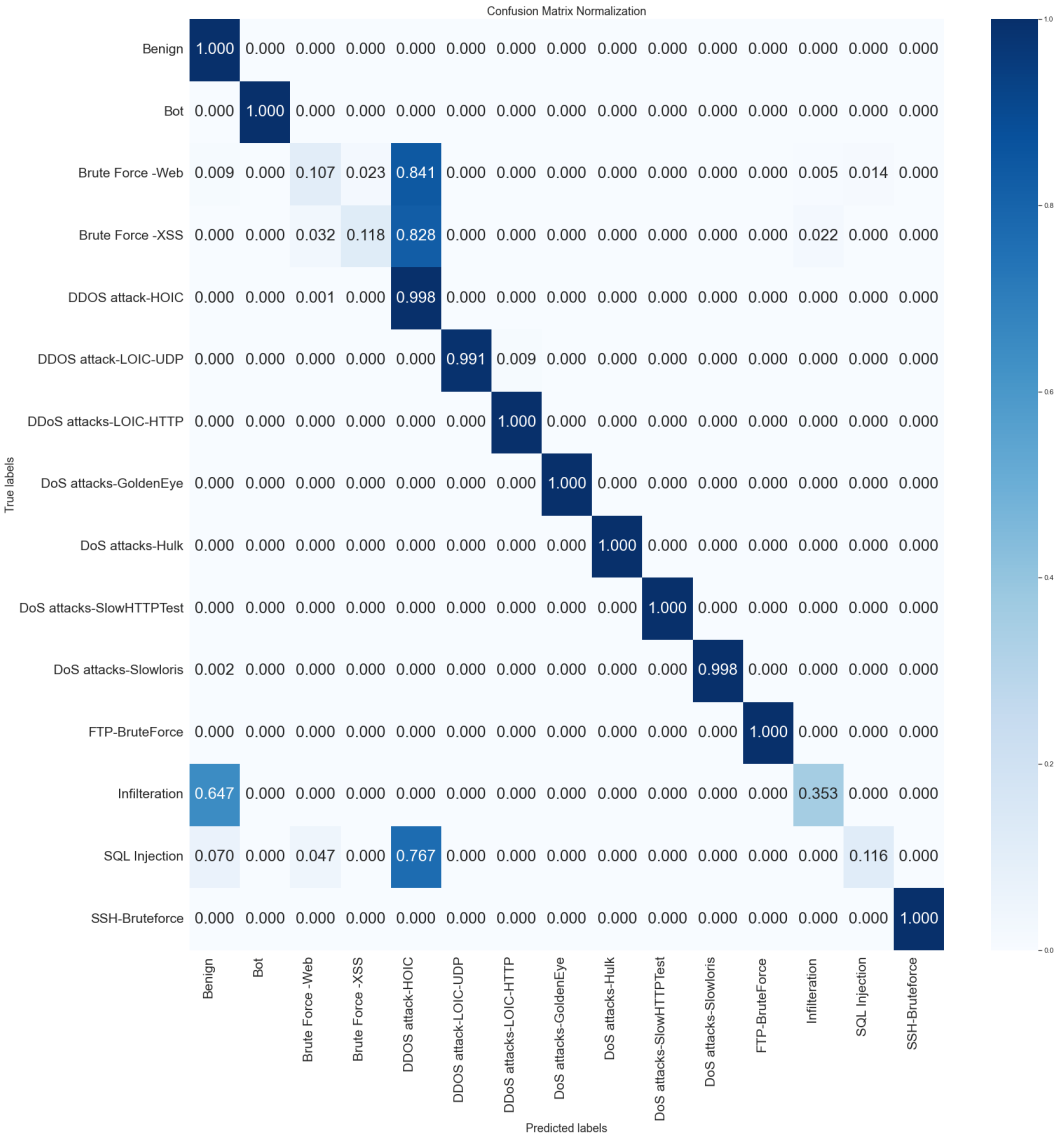


Figure 4.7: Confusion Matrix Random Forest NF-CSE-CIC-IDS-2018-v2.

In particolare, dalla matrice è possibile notare che gli attacchi di infilteration molto spesso vengono classificati come benigni. Per implementare questo scenario di attacco, è stato utilizzato il tool Metasploit [14]. Secondo lo schema di attacco utilizzato, la vittima dopo aver scaricato il file infetto, da Dropbox per le macchine Windows o da una memoria falsh USB infetta, per le macchine Macintosh. L'attaccante esegue l'Nmap e il portscan sull'intera rete della vittima [14]. Pertanto, attacchi simili potrebbero non avere pattern particolari rilevabili dal modello.

Le analisi svolte sui dataset NF-TON-IoT-v2 e NF-CSE-CIC-IDS-2018-v2, hanno permesso di confrontare gli stessi modelli su dataset diversi e valutarne le performance.

4.4 Analisi su combinazione di Dataset

Lo sviluppo di un Intrusion Detection System ottimale, prevede di utilizzare un modello di Machine Learning allenato sul maggior numero di dati provenienti da ambienti diversi con schemi di attacco differenti. In questo modo il sistema dovrebbe essere in grado di riconoscere il maggior numero di pattern già visti in passato e classificare correttamente gli attacchi.

Al fine di valutare il comportamento di più dataset combinati, si cerca di riprodurre tre possibili scenari:

1. **Scenario 1** Training e Test con la combinazione di CSE-CIC-IDS-2018-v2 e TON-IoT-v2

2. **Scenario 2** Training con la combinazione di CSE-CIC-IDS-2018-v2 e TON-IoT-v2 e test con CSE-CIC-IDS-2018-v2

3. **Scenario 3** Training con CSE-CIC-IDS-2018-v2 e test con TON-IoT-v2

Tali scenari cercano di simulare situazioni reali, nei primi due il modello è allenato a riconoscere attacchi di quel dataset. Nell'ultimo caso il modello non è allenato, l'IDS si trova di fronte a dati mai visti.

Il modello scelto precedentemente, le Random Forest, viene applicato ai 3 scenari per valutarne le performance.

Al fine di combinare i dataset è necessario considerare parte di un dataset e parte dell'altro dataset:

- **NF-TON-IoT-v2** formato da circa 4,2 milioni di campioni (17 milioni in totale)

- **CSE-CIC-IDS-2018-v2** formato da circa 4,2 milioni di campioni (19 milioni in totale)

Ottenuto un dataset formato da proporzioni eque dei due dataset si riconducono gli attacchi specifici, come DDOS attack-HOIC, a una classe di attacchi comune ai due dataset, per esempio DDos. Gli attacchi presenti in un solo dataset non vengono alterati.

Tale procedimento viene illustrato nella figura che segue:

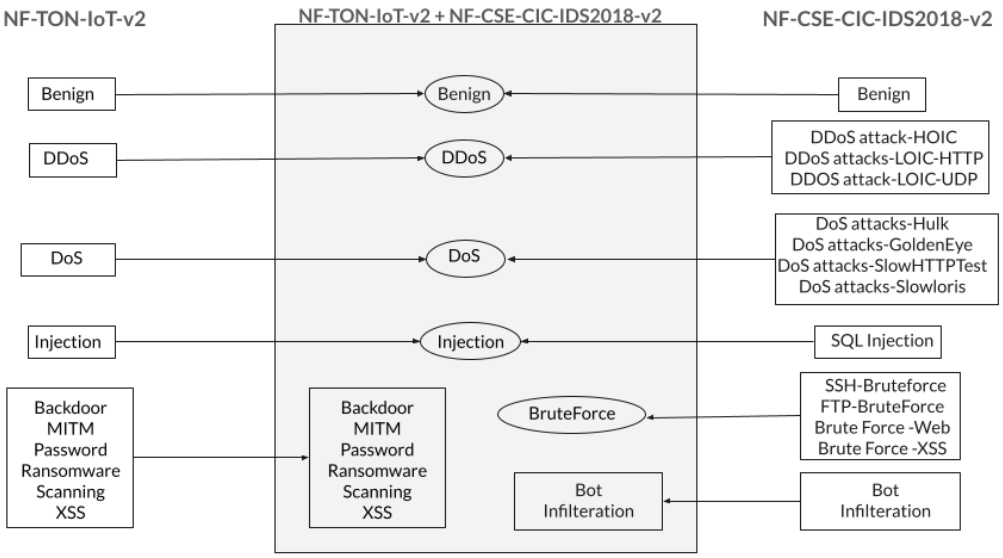


Figure 4.8: Class Combination NF-TON-IoT-v2 e CSE-CIC-IDS-2018-v2

La combinazione dei dataset produce la seguente distribuzione, rappresentata con un grafico a torta:

Figure 4.9: Distribution Pie Chart Combination TON-IoT-v2 e NF-CSE-CIC-IDS-2018-v2.

Il dataset ottenuto viene utilizzato per le successive analisi, in particolare nello Scnario 1 e Scenario 2.

4.4.1 Combinazione Dataset

Nel primo scenario i due dataset combinati (CSE-CIC-IDS-2018-v2 e TON-IoT-v2), formano un unico dataset, che viene suddiviso in train, validation e test. Al dataset ottenuto vengono applicate tutte le operazioni descritte nella sezione 4.1.

Il modello, le Random Forest, viene allenato sul dataset combinato. Il modello costruito viene testato sul set di test e i risultati sono i seguenti:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9954	0.9976	0.9965	1053574
Bot	1.0000	1.0000	1.0000	6440
Infilteration	0.9584	0.3168	0.4762	5236
backdoor	0.9988	1.0000	0.9994	840
brute-force	0.9920	0.9774	0.9847	5579
ddos	0.9942	0.9914	0.9928	163874
dos	0.9513	0.9419	0.9466	57411
injection	0.9541	0.8819	0.9166	34243
mitm	0.9160	0.3109	0.4642	386
password	0.9713	0.9645	0.9679	57666
ransomware	0.9695	0.9298	0.9493	171
scanning	0.9957	0.9952	0.9954	189071
xss	0.9368	0.9799	0.9578	122751
accuracy			0.9878	1697242
macro avg	0.9718	0.8683	0.8960	1697242
weighted avg	0.9878	0.9878	0.9872	1697242

Table 4.10: Scenario 1 Random Forest Classification Report.

Le performance del modello costruito sono buone, l' F1-Score e i tempi sono in linea con i risultati ottenuti applicando il modello ai singoli dataset. Gli errori di classificazione di MITM e Infilterarion, identificati precedentemente, restano anche nella combinazione fra dataset.

Nei casi di attacchi comuni ai dataset, come DDos e Dos, le performance complessive migliorano, rispetto ai dataset considerati singolarmente. Pertanto, la diversità di schemi di attacco per una categoria come Dos, migliora le performance di classificazione complessive.

La matrice di confusione evidenzia quanto descritto sopra:

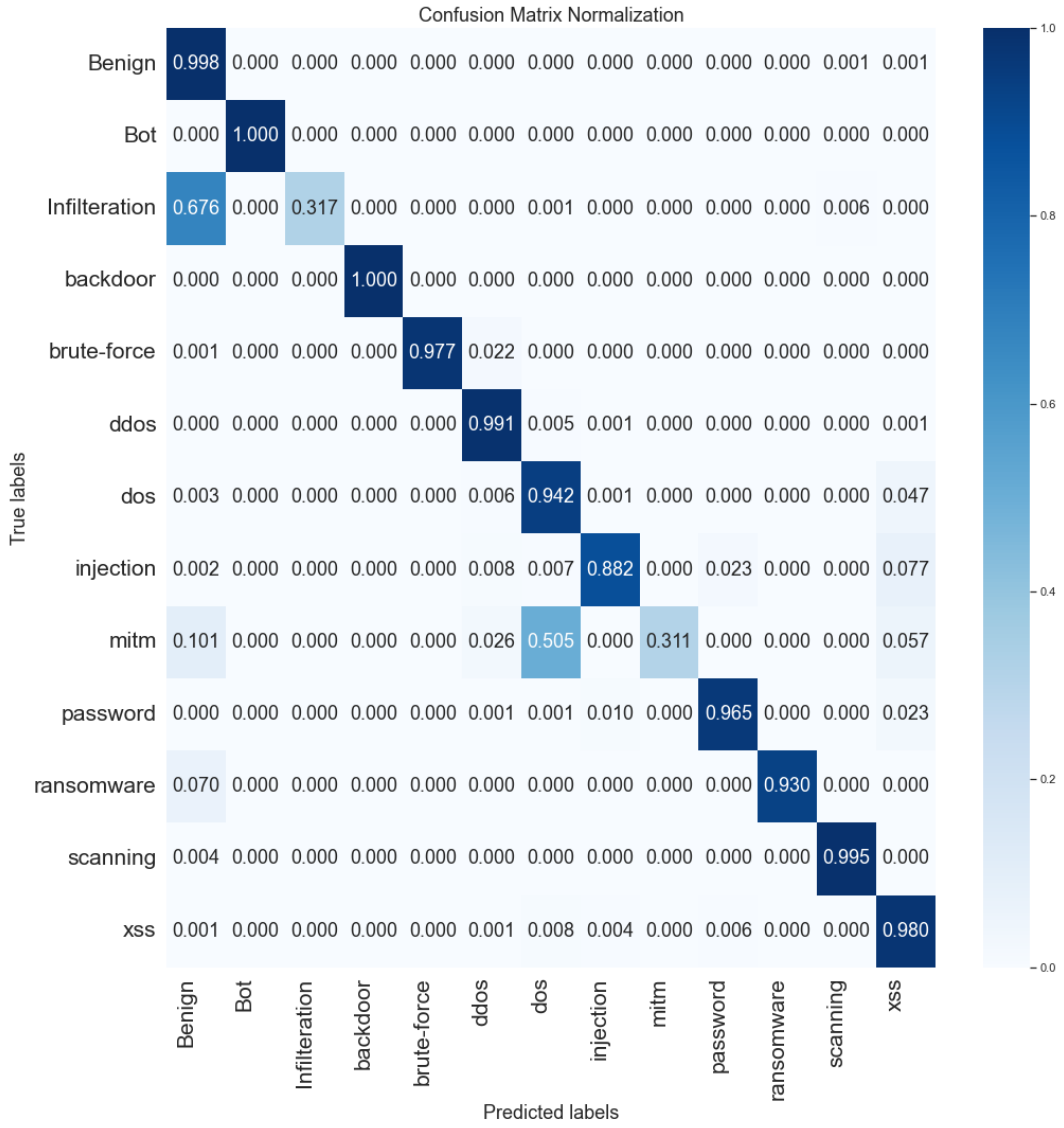


Figure 4.10: Confusion Matrix Random Forest Scenario 1

L'insieme di dati di training è formato da porzioni eque di ogni dataset, il modello è allenato per riconoscere attacchi di entrambi i dataset. Il set di test, anche esso con metà dei campioni di un dataset e metà dell'altro, è proporzionato alla capacità di classificazione del modello. Pertanto, la combinazione dei dataset porta a performance buone, comparabili alle precedenti.

4.4.2 Combinazione Dataset e Test su Dataset Allenato

Nel secondo scenario, il set di train è formato dalla combinazione di CSE-CIC-IDS-2018-v2 e TON-IoT-v2, costruito come decritto nella sezione precedente 4.4. Mentre il test è effettuato con 9.5 milioni di campioni del dataset CSE-CIC-IDS-2018-v2. Tale dataset di test viene modificato come descritto precedentemente, gli attacchi vengono ricondotti ad una macrocategoria, presente nel dataset di train. Gli attacchi non presenti nel dataset di train, come BruteForce, vengono rimossi dal test set.

Il risultati ottenuti sull'insieme di dati utilizzato per il test vengono indicati nella tabella che segue:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9972	0.9996	0.9984	8317784
Bot	1.0000	1.0000	1.0000	71548
Infilteration	0.9879	0.6014	0.7476	58180
brute-force	0.9929	0.9876	0.9903	61992
ddos	0.9987	0.9993	0.9990	695135
dos	0.9966	1.0000	0.9983	242000
injection	0.5858	0.4583	0.5143	216
accuracy			0.9971	9446855
macro avg	0.9370	0.8637	0.8925	9446855
weighted avg	0.9973	0.9971	0.9969	9446855

Table 4.11: Scenario 2 Random Forest Classification Report.

Il risultati ottenuti sono buoni, il modello è stato allenato per riconoscere tali attacchi, infatti, nella maggior parte dei casi vengono classificati correttamente come facilmente visibile nella matrice di confusione.

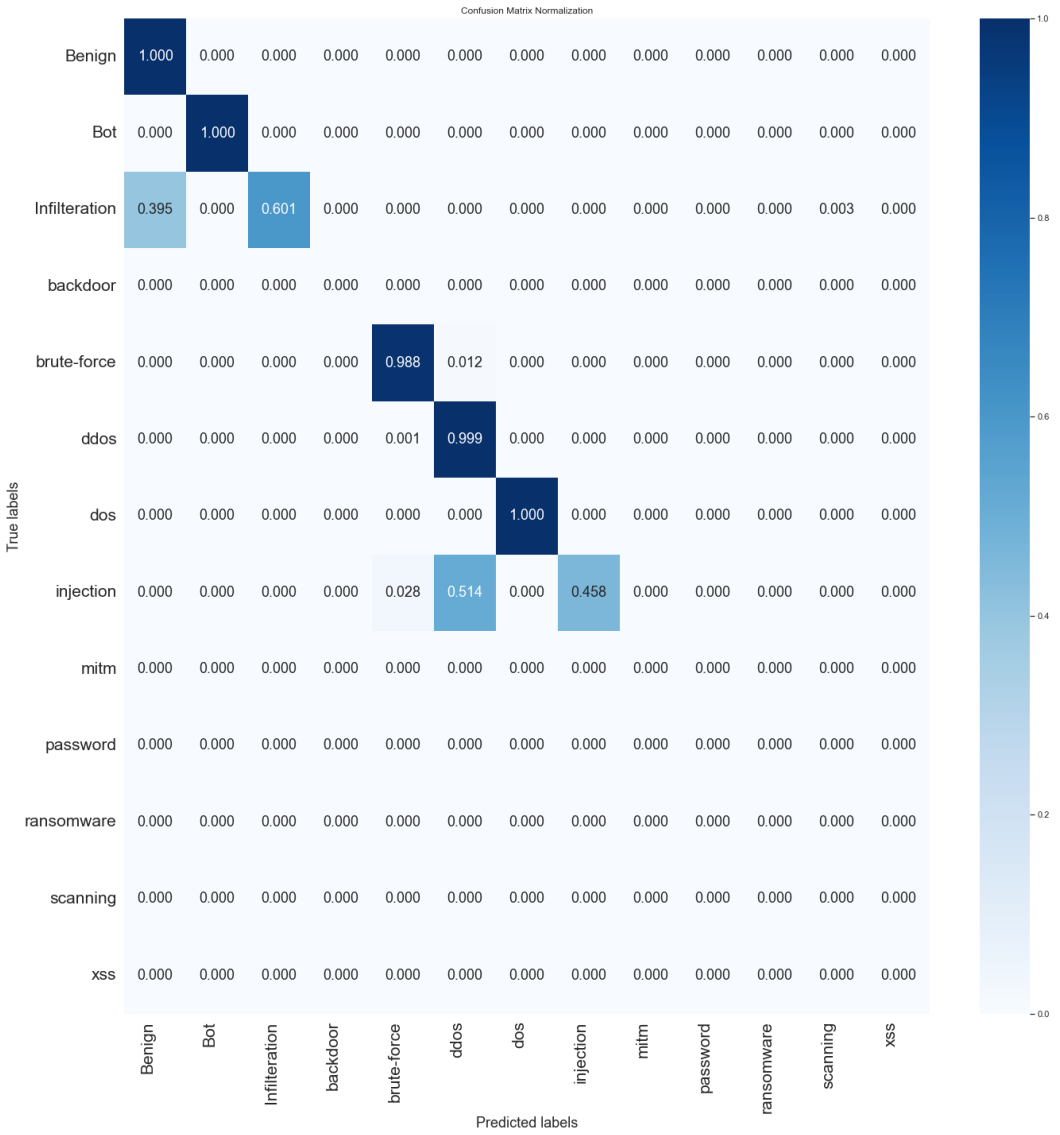


Figure 4.11: Confusion Matrix Random Forest Scenario 2

I valori pari a zero nella matrice sono attacchi non presenti nel dataset di test. Tali valori sono presenti poichè il modello ha classificato alcuni samples come attacchi non presenti nel dataset di test. La normalizzazione porta valori pari a zero poichè rispetto a tutto il campione (9.5 milioni) sono poco significativi.

Gli errori più significativi sono su attacchi come infilteration e injection, errori già riscontrati nelle analisi su singolo dataset. L'attacco di infilteration come nello scenario 1 viene spesso classificato come benigno, ciò è giustificato da quanto spiegato precedentemente.

Pertanto possiamo concludere che nonostante il modello sia allenato per riconoscere molti tipi di attacchi in ambienti di rete diversi testando il modello con un dataset specifico le performance sono buone. Le predizioni nella maggior parte dei casi sono corrette.

4.4.3 Train Dataset e Test Nuovo Dataset

Nello scenario 3, viene replicata una situazione in cui il modello è allenato a riconoscere solamente attacchi provenienti dal dataset NF-TON-IoT-v2. Il test viene effettuato con un altro dataset CSE-CIC-IDS-2018-v2, gli attacchi presenti solamente nel dataset di test vengono eliminati. La stessa operazione di unificazione degli attacchi fra dataset viene effettuata anche in questo caso, attacchi con categorie specifiche nel test set vengono ricondotti ad attacchi più generici nel train set.

Questa situazione simula uno scenario in cui il sistema di intrusione non è allenato a riconoscere attacchi provenienti da un nuovo ambiente di rete con le stesse tipologie di attacco, ma schemi di attacco diversi.

I risultati ottenuti testando il modello sono i seguenti:

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.9448	0.9275	0.9336	8317784
ddos	0.0000	0.0000	0.0000	695135
dos	0.0001	0,0001	0.0001	242000
injection	0.0000	0.0000	0.0000	216
accuracy			0.8336	9255135
macro avg	0.2362	0.2319	0.2334	9255135
weighted avg	0.8491	0.8336	0.8390	9255135

Table 4.12: Scenario 3 Random Forest Classification Report.

La matrice di confusione mostra gli errori di classificazione:

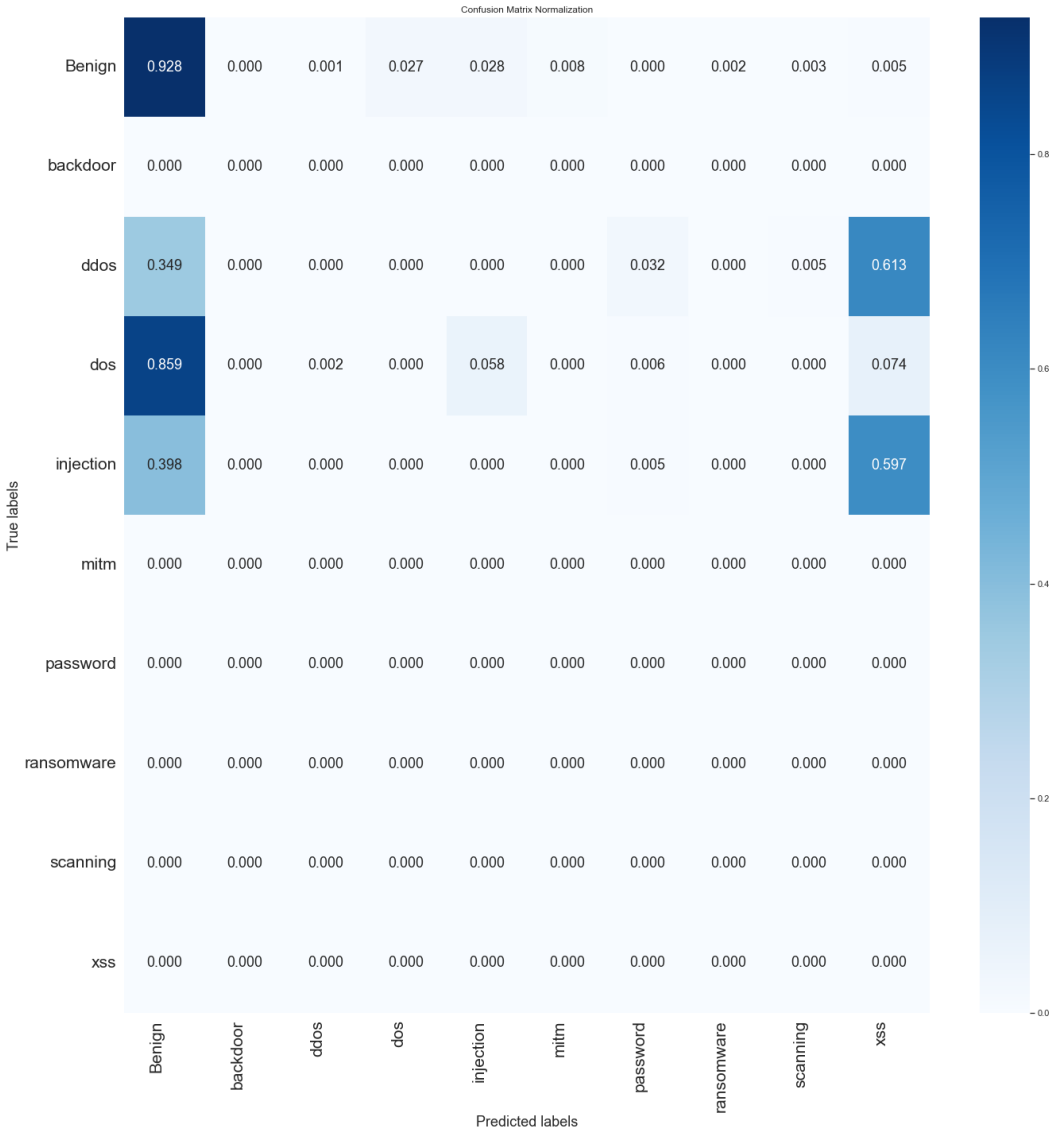


Figure 4.12: Confusion Matrix Random Forest Scenario 3

I risultati ottenuti sono pessimi, la maggior parte degli attacchi viene classificata come traffico benigno. In compenso, attacchi come DDos e Dos vengono comunque rilevati come attacchi, ma nella categoria sbagliata. Inoltre, parte del traffico benigno viene classificato come attacco, ciò produce falsi allarmi ma evita che traffico maligno non venga segnalato.

Il modello non è allenato per riconoscere attacchi appartenenti alla stessa categoria ma con schemi di attacco diversi. Infatti, gli attacchi nei due dataset vengono effettuati utilizzando tool diversi, che simulano attacchi in maniera diversa, i flussi seguiranno pattern diversi. Inoltre, il dataset CSE-CIC-IDS-2018 utilizzato per il test contiene diverse sotto categorie di attacco, molto specifiche rispetto al dataset di train. Per esempio nel caso degli attacchi Dos, spesso classificati come traffico benigno, nel dataset di train vengono simulati con il tool UFONet toolkit [1]. Mentre nel dataset di test con altri tool come *LOIC, HOIC, Hulk, GoldenEye, Slowloris, and Slowhttptest* [14]. Pertanto, nonostante la tipologia di attacco sia la stessa, la metodologia di attacco può cambiare, di conseguenza i flussi di rete prodotti.

Al fine di confermare quanto ipotizzato precedentemente, si analizzano le distribuzioni delle feature più importanti nei due dataset, la distribuzione delle feature per una singola classe non è sovrapponibile con la distribuzione dell'altro dataset. Questo spiega perchè addestrando su un dataset e testando sull'altro le performance non sono buone: gli attacchi sono diversi, le distribuzioni sono diverse, pertanto il modello non è in grado di classificare correttamente gli attacchi.

Per completezza, è stato ripetuto lo scenario 3, in maniera inversa. Il modello viene allenato utilizzando NF-CSE-CIC-IDS-208-v2 e testato utilizzando NF-TON-IoT-v2, con tutte le considerazioni fatte nel caso dello scenario 3. I risultati sono simili al caso precedente, gran parte degli attacchi viene classificata in maniera errata. Non si riportano i risultati ottenuti poichè simili al caso precedente.

I risultati ottenuti dai tre scenari proposti vengono riassunti nella tabella che segue:

		Scenario 1	**Scenario 2**	**Scenario 3**
Precision	Macro AVG	0.9718	0.9370	0.2362
	Weighted AVG	0.9878	0.9973	0.8491
Recall	Macro AVG	0.8683	0.8637	0.2319
	Weighted AVG	0.9878	0.9971	0.8336
F1-Score	Macro AVG	0.8960	0.8925	0.2334
	Weighted AVG	0.9872	0.9969	0.8413
Percentage False Benign		0,46%	2,69%	5,52%
Model Fitting Time(s)		289,96	469,00	376,87
Prediction Time(μs)		6,17	10,47	5,86

Table 4.13: Sum-Up Table of Proposed Scenarios

Le analisi svolte ci permettono di concludere che utilizzando un dataset per il train e un nuovo dataset per il test, gli attacchi non vengono riconosciuti nella maggior parte dei casi. Ciò è giustificato dalla tipologia specifica di attacco, effettuata con tool diversi seguendo schemi di attacco diversi, e confermato dalla distribuzione delle feature più importanti confrontate per la singola classe, la distribuzione tra un dataset e l'altro non è sovrapponibile.

Mentre, se viene utilizzato lo stesso dataset per allenare e testare il modello le performance sono migliori. Infatti, nei primi due scenari il modello riconosce meglio gli attacchi. I pattern presenti nel set di test sono già stati identificati nel set di train. Nonostante ciò, nello scenario 2, rispetto allo scenario 1, la media pesata delle metriche prese in considerazione è maggiore, ma la media aritmetica minore. Infatti, alcune classi vengono classificate in maniera completamente errata e molti più campioni maligni vengono classificati come benigni rispetto allo scenario 1.

Concludendo, per ottenere risultati buoni è necessario che il modello sia allenato a riconoscere schemi di attacco specifici, ciò avviene nei primi due scenari. Viceversa, nel terzo scenario il modello si trova di fronte ad attacchi su cui il modello non è allenato e nella maggior parte dei casi sbaglia la classificazione.

5 Conclusioni e Sviluppi Futuri

Le analisi svolte hanno permesso di confrontare diverse tecniche di Machine Learning per realizzare sistemi con un'importante applicazione pratica. Infatti, l'utilizzo di Network Intrusion Detection System (NIDS) è di fondamentale importanza per proteggere passivamente la rete da possibili attacchi.

Dalle analisi svolte emerge che il miglior metodo sembra essere le Random Forest. Le performance ottenute sono confrontabili con lo stato dell'arte, applicando il modello al dataset NF-TON-IoT-v2 si ottiene *Weighted F1-Score* pari a 0.9819, calcolato a partire da Precision e Recall simili. Il valore di *Macro F1-Score* (0.9216) è comparabile, ciò permette di concludere che la maggior parte delle classi, nonostante il numero ridotto di campioni, vengono classificate correttamente, infatti, i flussi di attacco classificati erroneamente come traffico benigno sono lo 0.30% di tutti i campioni classificati come traffico benevolo.

I risultati migliori si ottengono utilizzando i dataset individualmente. Combinando i dataset e utilizzando lo stesso dataset per train e test le performance diminuiscono leggermente, il classificatore scelto è valido, riesce a imparare anche su pattern più complessi. I risultati peggiori si ottengono nel caso in cui si allena il modello su un dataset e si testa su un altro, nonostante gli attacchi siano gli stessi, lo schema di attacco è diverso, infatti, il modello non classifica correttamente gli attacchi. Pertanto, una corretta etichettatura dei dataset consente di costruire classificatori efficaci ed efficienti, etichette specifiche rappresentative dello schema di attacco, permettono di costruire modelli più precisi.

I risultati ottenuti permettono di capire che al fine di realizzare un IDS efficace è necessario avere una grande quantità di dati provenienti da diversi ambienti di test, la numerosità e la varietà di attacchi rende necessario allenare modelli in grado di riconoscere molti pattern diversi.

Considerato ciò, un possibile sviluppo è l'utilizzo del Federated Learning (FL), un ramo del Machine Learning. Tale approccio addestra un algoritmo su più dispositivi decentralizzati che sono in possesso di dati locali, tali dati non vengono scambiati, a vantaggio della privacy e dell'efficienza computazionale, viene ridotto lo scambio di dati all'interno della rete.

Concludendo, il lavoro svolto ha permesso di analizzare un'applicazione pratica di alcune tecniche di Machine Learning (Alberi di Decisione, Random Forest, Extra Trees, Boosted Decision Tree, Reti Neurali). I modelli costruiti hanno permesso di raggiungere performance confrontabili con lo stato dell'arte.

Inoltre, l'utilizzo di SHAP, ha permesso di comprendere il problema dell'explainability,

per interpretare i risultati ottenuti. Dalle analisi svolte emerge che la feature che influisce maggiormente nella classificazione è *Longest Flow Packet*, cioè la lunghezza (in byte) del pacchetto più grande trasmesso nel flusso. Nell'explanation per classe fra le feature più rilevanti, oltre a Longest Flow Packet, vi sono *L4_Dst_Port*, cioè la porta di destinazione a livello di Trasporto e *In_Bytes*, il numero di bytes in ingresso. Il contributo di SHAP consente di migliorare il grado di fiducia nei modelli sviluppati, infatti è possibile capire meglio le scelte del modello identificando quali feature sono più importanti nella classificazione di un determinato attacco.

Bibliografia

[1] Abdullah Alsaedi et al. "TON-IoT Telemetry Dataset: A New Generation Dataset of IoT and IIoT for Data-Driven Intrusion Detection Systems". In: *IEEE Access* 8 (2020), pp. 165130–165150. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3022862.

[2] Géron Aurélien. *Hands-on machine learning with SCIKIT-learn, Keras, and tensorflow: Concepts, tools, and Techniques*. O'REILLY MEDIA, 2019.

[3] Clarence Chio and David Freeman. *Machine Learning and security*. O'Reilly Media, 2018.

[4] Ali Ghorbani, Wei Lu, and Mahbod Tavallaee. *Network Intrusion Detection and Prevention - Concepts and Techniques*. Vol. 47. Jan. 2010. ISBN: 978-0-387-88770-8.

[5] *GridSearchCV*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[6] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. 7th ed. Boston, MA: Pearson, 2016. ISBN: 978-0-13-359414-0.

[7] *LabelEncoder*. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[8] Thi-Thu-Huong Le et al. "Classification and Explanation for Intrusion Detection System Based on Ensemble Trees and SHAP Method". In: *Sensors* 22.3 (2022). ISSN: 1424-8220. DOI: 10.3390/s22031154. URL: https://www.mdpi.com/1424-8220/22/3/1154.

[9] *Machine Learning-Based NIDS Datasets*. https://staff.itee.uq.edu.au/marius/NIDS_datasets/.

[10] *MinMaxScaler*. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[11] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. "An Explainable Machine Learning-based Network Intrusion Detection System for Enabling Generalisability in Securing IoT Networks". In: *CoRR* abs/2104.07183 (2021). arXiv: 2104.07183. URL: https://arxiv.org/abs/2104.07183.

[12] Mohanad Sarhan et al. "NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems". In: *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer International Publishing, 2021, pp. 117–135. DOI: 10.1007/978-3-030-72802-1_9. URL: https://doi.org/10.1007%2F978-3-030-72802-1_9.

[13] Mohanad Sarhan et al. "Towards a Standard Feature Set of NIDS Datasets". In: *CoRR* abs/2101.11315 (2021). arXiv: 2101.11315. URL: https://arxiv.org/abs/2101.11315.

[14] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *ICISSP*. 2018.

[15] *SKLEARN Official Documentation*. `https://scikit-learn.org/stable/index.html`.

[16] *StandardScaler*. `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`.

[17] The Cylance Data Science Team. *Introduction to artificial intelligence for securit*. The Cylance Press, 2017.

[18] *Weighting Confusion Matrices by Outcomes and Observations*. `https://www.r-bloggers.com/2020/12/weighting-confusion-matrices-by-outcomes-and-observations/`.

[19] *What is intrusion detection system (IDS)*. `https://ehindistudy.com/2015/10/31/what-is-intrusion-detection-system-ids-in-hindi/`.

Le prime persone a cui devo dire grazie per questo traguardo sono i miei genitori e la mia famiglia che mi hanno sempre sostenuto e incoraggiato. Senza di voi, non avrei mai potuto intraprendere questo percorso di studi, non finirò mai di ringraziarvi per avermi permesso di arrivare fin qui. Grazie per esserci sempre stati.

Vorrei ringraziare il Dott. Marco Savi, relatore di questa tesi, e Jacopo Talpini, correlatore di questa tesi, che mi hanno guidato nell'affrontare un argomento complesso, ma allo stesso tempo interessante. Grazie a voi ho avuto modo di avvicinarmi al mondo della ricerca, acquisendo un prezioso bagaglio di conoscenze e competenze che mi aiuteranno nella vita e nel lavoro.

Meritano un "grazie" sincero tutti gli amici, di lunga data o incontrati in questi anni, persone con cui ho condiviso attimi di gioia e di tristezza, ma che nonostante tutto sono rimasti accanto a me in questi anni di Università. Senza di voi, sarebbe stato tutto più difficile: grazie per avermi trasmesso entusiasmo e coraggio per raggiungere i miei obiettivi.

Un grazie speciale a Michela, che mi è stata a fianco in questi anni di studio, è stata capace di capirmi e di sostenermi. Grazie a Michela ho capito che, in fondo, gli ostacoli esistono per essere superati. Grazie per tutto il tempo che mi hai dedicato.