Report descrittivo del notebook

Contesto e obiettivo

Il notebook svolge un'analisi del dataset, con lo scopo di comprendere la struttura dei record di traffico di rete, la composizione delle feature per protocollo e la distribuzione delle etichette legate agli attacchi. Tutto questo per preparare correttamente la fase di preprocessing e la successiva modellazione, evidenziando in anticipo criticità tipiche come colonne con tipi misti, sbilanciamento delle classi, ridondanza tra feature e differenze di scala tra variabili.

Caricamento dell'ambiente e del dataset

All'avvio vengono importate le librerie principali per analisi e visualizzazione, usando "pandas" e "numpy" per la gestione dei dati e "matplotlib" e "seaborn" per i grafici.
Vengono filtrati i warning per mantenere l'output più leggibile e viene impostato uno stile grafico coerente.

Dopo il caricamento, il notebook stampa dimensioni del dataframe, numero di righe e colonne e l'elenco delle colonne. Questa fase serve a verificare che il dataset letto sia quello atteso e che le feature disponibili siano quelle previste.

Ispezione iniziale e controllo di qualità

Viene mostrata un'anteprima delle prime righe per capire rapidamente il formato dei record e individuare i campi target.
Subito dopo, viene usato "df.info()" per osservare i tipi di dato e la presenza di non-null, mentre il conteggio dei valori mancanti per colonna permette di capire se saranno necessari filtri. Inoltre alcune colonne possono contenere sia stringhe (come indirizzi IP) sia valori numerici codificati (come "0"), finendo quindi classificate come "object".

Raggruppamento delle feature per protocollo di rete

Il notebook separa colonne numeriche e colonne categoriche, perché richiedono trattamenti diversi. Sfruttando i prefissi dei nomi delle colonne, calcola quante feature appartengono a ciascun protocollo, includendo ARP, ICMP, HTTP, TCP, UDP, DNS, MQTT e Modbus/TCP.
La motivazione è capire la "provenienza" delle feature e valutare se l'informazione è concentrata su un protocollo o se è distribuita su più strati di rete.

Il conteggio per protocollo viene visualizzato con un grafico a torta, utile come panoramica strutturale prima di passare all'analisi delle etichette.

Analisi delle etichette e dello sbilanciamento delle classi

L’analisi delle label viene fatta su due livelli. Da un lato si esplora “Attack_type”, mostrando la distribuzione dei soli attacchi escludendo la classe “Normal”, così da capire quali categorie di attacco dominano tra gli eventi malevoli e quali risultano minoritarie.
Dall’altro lato si analizza “Attack_label”, una variabile binaria che separa traffico normale e attacco: viene prodotto un grafico a barre con conteggi e viene stampato il rapporto di sbilanciamento.

La motivazione è che lo sbilanciamento influenza direttamente scelta delle metriche e strategie di addestramento. Per “Attack_type” viene inoltre mostrata la distribuzione completa usando un grafico ad istogramma, includendo “traffico normale” con scala logaritmica sull’asse y, perché una scala lineare tende a non mostrare numeri piccoli.
In aggiunta viene prodotta una vista “traffico malevolo” in modo da rendere più leggibile la composizione del traffico malevolo.

Distribuzione dei tipi di attacco

In questa sezione il notebook rappresenta sia grazie ad un grafico ad istogramma che grazie ad un grafico a torta, il volume dei pacchetti per tipo di attacco, differenziando per colore i pacchetti normali e i pacchetti malevoli; in questo modo si ha una rappresentazione grafica chiara del contenuto dei pacchetti suddiviso tra i vari tipi di attacco e il traffico normale.

Protocolli coinvolti per tipo di attacco

Per collegare attacchi e protocolli, il notebook costruisce una matrice binaria che indica, per ciascun “Attack_type”, quali protocolli risultano presenti. La presenza viene stimata controllando un insieme di colonne rappresentative per protocollo e verificando che esista segnale numerico positivo dopo conversione. La motivazione è capire quali famiglie di feature possono essere più discriminanti per alcune categorie e verificare coerenza tra tipologia di attacco e protocollo.

Il risultato viene mostrato con una heatmap annotata, che facilita l’individuazione di pattern tra attacchi e protocolli.

Distribuzione delle feature categoriche

Il notebook analizza le feature categoriche o miste, escludendo esplicitamente i target e un campo temporale, e per ogni colonna mostra i valori più frequenti e il numero di valori unici. La motivazione è identificare campi ad alta cardinalità, come indirizzi o identificativi, che possono richiedere encoding specifico o riduzione, e individuare colonne quasi costanti o con valori anomali che potrebbero introdurre rumore o leakage.

Analisi delle feature numeriche e correlazioni

Sul versante numerico vengono identificate le feature costanti, cioè con un solo valore unico, perché non portano informazione predittiva e possono essere rimosse.
Sulle feature numeriche rimanenti viene calcolata la correlazione con “Attack_label” per ottenere un primo indizio su quali variabili sono più associate alla presenza di attacco.
Viene calcolata anche la varianza, utile per anticipare la necessità di scaling.

Successivamente viene calcolata la matrice di correlazione tra le feature numeriche informative e vengono estratte le coppie con correlazione assoluta elevata oltre una soglia, così da evidenziare multicollinearità e ridondanza. La motivazione è prevenire instabilità e ridondanza nei modelli e supportare scelte come selezione di feature, regolarizzazione o riduzione di dimensionalità.

PREPROCESSING DEL DATASET

1. Valori mancanti

L’analisi ha evidenziato l’assenza di valori mancanti, e per questo non è stato necessario applicare tecniche di riempimento (media, mediana, moda) né rimuovere campi incompleti

2. Rimozione delle feature numeriche con elevata percentuale di zeri

Successivamente è stata analizzata la distribuzione dei valori nelle feature numeriche. È stata calcolata, per ciascuna colonna numerica, la percentuale di valori pari a zero. Sono state eliminate tutte le feature in cui oltre il 95% dei valori risultava pari a zero. Infatti molte variabili nel dataset sono attive solo in presenza di specifici protocolli (DNS, MQTT, HTTP) e risultano nulle nel resto. I motivi dell’eliminazione di queste feature con bassa varianza sono principalmente la scarsa informazione che portano e il beneficio di diminuire il dataset. La soglia del 95% rappresenta un compromesso tra una selezione troppo aggressiva e una troppo soffice. In questo modo vengono rimosse solo le variabili praticamente costanti, mantenendo quelle potenzialmente informative.

Sono state rimosse 25 feature con questa discriminazione.

3. Identificazione e codifica delle variabili categoriche

Una volta rimosse le feature poco informative, le variabili sono state suddivise in feature numeriche e categoriche -distinzione fondamentale poiché i modelli di machine learning richiedono input numerici. Le variabili categoriche devono quindi essere opportunamente

codificate con encoding. È stata adottata una strategia differenziata di encoding, in base alla natura e alla cardinalità delle variabili.

Il One-Hot Encoding è stato applicato alle variabili categoriche a bassa cardinalità, come ad esempio il metodo HTTP, che rappresentano categorie nominali prive di ordine. L'utilizzo del Label Encoding in questi casi avrebbe introdotto relazioni ordinali artificiali (ad esempio GET = 0, POST = 1), inducendo il modello a interpretare un ordine inesistente.

Il Label Encoding è stato invece applicato a variabili ad alta cardinalità, come: indirizzi IP sorgente e destinazione, porte TCP e timestamp. L'utilizzo del One-Hot Encoding su queste feature avrebbe causato un'esplosione dimensionale, generando un numero estremamente elevato di colonne, con conseguente aumento del costo computazionale e rischio di overfitting.Il Label Encoding mantiene la dimensione, pur introducendo un ordine numerico artificiale. Anche se le Random Forest sono poco sensibili a tale ordine, si tratta di un compromesso tra efficienza e significato.

Per alcune variabili contenenti payload, query o stringhe lunghe è stato utilizzato il Binary Encoding data la loro cardinalità molto alta e
i valori spesso quasi unici. Il One-Hot Encoding sarebbe risultato impraticabile, mentre il Label Encoding avrebbe imposto una struttura ordinata, forte e poco significativa. Il Binary Encoding rappresenta un compromesso efficace perché riduce la dimensionalità e preserva più informazione rispetto al Label Encoding.

4. Codifica e Analisi della variabile target

La variabile target "Attack Type", che rappresenta la tipologia di traffico (Normale, DoS, Ransomware, ecc…), è stata trasformata in formato numerico mediante Label Encoding; necessaria perché la classificazione è di tipo multiclasse.

L'analisi della distribuzione della variabile Attack Type evidenzia un forte sbilanciamento: la classe "Normal" rappresenta circa il 72% dei campioni, mentre tutte le altre (che descrivono attività maligne o sospette) hanno frequenze inferiori all'1%.

In presenza di tale squilibrio, un modello tende a privilegiare la classe maggioritaria, ottenendo un accuracy elevata ma scarse prestazioni sulle classi minoritarie, che nel contesto di un sistema di Network Intrusion Detection sono invece cruciali.

Per bilanciare le classi del target è stato usato il RandomUnderSampler, che identifica la classe con il minor numero di campioni (614) e riduce tutte le altre classi, seleziona casualmente i campioni da mantenere, allo stesso numero di campioni di questa classe minoritaria.

Un ulteriore motivo che giustifica l'uso di RandomUnderSampler è la riduzione estremamente significativa di osservazioni, che purtroppo ci costringe a perdere molte informazioni.

5. Suddivisione in training set e test set

Il dataset preprocessato è stato suddiviso in 80% dati di training e 20% dati di test, con un 25% dell'80% del training riservato alla validation, risultando in questa spartizione:

- 60% per il training
- 20% per la validation
- 20% per il test

La suddivisione è stata effettuata in modo riproducibile, grazie all'impostazione di un seed fisso. La suddivisione 60/20/20 garantisce una sufficiente quantità di dati per l'apprendimento, la validazione e un insieme di test adeguato per valutare la capacità di generalizzazione.

Modelli

Il primo modello implementato è stato AdaBoost.
La ricerca degli iperparametri è stata condotta tramite GridSearchCV con validazione incrociata a 5 fold e l'utilizzo come weak learner il Decision Tree Classifier con max_depth=3.
La metrica di ottimizzazione è stata impostata su *f1_weighted*, coerente con la necessità di valutare piu' classi e avere una visuale sulle prestazioni in modo più informativo della sola accuracy.
La valutazione finale è stata prodotta controllando l accuracy e visualizzata tramite matrice di confusione, così da osservare sia la performance globale sia la qualità classe per classe e gli errori ricorrenti di confusione tra categorie.

Random Forest

Il secondo modello è stato Random Forest
Anche qui è stata utilizzata una GridSearchCV a 5 fold
In questo caso l'ottimizzazione è stata fatta su f1 per incentrare i risultati sul avere sia pochi falsi negativi che pochi falsi positivi,
La valutazione, come per AdaBoost, è stata effettuata tramite matrice di confusione visualizzata in heatmap, in modo da ottenere un quadro immediato delle classi più problematiche e del tipo di errori commessi.

Rete neurale

Il terzo modello sviluppato è una rete neurale implementata in PyTorch
La costruzione del modello è stata resa parametrica, prevedendo un primo layer lineare seguito da ReLU, più layer nascosti ripetuti in funzione della profondità selezionata, con dropout per regolarizzare e ridurre overfitting.
L'addestramento è stato impostato con CrossEntropyLoss, adatta alla classificazione multi-classe, e ottimizzazione Adam.
La selezione degli iperparametri è stata eseguita con una grid search "manuale",
Per ogni configurazione è stato introdotto un meccanismo di early stopping basato sulla validation loss con patience pari a 10, così da interrompere l'allenamento quando non si osservano più miglioramenti e limitare sia overfitting sia spreco computazionale.
La scelta della configurazione migliore è stata fatta confrontando le accuracy ottenute, e al termine è stato riportato il best_config e la performance finale sul test set.

Nel complesso, la logica del lavoro sui modelli è stata quella di costruire un confronto ragionato tra Random Forest e AdaBoost, tradizionalmente molto efficaci su dati strutturati, e una rete neurale, che può diventare competitiva quando la rappresentazione delle feature e la regolarizzazione sono curate.
L'utilizzo di metriche diverse e di strumenti diagnostici come classification report e matrici di confusione ha permesso non solo di misurare "quanto" un modello funziona, ma anche "come" sbaglia, informazione cruciale quando l'obiettivo non è solo massimizzare un numero, ma capire la qualità della generalizzazione e l'affidabilità classe per classe.