DOGE: DDoS general-purpose classifier

Progetto di Manutenzione Preventiva

Davide De Zuane, Rahmi El Mechri, Roberto Mustillo

August 2024

1 Introduzione

Il progetto è stato avviato a seguito delle riflessioni presentate nell'articolo [4], che mette in luce la mancanza di generalità nei classificatori attuali e l'importanza della lunghezza delle sequenze nell'analisi degli attacchi DDoS.

L'obiettivo principale di questo lavoro è dunque sviluppare un classificatore per gli attacchi DDoS che sia il più generale possibile rispetto al dataset utilizzato. Per cercare di raggiunere tale obiettivo abbiamo cercato di escludere quelle che sono le caratteristiche specifiche della rete e seguendo le indicazioni che emergono dalla letteratura scientifica su come raggiungere tale scopo. Nella relazione che segue, descriveremo i vari passaggi del nostro percorso e analizzeremo i risultati ottenuti. Il codice sorgente relativo a questo lavoro è disponibile in una repository pubblica [2].

Il lavoro svolto è stato suddiviso nelle seguenti fasi:

- **Dataset analysis**: in questa fase analizziamo i dataset forniti per comprendere la loro struttura e contenuto.
- **Initial Feature Selection**: selezioniamo inizialmente un sottoinsieme di caratteristiche che riteniamo potenzialmente utili per il classificatore. Questa selezione si basa su considerazioni preliminari, come la pertinenza delle feature rispetto al problema da risolvere, l'intuizione o la conoscenza preliminare del dominio.
- **Feature Generation**: in questa fase, creiamo nuove feature a partire da quelle esistenti con l'obiettivo di migliorare la rappresentazione dei dati per facilitare l'apprendimento del modello.
- **Data preprocessing**: comprende varie attività per rendere i dati pronti per l'analisi, come per esempio la feautre encoding e la gestione dei valori nulli.
- **Feature Selection**: eseguiamo una selezione delle feature più rilevanti tramite l'utilizzo di tecniche, l'importance nel caso del random forest. L'obiettivo è ridurre il numero di feature eliminando quelle meno utili, migliorando così l'efficienza e le prestazioni del modello.
- **Training** : addestriamo il classificatore utilizzando i dati selezionati e utilizziamo diverse strategie per mitigare il problema dell'overfitting e a fornire una stima più accurata delle prestazioni del modello.
- **Testing e Confronto**: andiamo a valutare le prestazioni reali. Questa fase include il confronto dei risultati ottenuti con diversi modelli o con diverse configurazioni dello stesso modello.

2 Dataset Analysis

Tutte le informazioni riguardo il dataset utilizzato sono disponibili al seguente link, ne riportiamo le caratteristiche principali.

2.1 Generazione

Questo dataset è stato generato in uno scenario di automazione di processo su piccola scala utilizzando apparecchiature MODBUS/TCP, per la ricerca sull'applicazione delle tecniche di machine learning alla sicurezza informatica nei Sistemi di Controllo Industriale. Il banco di prova emula un processo CPS (Cyber-Physical System) controllato da un sistema SCADA che utilizza il protocollo MODBUS/TCP. Esso consiste in una pompa di liquido simulata da un motore elettrico controllato da un inverter di frequenza variabile (che permette diverse velocità del rotore), a sua volta controllato da un Controllore Logico Programmabile (PLC). La velocità del motore è determinata da una serie di soglie di temperatura del liquido predefinite, la cui misurazione è fornita da un dispositivo MODBUS Remote Terminal Unit (RTU) che fornisce un misuratore di temperatura, simulato da un potenziometro collegato a un Arduino. Il PLC comunica orizzontalmente con l'RTU, fornendo informazioni preziose su come questo tipo di comunicazioni possa influenzare l'intero sistema. Il PLC comunica anche con l'interfaccia uomo-macchina (HMI) che controlla il sistema.

2.2 Struttura

La struttura del dataset è mostata nel seguente albero e all'interno di ogni cartella abbiamo dei file pcap che adottano la seguente convenzione per il naming:

```
1     <capture interface>dump-<attack>-<attack subtype>-<attack duration
         >-<capture duration>
```

```
captures1/
|-- clean/
|-- mitm/
|-- modbusQuery1/
|-- modbusQuery2/
|-- pingFloodDDoS/
‘-- tcpSYNFloodDDoS/

captures2/ & captures3/
|-- modbusQueryFlooding/
|-- pingFloodDDoS/
‘-- tcpSYNFloodDDoS/
```

2.3 Osservazioni

Intuitivamente i risultati migliori si ottengono andando a considerare dataset in cui la presenza di pacchetti malevoli è maggiore in relazione alla cattura totale. Questa osservazione è confermata dai risultati riportati nel paper di riferimento [4], che evidenzia come le migliori performance dei classificatori siano associate a dataset contenenti attacchi della durata di circa 15 minuti rispetto ai 30 minuti di cattura.

In base a queste considerazioni, abbiamo deciso di focalizzare il nostro studio su questa specifica categoria di dataset. Pertanto, tutte le fasi successive della relazione fanno riferimento ai dataset strutturati con una durata di attacco di 15/30 minuti. Questo approccio ci ha permesso di garantire che le condizioni di test e valutazione fossero allineate con le configurazioni che hanno dimostrato di offrire le migliori performance nel contesto della rilevazione e classificazione degli attacchi.

3 Initial Feature Selection

In primo luogo abbiamo utilizzato la nostra conoscenza riguardo le reti per estrarre quelle che, secondo noi tra le informazioni contenute all'interno del pcap, sono le carateristiche da considerare.

In particolare per il classificatore le feature più importanti che abbiamo deciso di considersare, e che poi utilizzeremo in fase di feature generation, sono quelle riportate in *Tabella 1*.

Feature	**Description**
`Time`	Il timestamp in cui il pacchetto è stato catturato.
`Source`	L'indirizzo IP del mittente del pacchetto.
`Destination`	L'indirizzo IP del destinatario del pacchetto.
`Protocol`	Il protocollo utilizzato nel pacchetto (ad es., TCP, UDP).
`Length`	La lunghezza del pacchetto.
`S-Port`	Il numero di porta sorgente utilizzato nel pacchetto.
`D-Port`	Il numero di porta di destinazione utilizzato nel pacchetto.
`Delta-Time`	La differenza di tempo tra pacchetti consecutivi.
`ACK`	Flag di riconoscimento che indica la ricezione di un pacchetto.
`SYN`	Flag di sincronizzazione usato per iniziare una connessione.

Table 1: Feature iniziali

4 Feature Generation

Basandoci sulla letteratura presente, riferendoci in particolare a [1], ed analizzando i problemi specifici che stiamo prendendo in considerazione, abbiamo generato le seguenti features:

- **Entropia source IP**: è una feature che misura la diversità degli indirizzi IP di origine all'interno di un determinato intervallo temporale. L'entropia, in questo contesto, quantifica il livello di dispersione o casualità delle sorgenti di traffico.
- **Entropia source Port**: è una misura della diversità delle porte di origine utilizzate nei pacchetti di rete all'interno di una determinata finestra temporale. L'entropia, in questo contesto, quantifica il livello di casualità o dispersione delle porte di origine.
- **Entropia destination IP**: misura la diversità degli indirizzi IP di destinazione all'interno di un periodo di tempo specifico. Questa metrica riflette quanto il traffico di rete sia distribuito verso diverse destinazioni.
- **Entropia destination Port**: misura la diversità delle porte di destinazione all'interno di una serie temporale. Questa metrica riflette quanto il traffico di rete sia distribuito verso diverse porte di destinazione. Un alto valore di destination port entropy suggerisce che il traffico è diretto verso una varietà di porte di destinazione, indicando la possibile presenza di vari servizi o una scansione di porte. Un basso valore, invece, indica che il traffico è concentrato su poche porte.
- **Packet rate**: misura il numero di pacchetti trasmessi in un'unità di tempo specifica all'interno della serie temporale. Questo valore fornisce un'indicazione del volume di traffico di rete, permettendo di quantificare l'intensità delle comunicazioni in un determinato intervallo temporale.
- **Flow packet number**: è una feature che rappresenta il numero totale di pacchetti associati a un flusso di rete specifico all'interno di un intervallo temporale definito. Un flusso di rete è comunemente identificato dall'insieme di parametri come l'indirizzo IP sorgente, l'indirizzo IP di destinazione, la porta sorgente, la porta di destinazione e il protocollo utilizzato. Questa feature quantifica il volume di pacchetti che costituiscono il flusso, offrendo una panoramica sul traffico generato da una particolare connessione o interazione.
- **Byte rate**: misura il volume totale di dati (in byte) trasmessi per unità di tempo all'interno di una serie temporale. Questa metrica indica la quantità di dati scambiati in un determinato intervallo temporale, fornendo una chiara indicazione del carico di traffico di rete.
- **Rapporto Syn/Ack**: misura il rapporto tra il numero di pacchetti SYN (synchronize) e i pacchetti ACK (acknowledge) osservati in una serie temporale. Un rapporto elevato di SYN rispetto agli ACK può indicare situazioni anomale come tentativi di attacchi SYN flood, mentre se è bilanciato rispecchia il normale funzionamento della rete.

- **Protocol distribution**: si tratta di una serie di feature che descrivono la distribuzione percentuale del traffico in termini di protocolli utilizzati all'interno di una serie temporale. Nello specifico, indica la proporzione del traffico che utilizza TCP, Modbus o Ping. Un improvviso aumento del traffico di un certo protocollo che potrebbe indicare un'attività malevola.

Alcune di queste feature sono calcolate tenendo conto di tutta la storia del dataset, come ad esempio il flow packet number, mentre altri sono calcolati in fase di creazione delle sequenze, come dimensione media pacchetti e packet rate. Questo tipo di dato è quindi dipendente dalla lungheza della sequenza.

Il codice per l'estrazione delle feature si trova in appendice A.

5 Data Preprocessing

In questa sezione analizziamo le tecniche di Data Preprocessing applicate per risolvere alcuni dei classici problemi presenti nel Machine Learning, così da garantire una corretta classificazione.

5.1 Normalizzazione

Per loro natura i classificatori KNN e SVC risultano sensibili alla scala delle feature, ciò ha richiesto l'utilizzo di una tecnica di normalizzazione: il *Min-Max Scaler*. Tale tecnica va a normalizza le feature trasformandole in un intervallo, tipicamente tra 0 e 1, assicurando che tutte contribuiscano equamente alla classificazione.

$$x^{'} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- x è il valore originale della feature
- x_{min} e x_{max} rispettivamente il valore max e min della feature nel dataset
- $x^{'}$ è il valore scalato che sarà compreso in $[0, 1]$

Al contrario, Random Forest e Decision Tree non necessitano di normalizzazione poiché non sono influenzati dalla scala delle feature, basando le loro decisioni su soglie di valore.

5.2 Gestione valori nulli

Durante la fase di processamento dei dati, non è stata necessaria la gestione dei valori nulli, poiché abbiamo progettato la fase di feature generation in modo tale da prevenire la presenza di dati mancanti.

5.3 Creazione tensori

Per la classificazione di serie temporali con HYDRA, i dati devono essere trasformati in un formato adatto alle convoluzioni 1D. Prima, i dati vengono suddivisi in finestre temporali di lunghezza fissa, creando sequenze. Poi, queste sequenze sono convertite in tensori tridimensionali utilizzando PyTorch: Batch Size (numero di sequenze), Sequence Length (lunghezza di ogni sequenza) e Feature Size (numero di caratteristiche per passo temporale). HYDRA applica convoluzioni 1D su questi tensori per estrarre pattern temporali per permettere una classificazione ottimale.

5.4 Gestione della class imbalance

Nel contesto del nostro lavoro, che si concentra sulla classificazione di attacchi DDoS, il problema del class imbalance si manifesta in modo significativo. Questo squilibrio deriva

dalla natura del dataset utilizzato, in cui le varie classi, rappresentanti diversi tipi di attacchi e traffico normale, presentano una distribuzione disomogenea.

Tradizionalmente, per affrontare il problema del class imbalance, si possono adottare tecniche di riequilibrio dei dati. Tuttavia, data la specificità del problema trattato, abbiamo scelto di non intervenire direttamente sui dati per modificare l'equilibrio delle classi. Questa decisione si basa sull'osservazione che l'utilizzo di sequenze basate su finestre di tempo offre una soluzione più naturale ed efficace per mitigare l'imbalance. Grazie a questa suddivisione, ogni classe è rappresentata da un numero uguale di osservazioni nel dataset finale, il che consente di bilanciare l'influenza delle classi durante la fase di addestramento del modello.

5.5 Scaling

Per i dati generati con Hydra applichiamo una seconda trasformazione di scaling attraverso un classe fornita dalla libreria. Questa classe, denominata `SparseScaler` preservare la struttura sparsa dei dati, ottimizzandoli per modelli come Hydra dopo le convoluzioni.

Ne è raccomandato l'utilizzo poichè a seguito dell'applicazione delle convoluzioni è frequente che i dati risultanti diventino più sparsi. Questo accade perché le convoluzioni possono amplificare la presenza di valori zero nelle mappe di caratteristiche, riducendo così la densità dei dati. Questa fase risutla fondamentale per preparare i dati in modo ottimale per le fasi successive di elaborazione o per l'inserimento in altri modelli.

Nel confronto delle performance del modello Hydra con e senza l'applicazione dello SparseScaler, si osserva un miglioramento dell'accuracy quando lo scaler viene utilizzato.

6 Feature Selection

Nella fase di feature selection, abbiamo adottato due distinti approcci per la generazione delle serie temporali, focalizzati rispettivamente sui pacchetti e sul tempo. Questo per aevre due prospettive diverse ripsetto agli attacchi e confrontarle.

- **Serie Temporali Basate sul Numero di Pacchetti**: Il primo approccio si è concentrato su serie temporali costruite a parità di numero di pacchetti.
- **Serie Temporali Basate sul Tempo**: Il secondo approccio prevede serie temporali costruite a parità di intervallo temporale.

6.1 Introduzione

Dopo la fase di generazione delle feature siamo andati ad utilizzare tecniche per determmianre quelle che sono le feature più importanti per la classificazione per ogni approccio. Abbiamo utilizzato diverse tenciche in modo da confrontare i vari risultati. Questo al fine di diminuire il numero di feature da considerare per evitare curse of dimensionality.

Tra le tecniche utilizzate abbiamo:

- **Chi-Square**: è una misura statistica utilizzata per valutare l'importanza delle feature in problemi di classificazione. Si basa sul test del chi-quadrato, una tecnica statistica comunemente usata per determinare se esiste una relazione significativa tra due variabili categoriali.
- **Random Forest Importances**: è una tecnica utilizzata per valutare e interpretare l'importanza delle feature in un modello di classificazione o regressione basato su Random Forest. L'importanza delle feature fornita da una Random Forest aiuta a identificare quali variabili contribuiscono maggiormente alle decisioni del modello.
- **Recursive Feature Elimination (RFE)**:) è una tecnica di selezione delle feature utilizzata per migliorare la performance dei modelli di machine learning riducendo il numero di variabili (o feature) presenti nel dataset. L'idea principale di RFE è quella di costruire modelli iterativi e di rimuovere progressivamente le feature meno importanti per ottenere un set di variabili più rilevante e gestibile.

6.2 Serie basate su pacchetti

I risultati per questa tipologia di serie temporali, con le tecniche descritte, sono riportati in *Tabella 3*. La serie temporale su cui sono stati calcolati è caratterizzata da una window size pari a 20 e ad una packet size pari a 20.

6.3 Serie basate sul tempo

Abbiamo ottenuto i seguenti risultati considerando una size di 20 e una window di 20:

Feature	Chi-2 Score	Importances	RFE
entropia_d_ip	500.85	0.010535	✗
entropia_d_port	6,785.74	0.080121	✓
entropia_s_ip	21,353.25	0.158119	✓
entropia_s_port	8,969.34	0.091637	✓
packet_rate	665,887,900	0.165575	✓
byte_rate	46,294,110,000	0.102822	✓
ping_rate	40,144.66	0.091592	✓
modbus_rate	3,545.83	0.018442	✓
tcp_rate	7,407.87	0.076059	✓
other_rate	940.75	0.004223	✗
synack_ratio	10,536.84	0.040974	✓
avg_deltatime	613.66	0.159903	✓

Table 2: Feature Selection Time Series Pacchetti

Feature	Chi-2 Score	Importances	RFE
entropia_d_ip	4.252658	0.015600	✗
entropia_d_port	7.133164	0.034389	✗
entropia_s_ip	296.3207	0.093706	✓
entropia_s_port	260.3345	0.064228	✓
packet_rate	159,998.8	0.048697	✓
byte_rate	8,956,747	0.053681	✓
ping_rate	519.5008	0.022624	✓
modbus_rate	187.3043	0.187848	✓
tcp_rate	68.37837	0.100644	✓
other_rate	4.555297	0.023983	✓
synack_ratio	22,497.23	0.140747	✓
avg_deltatime	3.874184	0.213853	✓

Table 3: Feature Selection Time Series Tempo

6.4 Osservazioni

Per quanto riguarda RFE le feature selezionato sono praticmante le stesse, quello che risulta più evidente è la differenza nei Chi-2-Score. Infatti dal confronto dei valori di Chi-Square Score, risulta evidente che le feature basate sui pacchetti mostrano punteggi molto

più elevati rispetto a quelle basate sul tempo. Questo suggerisce che le caratteristiche relative ai pacchetti (come byte_rate, packet_rate, e synack_ratio) sono molto più indicative nel distinguere tra traffico normale e attacchi rispetto a quelle basate sui tempi di osservazione.

L'approccio basato sui pacchetti, dal nostro punto di vista che lavoriamo sulle reti, risulta più realistico e potenzialmente più preciso nella pratica. Questi risultati indicano che, per una classificazione accurata del traffico in esame, è preferibile utilizzare feature che riflettono il comportamento dei pacchetti piuttosto che solo aspetti temporali. Questo approccio fornisce una base più solida per la rilevazione e la classificazione degli attacchi, migliorando così l'efficacia del sistema di monitoraggio e difesa.

7 Training

Durante la fase di training del nostro modello, abbiamo adottato due tecniche fondamentali per garantire una valutazione accurata e affidabile delle sue prestazioni: five-fold cross-validation con stratificazione e l'hold-out. Queste metodologie sono state scelte per assicurare che il modello non solo potesse apprendere dai dati disponibili, ma anche che fosse in grado di generalizzare efficacemente su nuovi dati, evitando il rischio di overfitting.

La *five-fold cross-validation* è stata utilizzata per valutare la performance del nostro modello in modo robusto e generalizzabile. Questa tecnica prevede la suddivisione del dataset in cinque parti (fold) di dimensioni approssimativamente uguali. Ogni fold viene utilizzato una volta come set di validazione, mentre i rimanenti fold servono per l'addestramento, garantendo che ogni dato sia utilizzato sia per il training sia per la validazione.

In combinazione con la cross-validation, abbiamo applicato la *stratificazione* per garantire una rappresentazione proporzionale delle classi all'interno di ogni fold. La stratificazione assicura che la distribuzione delle classi sia mantenuta costante in ogni fold, prevenendo il rischio che alcune classi siano sotto o sovra-rappresentate in determinati fold. Questa tecnica è particolarmente importante anche nel nostro caso, dove siamo in presenza di dataset sbilanciati.

L'entità dell'hold-out utilizzato durante questa fase è stata scelta in modo da mantenere un equilibrio tra la quantità di dati riservati per la validazione e quelli utilizzati per l'addestramento. In particolare, il 20% dei dati è stato separato come test set, non solo per valutare la capacità del modello di generalizzare su dati non visti, ma anche per monitorare il rischio di overfitting. Questa scelta ci ha permesso di ottenere una stima più accurata delle prestazioni del modello su nuovi dati, garantendo al contempo che la maggior parte del dataset fosse impiegata per il training.

8 Testing e Confronto

In questa fase, abbiamo seguito due approcci distinti per valutare le time series generate. Nel primo approccio, le serie temporali sono state direttamente date in pasto ad un classificatore tradizionale, che ha analizzato i dati per eseguire la classificazione. Nel secondo approccio, invece, le time series sono state elaborate utilizzando Hydra [3], una rete neurale convoluzionale.

I classificatori che abbiamo utilizzato sono gli stessi del paper di riferimento:

- *Random Forest*
- *Decision Tree*
- *K-Nearest Neighbors (KNN)*
- *Support Vector Classifier (SVC)*

8.1 Hyperparameter tuning

È stato necessario fare hyperparameter tuning unicamente per il KNN, per poter impostare:

- **n**: numero di punti che si vuole considerare come "neighbours" in fase di classificazione.
- **weights**: parametro che specifica come i "neighbours" influiscono nella votazione. In paricolare se si vuole che il voto contribuisca in modo inversamente proporzionale alla distanza (valore impostato a "distance"), o in modo uniform (valore impostato a "uniform").

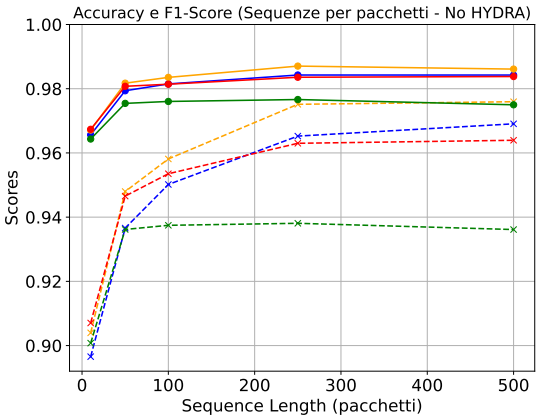
Come tecnica abbiamo utilizzato il Grid Search con 5-fold cross validation, che consiste nel testare tutte le possibili combinazioni di parametri specificando il range di prova. In particolare nel nostro caso abbiamo testato valori di n interi compresi tra 1 e 30, testando con peso uniforme e proporzionale alla distanza. In fase di classificazione abbiamo utilizzato i parametri ottenuti in questo modo.

8.2 Serie Basate su Pacchetti

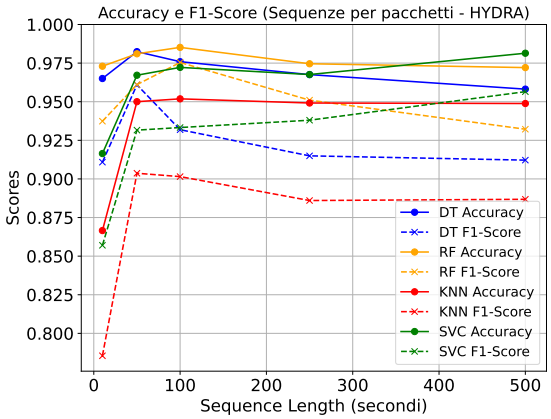
In *Figura 1* riportiamo il confronto tra i risultati dei vari classificatori utilizzati, riportandone l'accuracy e l'F1 (espressi in percentuale). Abbiamo anche considerato come variano tali valori al variare della dimensione della serie temporale considerata.

8.3 Serie Basate su Tempo

Un approccio analogo si è utilizzato nel considerare le serie temporali basate sul tempo, in cui siamo andati a valutare l'accurracy al variare della dimensione dell'intervallo di tempo considerato. I risultati sono riportati in *Figura 3*.

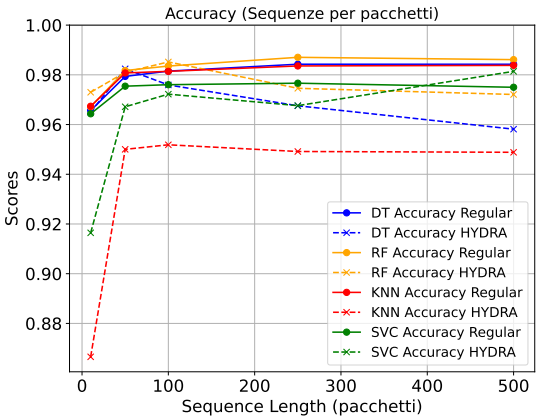


(a) Utilizzo di Dati Aggregati

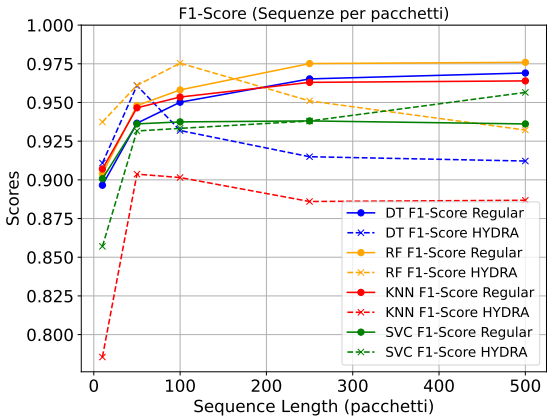


(b) Utilizzo di Hydra

Figure 1: Serie Temporali su Pacchetti

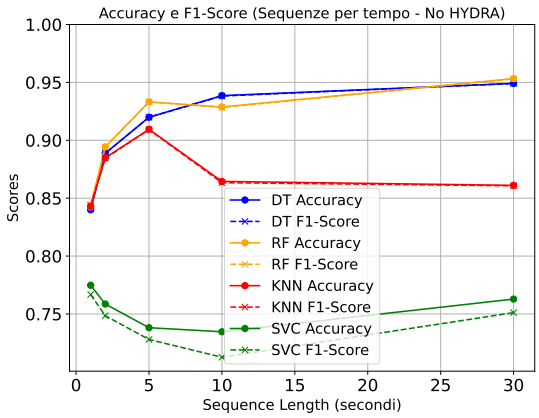


(a) Comparazione accuracy

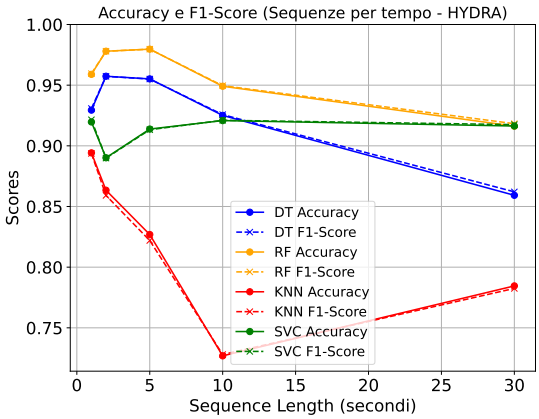


(b) Comparazione f1score

Figure 2: Comparazione risultati (serie basate su pacchetti)

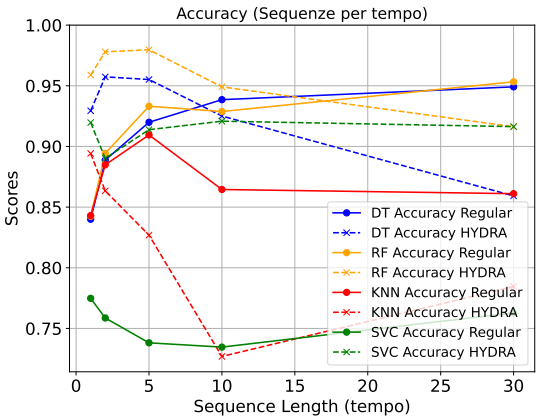


(a) Utilizzo di Dati Aggregati

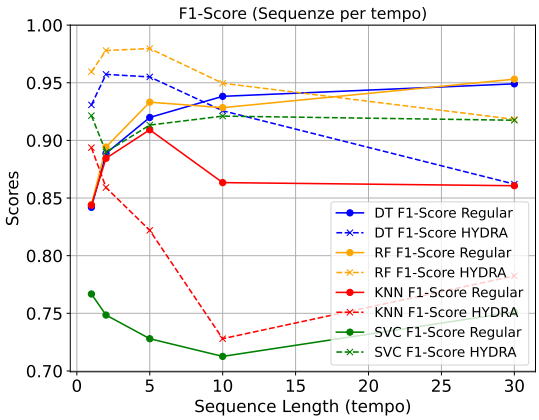


(b) Utilizzo di Hydra

Figure 3: Serie Temporali su Tempo



(a) Comparazione accuracy



(b) Comparazione F1Score

Figure 4: Comparazione risultati (Serie basate su tempo)

8.4 Osservazioni

In questa sezione descriviamo e commentiamo i risultati ottenuti attraverso la classificazione, osservabili in 1, 2, 3, 4.
Questo confronto ci aiuta ad identificare quale combinazione di classificatore e lunghezza della serie temporale offre i migliori risultati in termini di accuratezza nella classificazione degli attacchi.

8.4.1 Accuracy e F1Score

Dalla comparazione dei risultati con serie basate su pacchetti in Figura 2 vediamo che senza l'utilizzo di Hydra si hanno in genere valori di accuracy e f1score più elevati. Invece, dalla comparazione dei risultati con serie basate su tempo in Figura 4, si può notare che solo per le sequenze più piccole, con l'utilizzo di Hydra, si hanno valori di accuracy e f1score più elevati, che tendono però ad abbassarsi con l'aumentare della sequence length.Questo vale ad eccezione del classificatore Support Vector, le cui accuracy ed f1score sono migliori utilizzando Hydra.

È opportuno evidenziare che in entrambi i casi, ad eccezione per le sequenze di dimensione ridotte, utilizzando hydra il KNN classifier ha un deterioramento delle prestazioni peggiore rispetto agli altri classificatori.

Senza l'utilizzo di Hydra i classificatori basati su Random Forest e Decision Tree tendono a migliorare le prestazioni all'aumentare della lunghezza delle sequenze in entrambi i casi. Utilizzando Hydra invece, non si hanno particolari variazioni nel caso delle serie basate su pacchetti, contrariamente a quanto avviene nel caso delle sequenze basate su tempo.

Per quanto riguarda KNN distinguiamo 4 casi:

- Nel caso di serie su pacchetti, senza l'utilizzo di Hydra, si ha un miglioramento continuo di accuracy e f1-score.
- Nel caso di serie su pacchetti, con l'utilizzo di Hydra, siamo di fronte ad un miglioramento nelle piccole sequenze e poi ad un deterioramento, fino a rimanere costante.
- Anche nel caso di serie su tempo, senza l'utilizzo di Hydra, si ha un miglioramento per le piccole sequenze, seguito da un lieve deterioramento fino a rimanere costante.
- Nel caso di serie su tempo, con l'utilizzo di Hydra, si ha un netto deterioramento seguito da un lieve miglioramento.

Il SVC classifier, invece, distinguiamo:

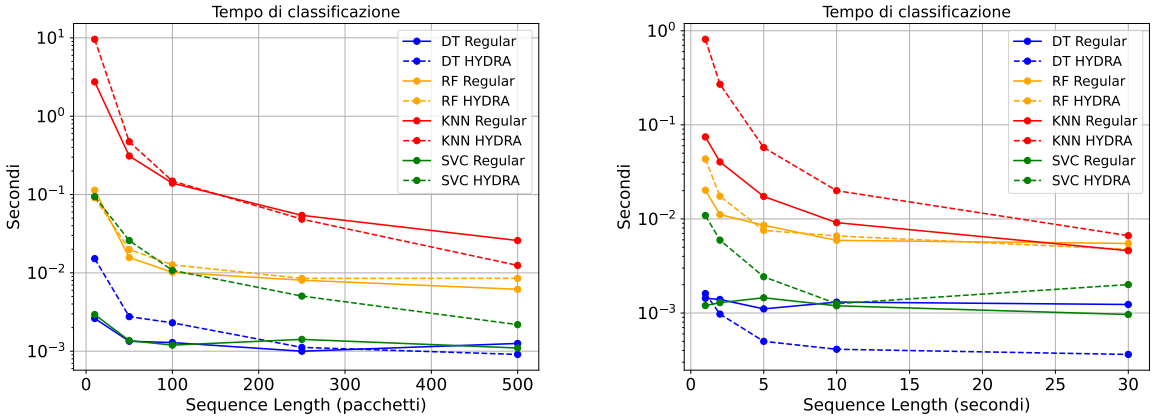
- Nel caso di serie su pacchetti, senza l'utilizzo di Hydra, si ha un miglioramento iniziale di accuracy e f1-score che tende ad essere costante.

- Anche nel caso di serie su pacchetti, con l'utilizzo di Hydra, siamo di fronte ad un miglioramento quasi costante.
- Nel caso di serie su tempo, senza l'utilizzo di Hydra, si ha un deterioramento per le piccole sequenze, seguito da un lieve miglioramento per le sequence length più grandi.
- Nel caso di serie su tempo, con l'utilizzo di Hydra, si ha un deterioramento quasi continuo.

Nella relazione, non riportiamo direttamente alcuni dei parametri di valutazione come le matrici di confusione e di correlazione per ciascun modello. Tuttavia, questi dati sono disponibili nel codice allegato, che contiene tutte le analisi dettagliate e i risultati ottenuti durante le nostre sperimentazioni.

8.4.2 Tempi di training e classificazione

In *Figura 5* sono riportati i tempi di classificazione impiegati dai classificatori considerando le due tipologie di serie temporali. In *Figura 6* sono riportati invece i tempi necessari per il training. La scala dei tempi è logaritmica per evidenziare la distribuzione non uniforme dei tempi qunado si considerano serie di piccole dimensioni.

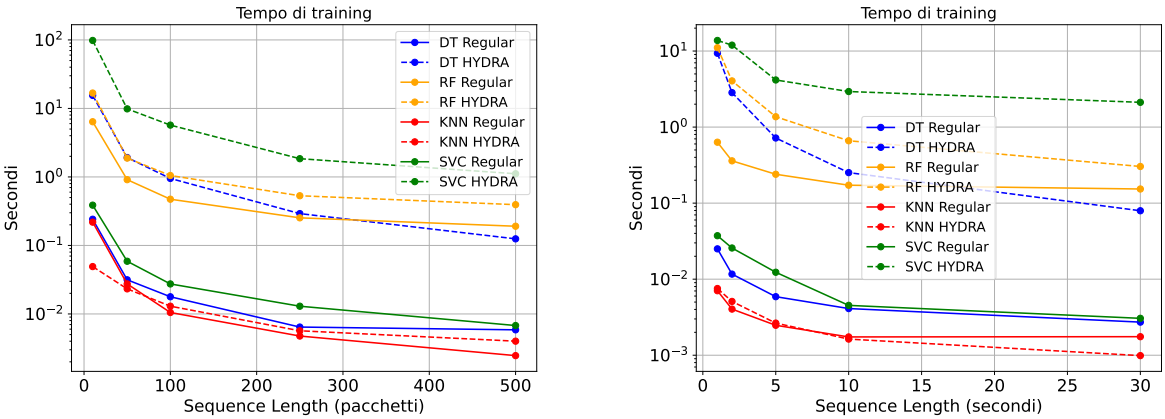


(a) Sequenze per pacchetti

(b) Sequenze per tempo

Figure 5: Tempi di classificazione

I tempi medi di classificazione ottenuti, che riflettono i tempi complessivi necessari per classificare gli hold-out, sono compatibili con un utilizzo in tempo reale per la rilevazione di attacchi DoS, anche nel caso di classificatori più onerosi come il KNN. Questo indica che i classificatori sono in grado di elaborare i dati e fornire risultati in modo sufficientemente veloce da poter essere impiegati in modo real-time.



(a) Sequenze per pacchetti

(b) Sequenze per tempo

Figure 6: Tempi di training

9 Conclusioni

L'obiettivo principale del progetto di riuscire a costruire dei classificatori con delle prestazioni solide che non dipendano eccessivamente dalle caratteristiche della rete non può dirsi pienamente raggiunto senza testare i modelli ottenuti su ambienti diversi ed eterogenei. Ciò nonostante, per la rete presa in esame, siamo riusciti ad ottenere dei modelli con prestazioni ottime, generando dataset le cui feature non dipendono in modo esclusivo dalle caratteristiche della rete, garantendo quindi un bias limitato, che andrà verificato in futuro tramite altre applicazioni. I tempi di classificazione dei modelli permettono un utilizzo real-time della classificazione se opportunamente programmata. L'inclusione del framework HYDRA non ha apportato particolari miglioramenti alle prestazioni, mostrando risultati simili o peggiori nella maggioranza delle combinazioni. Lavori futuri possono concentrarsi sulla generazioni di modelli che facciano rilevamento di nuovi attacchi informatici su reti. Il materiale ottenuto è disponibile nella sua interezza nella repository di progetto [2].

References

[1] Muhammad Aamir and Syed Mustafa Ali Zaidi. "DDoS attack detection with feature engineering and machine learning: the framework and performance evaluation". In: *International Journal of Information Security* 18 (2019), pp. 761–785.

[2] Roberto Mustillo Davide De Zuane Rahmi El Mechri. URL: `https://github.com/rahmec/general-ddos-classification`.

[3] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. URL: `https://github.com/angus924/hydra`.

[4] Ivo Frazão et al. "Denial of Service Attacks: Detecting the Frailties of Machine Learning Algorithms in the Classification Process:" in: Dec. 2018, pp. 230–235. ISBN: 978-3-030-05848-7. DOI: 10.1007/978-3-030-05849-4_19.

A Feature generation

Riportiamo di seguito il codice python con cui sono state generate le feature. Per velocizzare il processo sono stati utilizzati i meccanismi per il multiprocessing integrati in python.

```python
import logging
import pandas as pd
import time
from multiprocessing import Pool

def add_flow_number_column(pcap_df):
    pcap_df['Flow'] = pcap_df['Source'] + '->' + pcap_df['Destination']
    pcap_df['FlowNumber'] = 0
    flows = {flow_key: 0 for flow_key in pcap_df['Flow'].unique()}
    for index, row in pcap_df.iterrows():
        pcap_df.at[index, 'FlowNumber'] = flows[row['Flow']]
        flows[row['Flow']] += 1
    return pcap_df

def aggregate_sequences(df, seq_length, window_size, mean):
    aggregated_data = []

    df_range = []
    if mean == "packets":
        df_range = range(0, len(df) - seq_length + 1, window_size)
    elif mean == "time":
        df_range = range(0, 60*30-seq_length, window_size)

    for start in df_range:
        if mean == "packets":
            sequence = df.iloc[start:start + seq_length]
            seq_seconds = sequence['Time'].iloc[seq_length-1] - sequence[
                'Time'].iloc[0]
            seq_packets = seq_length
        elif mean == "time":
            sequence = df[df['Time'] >= start]
```

```
            sequence = sequence[sequence['Time'] < start + seq_length]
            seq_seconds = seq_length
            seq_packets = len(sequence)
            if seq_packets == 0:
                continue

        length_sum = sequence['Length'].sum()
        byte_rate = length_sum/seq_seconds
        packet_rate = seq_packets/seq_seconds
        avg_flow_number = sequence['FlowNumber'].mean()
        avg_deltatime = sequence['Delta-Time'].mean()
        num_modbus = (sequence['Protocol'] == 'Modbus/TCP').sum()
        num_tcp = (sequence['Protocol'] == 'TCP').sum()
        num_ping = (sequence['Protocol'] == 'ICMP').sum()
        num_other = seq_packets - num_modbus - num_tcp - num_ping
        num_syn = (sequence['SYN'] == 'Set').sum()
        num_ack = (sequence['ACK'] == 'Set').sum()
        modbus_rate = num_modbus/seq_length
        tcp_rate = num_tcp/seq_packets
        ping_rate = num_ping/seq_packets
        other_rate = num_other/seq_packets
        if num_syn == 0:
            synack_ratio = 0
        elif num_ack == 0:
            synack_ratio = 1
        else:
            synack_ratio = num_syn/num_ack
        source_entropy = sequence['Source'].nunique()/seq_packets
        destination_entropy = sequence['Destination'].nunique()/
           seq_packets
        s_port_entropy = sequence['S-Port'].nunique()/seq_packets
        d_port_entropy = sequence['D-Port'].nunique()/seq_packets
        label = sequence['Label'].iloc[0]

        aggregated_row = {
                'byte_rate' : byte_rate,
                'packet_rate' : packet_rate,
                'avg_flow_number' : avg_flow_number,
                'avg_deltatime' : avg_deltatime,
                'source_entropy' : source_entropy,
                'destination_entropy' : destination_entropy,
                's-port_entropy': s_port_entropy,
                'd-port_entropy': d_port_entropy,
                'synack_ratio' : synack_ratio,
                'modbus_rate' : modbus_rate,
                'tcp_rate' : tcp_rate,
                'ping_rate' : ping_rate,
                'other_rate' : other_rate,
                'label' : label
        }

        aggregated_data.append(aggregated_row)

```

```
    aggregated_df = pd.DataFrame(aggregated_data)

    return aggregated_df

def aggregate_df(seq_length, mean):
    window_size = seq_length
    aggregated_dfs = []
    for df in dfs:
        agg_df = aggregate_sequences(df, seq_length, window_size, mean)
        aggregated_dfs.append(agg_df)

    aggregated_df = pd.concat(aggregated_dfs)
    if mean == "packets":
        print(f'CREATED csv/aggregated_{seq_length}packets_df.csv')
        aggregated_df.to_csv(f'csv/aggregated_{seq_length}packets_df.csv'
            , index=False)
    elif mean == "time":
        print(f'CREATED csv/aggregated_time_{seq_length}s_df.csv')
        aggregated_df.to_csv(f'csv/aggregated_time_{seq_length}s_df.csv',
            index=False)

clean_df = pd.read_csv('csv/clean15-30.csv')
tcpsynflood_df = pd.read_csv('csv/tcpsynflood15-30.csv')
pingflood_df = pd.read_csv('csv/pingflood15-30.csv')
modbusflood_df = pd.read_csv('csv/modbusflood15-30.csv')

clean_df['Label'] = 0
tcpsynflood_df['Label'] = 1
pingflood_df['Label'] = 2
modbusflood_df['Label'] = 3

dfs = [clean_df, tcpsynflood_df, pingflood_df, modbusflood_df]

packet_seq_lengths = [10,20,50,100,250,500]
time_seq_lengths = [1,2,5,10,30]
threads = []

print("Generating flow numbers")

start = time.time()
pool1 = Pool(processes=len(dfs))

df_results = []
for df in dfs:
    df_results.append(pool1.apply_async(add_flow_number_column, [df]))

pool1.close()
pool1.join()

for i in range(len(dfs)):
    dfs[i] = df_results[i].get()

print("Generating and starting processes")
```

```
pool2 = Pool(processes=(len(packet_seq_lengths)+len(time_seq_lengths)))
packet_res = []
for seq_length in packet_seq_lengths:
    packet_res.append(pool2.apply_async(aggregate_df, [seq_length,"packets"]))
print(f"Started processes for packet sequences with length {packet_seq_lengths}")
time_res = []
for seq_length in time_seq_lengths:
    time_res.append(pool2.apply_async(aggregate_df, [seq_length,"time"]))
print(f"Started processes for time sequences with length {time_seq_lengths}")

pool2.close()
print("Waiting until all processes are done")
pool2.join()
end = time.time()

for res in (packet_res + time_res):
    if res.get() != None:
        print(res.get())

print("CSVs created!")
print(f"Execution time: {end-start} seconds")
```