ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Triennale in Ingegneria e Scienze Informatiche

Uso del Machine Learning per la detection dei domini DGA (Domain Generation Algorithm)

Tesi di laurea in:
Programmazione a Oggetti

Relatore
Prof. Mirko Viroli

Candidato
Simone Collorà

I Sessione di Laurea
Anno Accademico 2024-2025

Sommario

I Domain Generation Algorithms (DGA) sono algoritmi che generano domini in modo pseudo casuale. Questi domini vengono utilizzati dai malware per comunicare con i loro server, i Command and Control servers (C&C). I DGA sono stati sviluppati per superare le limitazioni dei metodi precedenti come i domini hard-coded, i quali sono prevedibili e facilmente bloccabili. I DGA, invece, sono quasi impossibili da prevedere e da bloccare con i metodi tradizionali, come le blacklist, poiché, i domini generati, sono migliaia e pseudo casuali. Lo scopo di questo progetto di tesi è quello di analizzare e sviluppare soluzioni per la rilevazione dei DGA attraverso il Machine Learning, cercando di creare un modello in grado di rilevarli e distinguerli dai domini legit. Sono stati analizzati e sperimentati quattro algoritmi di Machine Learning per la rilevazione dei DGA, due algoritmi classici basati su tecniche di feature extraction, ovvero le caratteristiche estratte dai dati(i domini nel nostro caso), e due basati su tecniche di Deep Learning, che quindi impiegano delle reti neurali capaci di imparare direttamente dai dati. I due algoritmi di Machine Learning basati su feature extraction sono il Random Forest e l'XGBoost, mentre i due algortimi di Deep Learning sono la Long Short-Term Memory (LSTM) e la Bidirectional LSTM. I modelli sono stati addestrati e testati su un dataset di domini DGA e legit diviso in training, validation e test set. I risultati sono basati su diverse metriche e sono stati confrontati tra i vari modelli. I modelli con risultati migliori sono stati quelli basati su deep learning con un'accuretezza del 99% e una differenza di accuratezza del 7% circa rispetto agli altri modelli, mentre, quelli più performanti in termini di tempo e risorse, sono stati quelli basati su feature extraction impiegando quasi 30 volte in meno rispetto ai modelli di deep learning.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La sicurezza informatica è un argomento di crescente importanza nel mondo moderno. Con il passare del tempo, i sistemi di protezione sono diventati sempre più sofisticati e potenti ma, allo stesso tempo, anche gli hackers hanno sviluppato tecniche sempre più avanzate per eludere i sistemi di protezione. Tra queste vi è sicuramente l'uso di Botnets e dei Command and Control(C&C) servers. I C&C sono dei server che manipolano una rete di computer infetti da malwares, chiamati Botnets, permettendo all'attaccante di eseguire codice malevolo da remoto. Il malware, però, deve conoscere un indirizzo IP o un dominio per contattare il server. L'attaccante potrebbe inserire in modo hard-coded l'indirizzo IP del server nel codice del malware, ma questo metodo è facilmente rilevabile e bloccabile. Gli hackers, quindi, preferiscono utilizzare dei domini generati in modo pseudo casuale da degli algoritmi per nascondere i loro server chiamati Domain Generation Algorithms(o DGA). Questi algoritmi generano migliaia di domini al giorno che vengono poi utilizzati dai malware per contattare i server C&C, rendendo difficile per i sistemi di protezione bloccarli poiché appena un dominio viene bloccato, il malware passa semplicemente al prossimo dominio generato. I metodi tradizionali per bloccare i domini malevoli come le blacklist, risultano inefficaci per i DGA poiché i domini generati sono troppi ed è molto difficile e dispendioso risalire al seed usato per generare i domini tramite reverse engineering. Perciò, negli ultimi anni, sono stati sviluppati vari metodi di Machine Learning capaci di rilevare e bloccare i DGA in modo più efficace.

Struttura della tesi La tesi si dividerà in 3 capitoli. Nel primo capitolo verrano descritti concetti di base riguardanti DNS, DGA e Botnets e Machine Learning con una breve descrizione delle reti neurali e degli algoritmi utilizzati. Nel secondo

capitolo verrà descritto il progetto, i suoi obiettivi e i risultati ottenuti. Nell'ultimo capitolo verranno discusse le conclusioni e i possibili sviluppi futuri.

Capitolo 2

Background

Di seguito sono descritti i concetti base su cui si basa il progetto.

2.1 Domini e Domain Name System

Un **dominio** è un nome univoco che identifica un sito web. I domini sono composti da due parti principali, il nome e l'estenzione del dominio. Il nome del dominio è la parte principale che identifica il sito web ad esempio `google` in `google.com`. L'estensione del dominio, invece, è la parte finale che identifica la provenienza del sito o lo scopo. Degli esempi possono essere `.com`, il più usato, per i siti commerciali, `.it` per i siti italiani, `.org` per le organizzazioni. Un dominio può essere registrato da chiunque tramite un **Registrar**, ovvero un'azienda che gestisce la registrazione dei domini. Il dominio sarà unico e non potrà essere registrato da nessun altro fino a quando esso non verrà rimosso o scadrà. Un dominio può essere registrato per un periodo di tempo che va da un minimo di un anno a un massimo di dieci anni. Un dominio può essere composto da lettere, numeri e trattini ma non può iniziare o finire con un trattino. Un dominio può essere composto da un massimo di 63 caratteri e non può contenere spazi o caratteri speciali e può essere diviso in sottodomini, che sono parte del dominio principale e sono separati da un punto. Un esempio di sottodominio può essere `translate.google.com` in cui `translate` è il sottodominio di `google.com` per Google Translate. Il dominio solo però non basta per raggiungere un sito web. È necessario conoscere l'indirizzo IP del server che ospita il sito web. Questo problema viene risolto dal Domain Name System.
Il **Domain Name System** (più spesso abbreviato DNS) è un sistema che traduce i nomi di dominio in indirizzi IP. Quando un utente inserisce un dominio nel browser,

il DNS cerca l'indirizzo IP associato a quel dominio e lo restituisce al browser che poi si connetterà al sito web desiderato.

2.2 Botnets e DGA

2.2.1 Botnets

I **Botnets** sono reti di computer infetti da malware, chiamati bot, che possono essere controllati da uno o più hackers, chiamati **botmaster**. La vita di un botnet nella maggior parte dei casi è divisa in 4 fasi:

1. **Infezione e propagazione**: Questo è il primo passaggio. L'hacker cerca di infettare un computer tramite vari metodi come email con link malevoli o Peer-to-peer(P2P) sharing. Una volta infettato un dispositivo, il malware cerca di infettare altri dispositivi nella rete.

2. **Rallying**: i bots cercano di contattare per la prima volta il server Command and Control(C&C) per far capire all'hacker che l'attacco è andato a buon fine.

3. **Commands and Reports**: il malware esegue le istruzioni ricevute dal server C&C e invia i risultati al botmaster. I bots ascoltano continuamente il server C&C o si connettono ad esso periodicamente. Appena ricevono un comando lo eseguono, inviano i risultati al botmaster e aspettano un nuovo comando.

4. **Abbandono**: Quando un bot non è più utile o utilizzabile, il botmaster può decidere di abbandonarlo. Il botnet, invece, sarà completamente distrutto quando tutti i bot saranno abbandonati o bloccati dalla vittima o quando il C&C server verrà bloccato

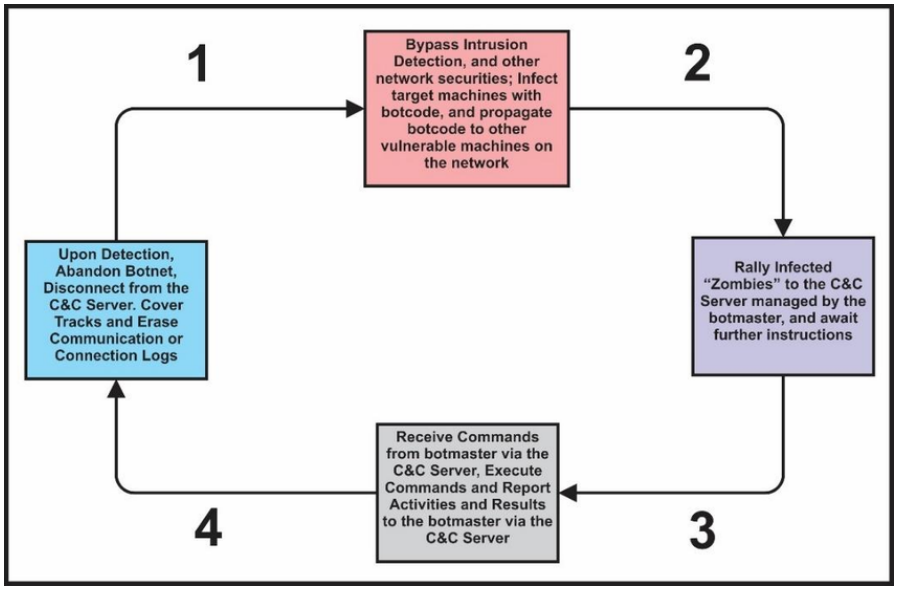

Figura 2.1: Ciclo di vita di un botnet [10]

2.2.2 Command and Control

Il meccanismo del Command and Control, spesso abbreviato in C&C, crea un canale di comunicazione tra il botmaster e i bot. Questo è essenziale per il funzionamento del botnet. Ci sono tre tipi di server C&C:

- **Centralizzati**: In questo tipo di server, il botmaster controlla tutti i bot tramite un server centrale. Questo è il metodo più semplice e veloce per controllare i bot ma è anche il più vulnerabile. Se il server centrale viene bloccato, tutti i bot non possono più ricevere comandi. Questo a sua volta è diviso in due categorie:
  - **Internet Relay Chat (IRC)**: IRC è un sistema di chat usato per comunicare tra i bot e il botmaster in tempo reale. Questo era più usato nella prima generazione di botnet. I bot si connettono al server IRC e aspettano i comandi dal botmaster. I bot seguono un approccio PUSH ovvero quando un bot si connette ad un determinato canale, esso rimane connesso.
  - **HTTP / HTTPS**: Il più usato. Con questa tecnica, i bot usano un URL o IP per contatattare il server C&C. Qua invece i bot seguono un approccio PULL. I bot si connettono al server C&C periodicamente e

controllano se ci sono nuovi comandi. Questo processo va ad intervalli regolari definiti dal botmaster.

- **Decentralizzati**: Questo tipo di C&C è basato su un sistema P2P senza un server centrale. In questo modo, computer infetti fanno sia da bot che da server. Questo metodo è più difficile da rilevare ma anche più complesso da implementare [12].

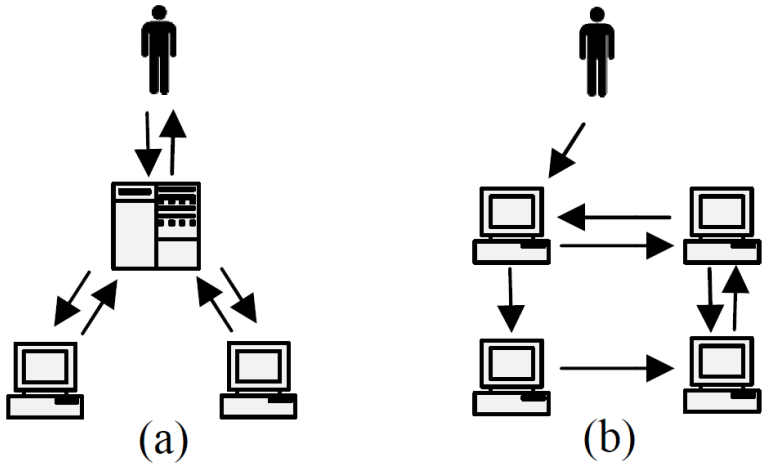

Figura 2.2: Esempio di server C&C. (a) centralizzato, (b) decentralizzato [9]

2.2.3 Domain Generation Algorithm

I Domain Generation Algorithm (DGA) sono algoritmi che generano migliaia di domini in modo pseudo casuale. Prima viene scelto un seed, che può essere la data odierna o anche le previsioni meteo [1] e, tramite un algoritmo di hashing, vengono generati i domini. Questi domini vengono poi utilizzati per contattare i server C&C. Non tutti i domini generati però sono registrati. Il computer infetto, tramite i DNS locali, cercherà di tradurre un dominio in un indirizzo IP. Se esso non riesce a contattare il server con un determinato dominio, proverà con il successivo finché non troverà un dominio valido che permetterà al malware di comunicare con il server C&C [2]. In questo modo, diventa più difficile per i sistemi di protezione rilevare e bloccare i loro attacchi. Si potrebbe pensare di bloccare direttamente i domini tramite una blacklist ma questo metodo risulta inefficace poiché vengono generati migliaia di domini continuamente. Si pensi che Conficker C, un famoso malware che utilizza DGA, è in grado di generare fino a 50.000 domini pseudo casuali al giorno [3]. Un altro modo per contrastare ciò potrebbe essere quello di

fare reverse engineering del DGA per capire quale seed viene utilizzato per generare i domini. Questo però risulta lento e dispendioso e possibilmente inefficace [8].

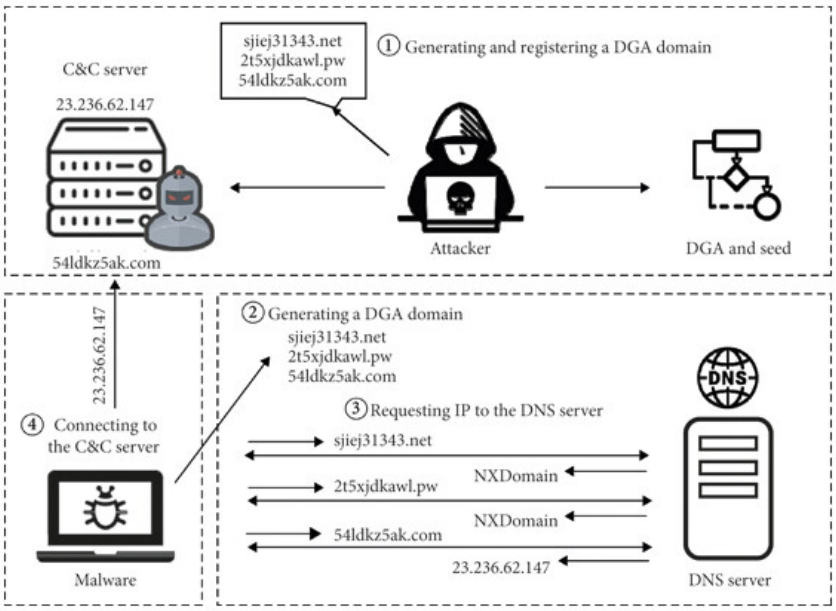

Figura 2.3: Esempio del funzionamento di un DGA [8]

Per contrastare i DGA, sono stati sviluppati vari metodi di Machine Learning in grado di rilevare i domini generati. Questi metodi hanno due lati poisitivi:

- Non richiedono un lungo processo di reverse engineering.
- Essendo l'AI una blackbox, è molto difficile se non impossibile per gli hackers eseguire un reverse engineering del modello.

2.3 Machine Learning

Il machine learning è la branca dell'intelligenza artificiale che si occupa dello sviluppo di algoritmi e tecniche finalizzate all'apprendimento automatico mediante la statistica computazionale e l'ottimizzazione matematica.[7]

2.3.1 Sviluppo di un modello di Machine Learning

Il primo passo per sviluppare un modello di Machine Learning è quello di raccogliere i dati in un dataset. Nel nostro caso, dovendo riconoscere se un dominio è

lecito o DGA, il dataset conterrà entrambi i tipi di domini che potranno avere o no un etichetta, DGA o legit nel nostro caso, A seconda di come è strutturato il dataset possiamo avere vari tipi di allenamento:

- **Supervised Learning**: È la tecnica più comune per allenare i modelli [15]. In questo tipo di apprendimento, il modello viene addestrato su un dataset etichettato.
- **Unsupervised Learning**: In questo tipo di apprendimento, il modello, deve scoprire dei pattern o delle relazioni senza avere nessuna etichetta. Il modello deve trovare degli oggetti che condividono delle caratteristiche simili, chiamati cluster
- **Reinforcement Learning**: In questo tipo di apprendimento, ogni azione ha un effetto nell'ambiente che può essere positivo o negativo.

Si è soliti dividere il dataset in tre parti ovvero training set, validation set e test set. Il **training set** è il dataset su cui viene addestrato il modello, il **validation set** è una parte del training set utile per riconoscere se il modello è in grado di generalizzare su dati nuovi ovvero non va in overfitting o underfitting durante l'addestramento. il **test set**, invece, è il dataset su cui viene testata la precisione del modello. Per la divisione del dataset non esiste una regola fissa, ma solitamente il training set è più grande del validation set e del test set. Un esempio potrebbe essere 90% training set, 10% validation set e 10% test set. Successivamente vengono inizializzati i parametri del modello che per un modello classico come il Random Forest possono essere il numero di alberi o il numero di features da usare. Per un modello di deep learning, invece, i parametri possono essere il numero di epoche, il numero di batch, il learning rate e il numero di neuroni per ogni layer. Un modello troppo semplice, con pochi alberi o pochi neuroni, potrebbe non essere in grado di apprendere i pattern nei dati (**Underfitting**), mentre al contrario, una modello troppo complesso potrebbe imparare troppo bene i dati e non generalizzare su dati nuovi (**Overfitting**). Dopo di che si passa alla fase vera e propria di addestramento. Per un modello classico dipende dal tipo di algoritmo usato. Per il random forest, ad esempio, viene creato un certo numero di alberi e, per ogni albero, viene calcolata la previsione che sarà la maggioranza delle previsioni degli alberi, mentre per XGBoost, viene calcolato l'errore del modello e viene creato un nuovo albero che cercherà di correggere l'errore del modello precedente ad ogni iterazione. Per i modelli di Deep Learning, invece, esso viene iterato per un certo numero di epoche su un batch di dati. Il numero di **epoche** è un iperparametro, ovvero un parametro che non cambia durante l'addestramento, che indica quante volte l'algoritmo

di apprendimento deve passare attraverso il dataset di training[23]. Un **batch** è un sottoinsieme del training set dopo il quale il modello aggiorna i propri pesi. Ad esempio abbiamo un training set di 10000 dati e un batch size di 100. Ogni 100 dati il modello calcolerà l'output e la loss function, cioè la funzione che calcola la differenza tra l'output previsto e quello reale, e aggiornerà i pesi di conseguenza. Infine viene testato il modello sul test set e, se la precisione è soddisfacente, il modello potrà essere usato.

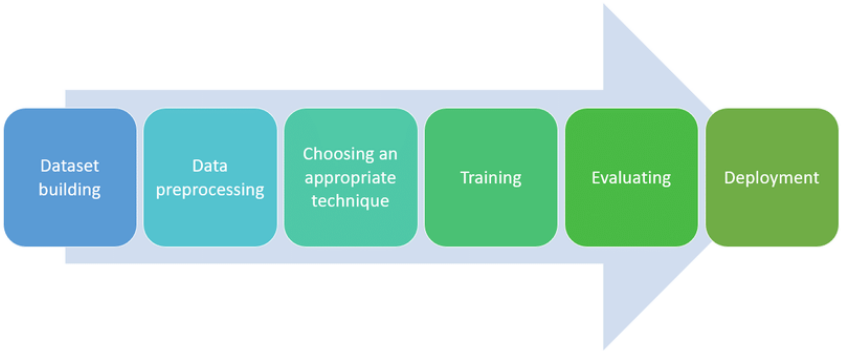

Figura 2.4: Pipeline di un modello di Machine Learning [24]

2.3.2 Reti Neurali

Una **Rete Neurale** o in inglese **Artificial Neural Network** (ANN) è un modello matematico che mira a simulare il funzionamento del cervello umano [14]. Il cervello umano è composto da miliardi di neuroni che comunicano tra di loro tramite sinapsi. Con le reti neurali artificiali, il funzionamento è analogo. A livello matematico, un neurone artificiale è composto principalmente da due componenti:

- **Pesi**: i pesi(weights in inglese) sono valori numerici che aiutano ogni nodo della rete neurale a determinare l'importanza di un input. Usando i pesi, il neurone può decidere se un input è importante o meno.

- **Funzione di attivazione**: la funzione di attivazione del neurone è la funzione che prende in input la sommatoria dei dati pesati con i pesi descritti in precedenza e produce un output che verrà poi inviato ad altri neuroni come input. Alcune delle funzioni di attivazione più comuni sono la funzione sigmoide e la funzione ReLU.

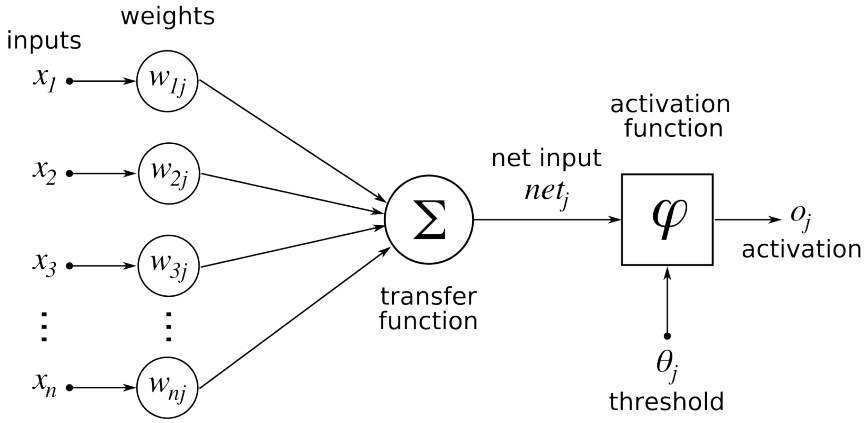

Figura 2.5: Esempio di neurone artificiale [17]

A sua volta una rete neurale è composta da vari strati:

- **Input Layer**: il primo strato della rete neurale e quello che riceve l'input esterno. Poiché la rete neurale è un modello matematico, l'input dovrà essere un vettore numerico. Questo vale anche se l'input da esaminare è una stringa di caratteri. In questo caso la stringa va convertita in un vettore numerico come vedremo in seguito.

- **Hidden Layer**: questo è lo strato intermedio della rete neurale in cui avviene il processo di apprendimento. Questo strato elabora gli input ricevuti dallo strato precedente modificando i pesi. Si possono avere anche più hidden layers

- **Output Layer**: Questo è l'ultimo strato della rete neurale. Fornisce i risultati finali ottenuti dalla rete neurale. Questo strato può essere composto anche da un solo neurone che fornisce un output binario (0 o 1) come sarà nel nostro caso(DGA o legit)

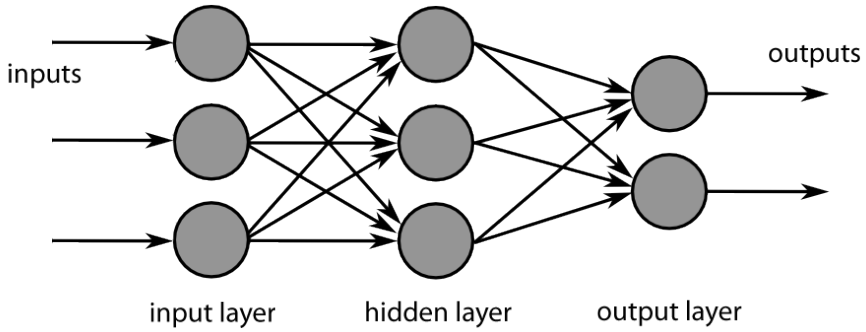

Figura 2.6: Esempio di rete neurale [18]

Abbiamo inoltre vari tipi di reti neurali:

- **Reti Neurali Feedforward(FNN)**: Una rete Feedforward elabora le informazioni in un solo verso. Non ha né cicli né memoria delle informazioni passate. Questo tipo di rete è usato ad esempio per il riconoscimento di pattern.

- **Reti Neurali Ricorrenti (RNN)**: A differenza di una rete Feedforward, in una RNN i neuroni possono andare anche a formare dei cicli. Questo permette alla rete di avere una specie di memoria e quindi di ricordare le informazioni passate. Usata per esempio per traduzione automatica o riconoscimento vocale.

- **Reti Neurali Convoluzionali (CNN)**: Questo tipo di rete è usato principalmente per dati strutturati a griglia ad esempio le immagini. Il nome "Convoluzionali" deriva dal fatto che la rete usa un'operazione matematica chiamata proprio convoluzione. Le CNN sono usate soprattuto nell'ambito della visione artificiale per il riconoscimento di oggetti [20].

2.4 Algoritmi di Machine Learning adatti per DGA

Ora che abbiamo visto i concetti di base del Machine Learning, delle reti neurali e dei DGA, vediamo alcuni degli algoritmi che possono essere usati per rilevare i domini generati da DGA.

2.4.1 Random Forest

Random Forest è un algoritmo esamble (cioè unisce più modelli) di Machine Learning supervisionato che usa un insieme di alberi decisionali per classificare i dati. Un albero decisionale è un modello che usa appunto una struttura ad albero per prendere decisioni. Un esempio può essere quello nella fig. 2.7, in cui, a seconda di determinate caratteristiche, l'albero decide se una persona rischia di avere un attacco cardiaco o meno. Il problema principale del singolo albero decisionale è che è molto suscettibile all'overfitting.

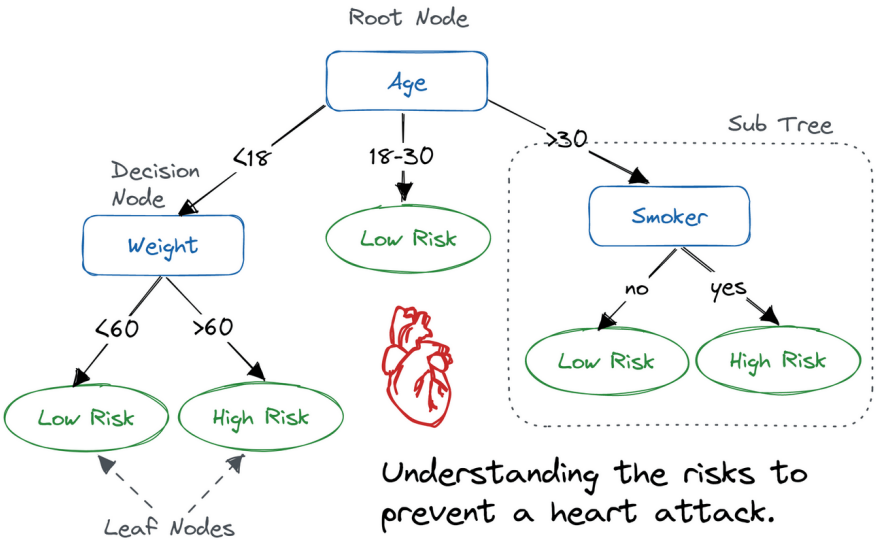

Figura 2.7: Esempio di albero decisionale [25]

Il random forest risolve questo problema aumentando l'accuratezza del modello sia in fase di training che in fase di testing. [30] Esso usa un apporccio basato su feature, ovvero vengono selezionate delle caratteristiche del dataset che possono essere utili per la classificazione. Ad esempio, nel nostro caso, le caratteristiche potrebbero essere il numero di vocali, lunghezza del dominio o l'entropia. L'algoritmo funziona nel seguente modo: Per prima cosa vengono creati n alberi decisionali, ognuno contenente una parte casuale del training set. Inoltre ogni albero decisionale viene addestrato su un sottoinsieme casuale di feature. Dopo di che, ogni albero fa la sua previsione, e la previsione viene fatta in base a ciò che la maggior parte degli alberi ha previsto se si tratta di un problema di classificazione mentre, qualora si trattasse di un problema di regressione, la previsione finale sarà la media delle previsioni di tutti gli alberi. Questa tecnica è chiamata **Bagging** o Bootstrap Aggregating.

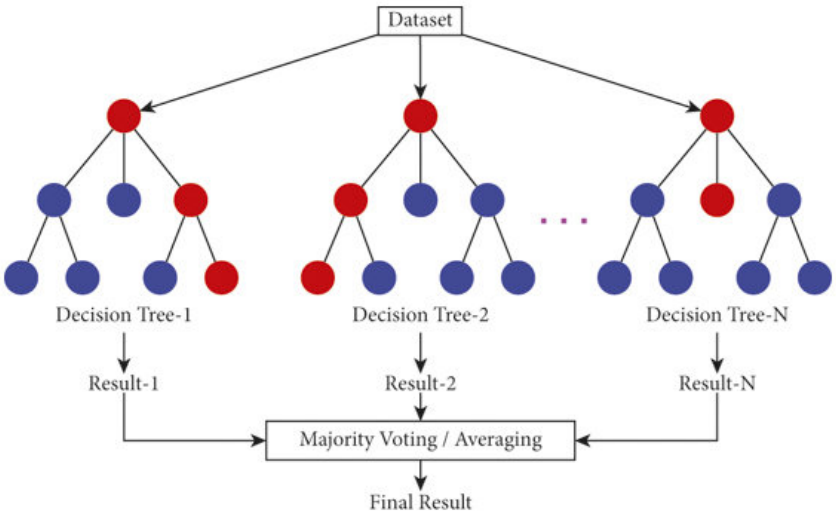

Figura 2.8: Esempio di Random Forest [26]

2.4.2 XGBoost

XGBoost, abbreviazione di eXtreme Gradient Boosting, è un algoritmo di Machine Learning esamble che combina più alberi decisionali come il Random Forest. La differenza principale tra i due algoritmi è che XGBoost usa un approccio **Boosting** invece che Bagging. Il Boosting è un metodo che combina più modelli messi in sequenza, in modo che ogni modello successivo cerchi di correggere gli errori del modello precedente. Il Boosting funziona nel seguente modo:

1. Viene creato un albero decisionale e viene allenato sui dati.

2. Viene calcolato quante predizioni sono state sbagliate

3. Viene creato un nuovo albero decisionale che verrà addestrato sugli errori del modello precedente.

4. Si ripete il processo fino a quando non verrà soddisfatto un criterio di stop.

La differenza tra il boosting standard e XGBoost è che quest'ultimo risulta più veloce, più performante e meno soggetto all'overfitting grazie all'uso di tecniche di regolarizzazione come il L1 e L2 regularization e la parallelizzazione del processo di addestramento [29]

2.4.3 Long Short Term Memory (LSTM)

Le LSTM sono un tipo di rete neurale ricorrente (RNN). A differenza delle RNN tradizionali, esse sono in grado di memorizzare informazioni per periodi di tempo più lunghi e possono avere più strati nascosti. Create da Hochreiter e Schmidhuber nel 1997 [21], le LSTM risolvono uno dei problemi più importanti delle RNN standard, il vanishing gradient problem. Questo problema si verifica poiché durante la backpropagation, i gradienti tendono appunto a svanire, rendendo difficile l'apprendimento di relazioni a lungo termine. Per risolverlo, le LSTM introducono una struttura dati vettoriale chiamata **Memory Cell**(o cella di memoria). Questa viene viene aggiornata ad ogni step tramite le operazioni decise dai "gates" (o porte). La struttura delle LSTM è composta nel seguente modo:

- **Input Gate**: determina quali nuove informazioni devono essere aggiunte allo stato della cella. Usa una funzione sigmoide per decidere quali valori saranno aggiunti e una tanh per creare nuovi valori

- **Forget Gate**: gestisce quali informazioni devono essere dimenticate dallo stato della cella. Usa una funzione sigmoide per decidere quali informazioni saranno mantenute o dimenticate. Se il valore è 0 le informazioni saranno dimenticate mentre se è 1 saranno mantenute.

- **Output Gate**: decide come sarà il prossimo hidden state Gli input sono passati a una funzione sigmoide e la cella aggiornata viene passata a una tanh e moltiplicata con l'output della sigmoide per decidere quali informazioni saranno passate.

Oltre alle LSTM tradizionali abbiamo anche le **BLSTM** (Bidirectional LSTM) che sono composte da due LSTM, che elaborano i dati in due direzioni diverse. A differenza del Random Forest e del XGBoost, le LSTM usano un approccio featureless e imparano direttamente dai dati.

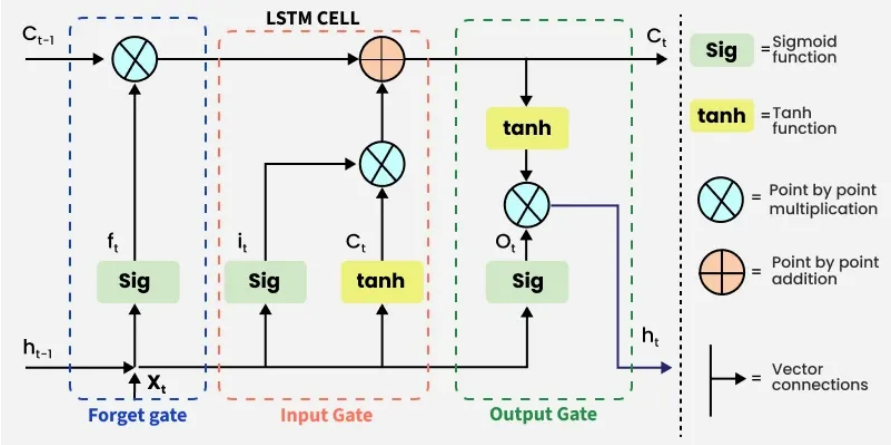

Figura 2.9: Struttura di un LSTM [32]

Capitolo 3

Progetto

Il progetto ha come obiettivo quello di sviluppare un sistema di rilevamento di domini generati da DGA tramite l'uso di tecniche di Machine Learning. Per il dataset ho deciso di usare il dataset di YangYang [33] che contiene circa 1.5 milioni di domini DGA e circa 1.5 milioni di domini legittimi.

3.1 Linguaggi e librerie usate

Per il progetto è stato usato il linguaggio Python poiché è il più usato in ambito Machine Learning e analisi dei dati e offre una vasta gamma di librerie adatte a questo scopo. Le librerie usate sono:

- **Pandas**: libreria per la manipolazione e l'analisi dei dati.
- **Numpy**: libreria per il calcolo scientifico e l'elaborazione di array.
- **math**: libreria per le funzioni matematiche di base.
- **Scikit-learn**: libreria per il Machine Learning che offre vari algoritmi e strumenti per la valutazione dei modelli Nel nostro caso è stata usata per la divisione del dataset in training, validation e test set, per l'implementazione del modello Random Forest e per il calcolo delle metriche di valutazione.
- **XGBoost**: libreria per l'implementazione dell'algoritmo XGBoost.
- **Tensorflow e Keras**: librerie per la creazione di reti neurali. Usate per la tokenization dei domini e per l'implementazione delle LSTM e BLSTM.
- **Matplotlib**: libreria per la visualizzazione dei dati e per la creazione di grafici.

3.2 Metriche di valutazione

Per valutare le prestazioni dei vari modelli si è deciso di usare i seguenti parametri: Accuracy, Pecision, Recall, F1-Score e Confusion Matrix.
L'**Accuracy** è il rapporto del numero di predizioni corrette sul numero totale di predizioni effettuate. La formula è la seguente:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

dove:

- TP è il numero di True Positive (Il numero di DGA classificati correttamente)
- TN è il numero di True Negative (Il numero di domini legittimi classificati correttamente)
- FP è il numero di False Positive (Il numero di domini legittimi classificati come DGA)
- FN è il numero di False Negative (Il numero di DGA classificati come domini legittimi)

La **Precision** è il rapporto tra il numero di True Positive e il numero totale di predizioni positive effettuate. La formula è la seguente:

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

La **Recall** è il rapporto tra il numero di True Positive e il numero totale di positivi ed è definita come segue:

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

L'**F1-Score** è la media armonica tra Precision e Recall. ed è definita come:

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.4}$$

Nel progetto è stata usato Classification Report di scikit-learn per calcolare queste metriche. La **Confusion Matrix** è una matrice che mostra il numero di predizioni corrette e sbagliate per ogni classe. Un vantaggio della Confusion Matrix è che permette di vedere quali classi sono state classificate correttamente e quali no e la quantità precisa di ogni classe.

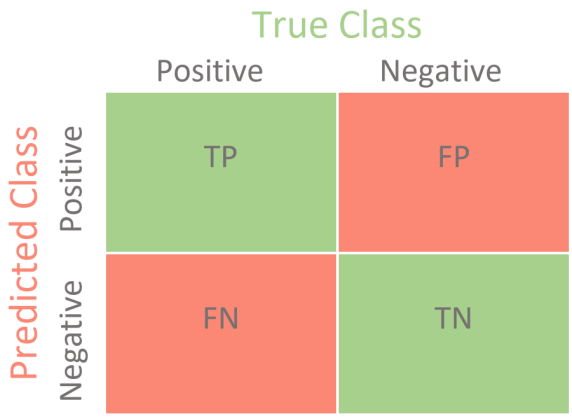

Figura 3.1: Esempio di Confusion Matrix [34]

3.3 Test con Random Forest

Il primo algoritmo testato è stato il Random Forest. Qui non abbiamo bisogno di convertire i domini in altri formati poiché il modello prenderà come dati non il dominio in sé ma le sue caratteristiche(o features). I parametri che prende il modello sono:

- **n_estimators**: il numero di alberi decisionali da creare Di default è 100. Si è provato ad aumentare il valore finchè, a parte un maggiore tempo di addestramento, non si è notato alcun miglioramento significativo. Alla fine si è deciso di usare il valore 200.

- **criterion**: la funzione usata per misurare la qualità di una divisione. Di default usa la funzione "gini".

- **max_depth**: la profondità massima di ogni albero. Di default è None, ovvero gli alberi cresceranno fino a quando tutti i nodi saranno puri o fino a quando tutte le foglie conterranno meno di min_samples_split campioni.

- **min_samples_split**: il numero minimo di campioni richiesti per dividere un nodo interno. Di default è 2.

- **min_samples_leaf**: il numero minimo di campioni richiesti per essere una foglia. Di default è 1.

- **max_features**: il numero massimo di feature da considerare per la divisione di un nodo. Di default è "sqrt" ovvero la radice quadrata del numero di feature.

- **random_state**: il seed per la generazione casuale dei dati. Di default è None, ma facendo così il modello resituirà risultati diversi. Perciò si è deciso di dargli un seed fisso.

A parte per il random_state e il numero di alberi gli altri parametri sono stati lasciati con i valori di default poiché, dopo vari test non sono stati riscontrati grossi miglioramenti. Inizialmente sono state usate come features la lunghezza del dominio, il numero di vocali, il numero di consonanti, la quantità di numeri e l'entropia del dominio. L'entropia misura la randomicità di una stringa e viene calcolata con la seguente formula:

$$Entropy = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i) \tag{3.5}$$

dove p_i è la probabilità di ogni carattere nel dominio.
Successivamente sono stati aggiunti altri parametri come il ratio tra consonanti e vocali, il numero di trattini, l'entropia per bigramma, e il numero di consonanti e vocali consecutive. Random Forest possiede un metodo che permette di calcolare l'importanza di ogni feature nel modello chiamato `feature_importances_` Grazie ad esso si è notato che le features più importanti sono state l'entropia e quante vocali consecutive ci sono nel dominio. Alcune features come se il dominio contiene un numero all'inizio o alla fine o se il dominio contiene una wordlist di parole comuni ai DGA sono state rimosse poiché non portavano un miglioramento significativo e anzi rischiavano di portare a un overfitting del modello.

	domain	type	length	entropy	consonants	consonants_ratio	vocals	vocals_ratio	digits	hyphens	consonants_vocals_ratio	consonant_clusters	digit_character
0	nylc.org	0	8	3.000000	5	0.625000	2	0.250000	0	0	2.500000	3	0
1	hesgoals.com	0	12	3.251629	7	0.583333	4	0.333333	0	0	1.750000	2	0
2	comic-doujin.com	0	16	3.155639	8	0.500000	6	0.375000	0	1	1.333333	2	0
3	yjeffanansnan.cc	1	16	2.952820	11	0.687500	4	0.250000	0	0	2.750000	4	0
4	simpliciaty.com	0	15	3.323231	9	0.600000	5	0.333333	0	0	1.800000	2	0
5	1tg1wuq15geug4rv4gfz1uzpys2.net	1	31	4.026619	17	0.548387	5	0.161290	8	0	3.400000	4	5
6	i-fa.biz	0	8	2.750000	3	0.375000	3	0.375000	0	1	1.000000	2	0
7	xfdtestnessbiophysicalohax.com	1	30	4.015061	20	0.666667	9	0.300000	0	0	2.222222	5	0
8	caqpktcxupjiejca.eu	1	19	3.471354	11	0.578947	7	0.368421	0	0	1.571429	3	0
9	tecamac.gob.mx	0	14	3.235926	8	0.571429	4	0.285714	0	0	2.000000	2	0

Figura 3.2: Parte del dataset con i domini e loro features

Il modello ha impiegato circa 10 minuti per l'addestramento e, dopo vari test, si è raggiunta un accuratezza del 91% ma con risultati peggiori nel riconoscimento dei DGA rispetto ai legit come si può vedere nella fig. 3.3.

	precision	recall	f1-score	support
Legit	0.89752	0.93036	0.91365	159595
DGA	0.92659	0.89219	0.90907	157245
accuracy			0.91142	316840
macro avg	0.91206	0.91127	0.91136	316840
weighted avg	0.91195	0.91142	0.91137	316840

Figura 3.3: Risultati del modello Random Forest

Dalla Confusion Matrix 3.4 si può vedere che il modello ha classificato più di 5000 False Negative ovvero domini DGA classificati come legittimi rispetto ai False Positive.

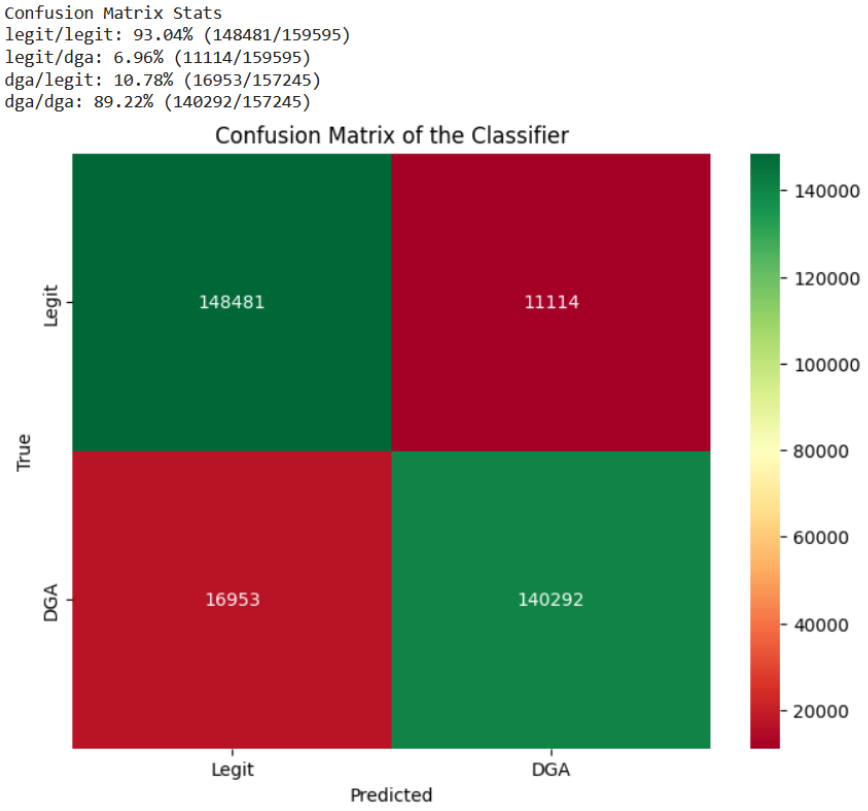

Figura 3.4: Confusion Matrix del modello Random Forest

3.4 Test con XGBoost

XGBoost come Random Forest non ha bisogno dei dati convertiti in vettori numerici ma delle loro features. Le features usate sono state le stesse del modello Random Forest. I parametri che XGBoost prende in input sono:

- **n_estimators**: il numero di alberi da creare. Di default è 100. Essendo un modello sequenziale, il numero di alberi può essere aumentato di molto e aggiungere un early stopping per evitare l'overfitting. L'early stopping è un metodo che interrompe l'addestramento quando il modello non migliora più. Il numero di alberi che si è scelto è 10000.

- **learning_rate**: il tasso di apprendimento del modello. Di default è 0.3. Questo parametro controlla quanto il modello impara ad ogni iterazione. Un valore più basso può portare a un modello più preciso ma richiede più iterazioni. Si è provato inizialmente il valore di default e progressivamente è stato abbassato fino a 0.01.

- **max_depth**: la profondità massima di ogni albero. Di default è 6. Più si aumenta questo valore, più il modello diventa complesso e rischia di andare in overfitting. Si è deciso di usare il valore di default.

- **min_child_weight**: il peso minimo di un nodo figlio. Di default è 1. Questo parametro controlla la quantità minima di campioni richiesti per creare un nodo figlio. Un valore più alto può portare a un modello più semplice e meno suscettibile all'overfitting ma se troppo alto esso può portare ad underfitting. Si è deciso di usare 3.

- **subsample**: la frazione di campioni da usare per addestrare ogni albero che deve essere compresa tra 0 e 1. Di default è 1 si è scelto di usare 0.85.

- **colsample_bytree**: la frazione di feature da usare per addestrare ogni albero che deve essere compresa tra 0 e 1. Di default è 1. Si è deciso di usare 0.85 anche qui.

- **gamma**: il valore minimo di perdita richiesta per fare una divisione. Di default è 0 e si è deciso di lasciarlo così.

- **reg_alpha**: il termine di regolarizzazione L1 da applicare. più alto è il valore, più il modello sarà semplice e meno suscettibile all'overfitting. Di default è 0 e si è deciso di usare 0.1.

- **reg_lambda**: il termine di regolarizzazione L2 da applicare. Come per reg_alpha, più alto è il valore, più il modello sarà semplice. Di default è 1 e si è lasciato così.

- **random_state**: il seed per la generazione casuale dei dati.

- **eta**: il tasso di apprendimento del modello. Di default è 0.3. Dopo vari test non si è notato alcun miglioramento perciò si è deciso di usare il valore di default.

Inizialmente si è addestrato il modello con i setting base, impiegando circa 7 minuti. Successivamente avendo notato che il modello poteva essere addestrato anche con la GPU e supportando la parallelizzazione, si è deciso di usarla e i tempi di addestramento sono scesi drasticamente, impiegando all'incirca 1 minuto per un addestramento completo. I risultati sono leggermente migliori rispetto al modello Random Forest con un accuretezza di quasi il 93%. L'F1-Score, come per il Random Forest, risulta maggiore per i domini legit, con uno 0.92 per i domini DGA e 0.93 per i domini legittimi, come si può vedere nella fig. 3.5.

	precision	recall	f1-score	support
Legit	0.91620	0.94542	0.93058	159595
DGA	0.94275	0.91224	0.92724	157245
accuracy			0.92895	316840
macro avg	0.92948	0.92883	0.92891	316840
weighted avg	0.92938	0.92895	0.92892	316840

Figura 3.5: Risultati del modello XGBoost

Dalla Confusion Matrix 3.6 si può vedere che il modello ha classificato il 3% in più di false negative rispetto ai false positive.

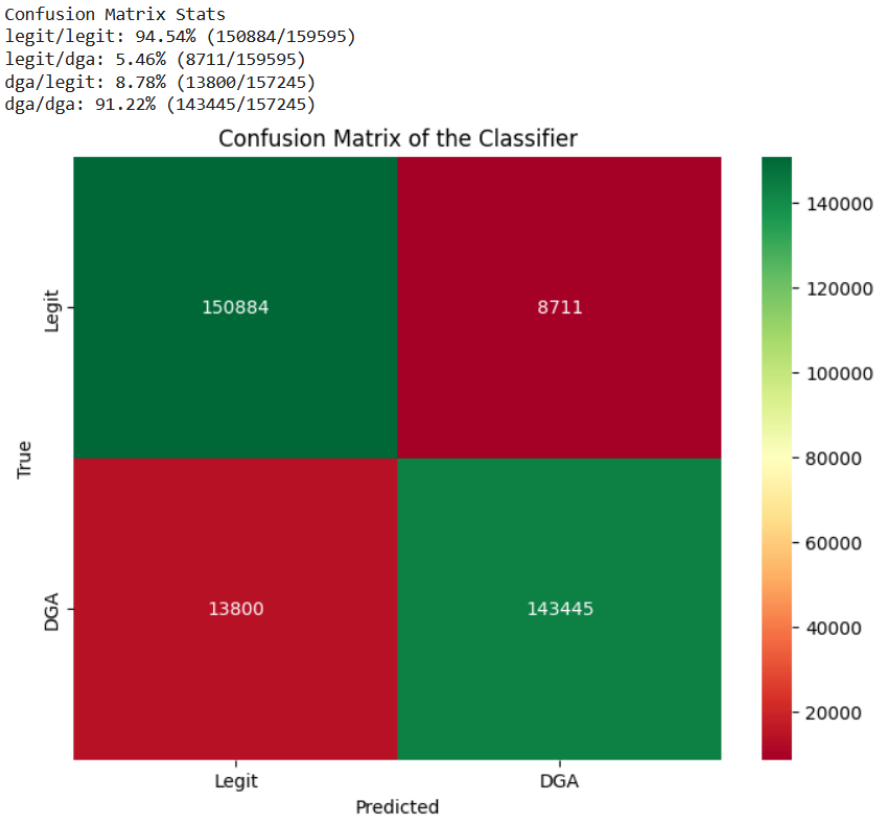

Figura 3.6: Confusion Matrix del modello XGBoost

3.5 Test con LSTM

Poiché LSTM è adatto anche per il riconoscimento di pattern ho deciso di usarlo per il progetto sia nella sua forma classica che in quella bidirezionale. A differenza dei modelli di Machine Learning precedenti, le LSTM non hanno bisogno delle features poiché esse imparano direttamente dai dati. Il modello però non è in grado di leggere direttamente i domini. Perciò bisogna convertire i domini in un formato leggibile dalla rete neurale e questo viene svolto tramite un processo chiamato **Tokenization**. Questo converte i dati in un formato leggibile dalla rete neurale. Ci sono vari metodi di tokenization come word tokenization, che divide le stringhe in più parole o la character tokenization che divide le stringhe in singoli caratteri. Essendo i domini delle semplici stringhe, spesso senza parole di senso compiuto e con caratteri speciali come il punto o il trattino, per questo progetto si è deciso di usare la character tokenization. Per fare ciò si è usato il metodo `Tokenizer` di Keras. Per questo modello si è deciso di usare un modello sequenziale ovvero un

modello in cui gli strati sono disposti in sequenza uno dopo l'altro.
Il primo strato è uno strato di embedding che serve per convertire i caratteri in vettori numerici poiché come detto in precedenza, le reti neurali possono leggere solo numeri.
Successivamente nel modello è presente uno strato di LSTM con 128 neuroni. Il numero di neuroni non è un valore fisso e non è determinato da una regola precisa. Esso può essere scelto in base alla complessità del problema o alla quantità di dati presenti nel dataset. In generale con troppi neuroni si rischia di andare in overfitting e con pochi il contrario. Dopo vari tentativi, 128 neuroni è sembrata la scelta migliore poiché con più neuroni il modello non imparava, rimanendo bloccato con un accuracy di 0.5(50%) circa.
Dopo di che è stato aggiunto uno strato di dropout ovvero uno strato che disabilita casualmente un certo numero di neuroni durante l'addestramento per evitare l'overfitting. Anche il rate di dropout non è un valore fisso. Dopo è presente un altro strato LSTM come il primo e un dropout e uno strato con attivazione relu con 64 neuroni che serve a catturare pattern più complessi. Inizialmente si era provato senza questo strato ma il modello aveva una precisione inferiore di circa il 5%. Infine l'ultimo è uno strato denso con attivazione sigmoide che serve a classificare i dati in due classi, DGA e non DGA. Per il training sono state scelte inizialmente 5 e poi 10 epoch. Il modello però non riusciva a generalizzare, quindi si è deciso di aumentare il numero di epoche a 50 e di usare l'early stopping, un metodo per interrompere l'addestramento quando il modello non migliora più e per evitare l'overfitting. Alla fine l'addestramento si è fermato all'epoch 26 impiegando circa 30 minuti e poco più di 1 minuto per epoca. Il modello raggiunge il 99% di accuracy e un alto F1-Score nel riconoscimento di entrambi i tipi di domini, come si può vedere nella fig. 3.7.

	precision	recall	f1-score	support
Legit	0.99222	0.99490	0.99356	159595
DGA	0.99481	0.99208	0.99344	157245
accuracy			0.99350	316840
macro avg	0.99351	0.99349	0.99350	316840
weighted avg	0.99350	0.99350	0.99350	316840

Figura 3.7: Risultati del modello LSTM

Dalla confusion matrix 3.8 si può vedere che il modello ha classificato quasi tutti i domini correttamente. Leggermente di più sono i False Negative ovvero domini

DGA classificati come legittimi.

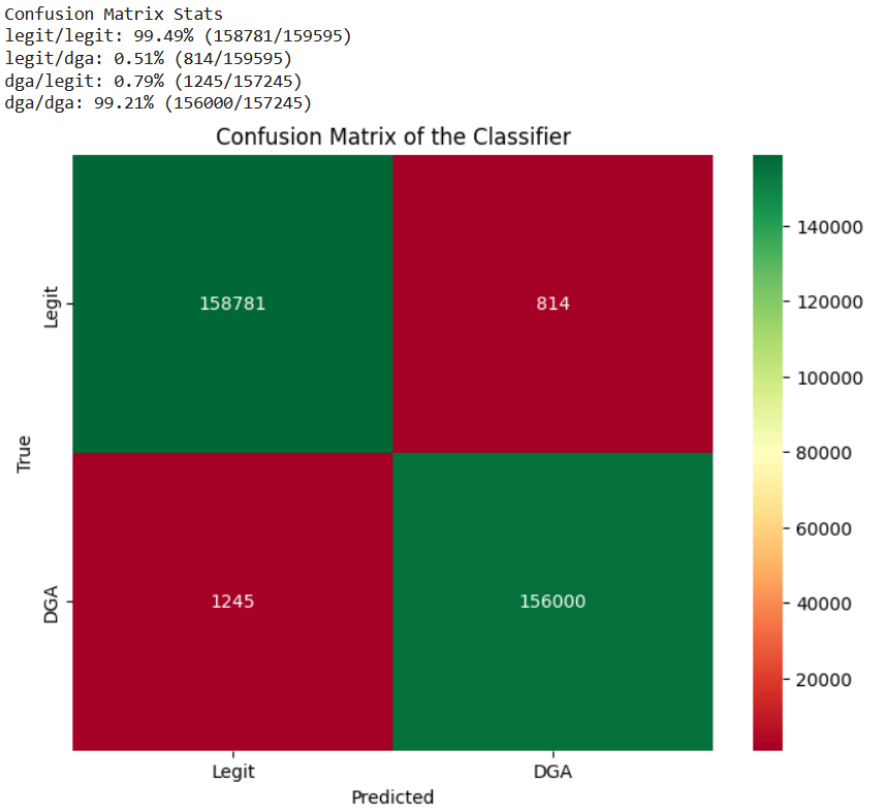

Figura 3.8: Confusion Matrix del modello LSTM

3.6 Test con BLSTM

Come detto in precedenza, le BLSTM sono una variante delle LSTM che consentono di elaborare i dati in entrambe le direzioni. Questo permette al modello di catturare meglio i pattern nei dati. Come nelle LSTM aumentare o dimunire il numero di neuroni non ha avuto effetti positivi sul modello perciò si è deciso di usare anche qui 128 neuroni per ogni strato BLSTM e uno denso con attivazione relu con 64 neuroni. Le epoche scelte sono anche qui 50 con early stopping. Le tempistiche di addestramento, però, sono risultate maggiori rispetto alle LSTM classiche, impiegando circa 4 minuti ad epoca e circa 1 ora e 10 minuti per 22 epochs totali. I risultati sono stati molto simili a quelli delle LSTM come si può vedere nella fig. 3.9.

	precision	recall	f1-score	support
Legit	0.99230	0.99450	0.99340	159595
DGA	0.99440	0.99217	0.99329	157245
accuracy			0.99334	316840
macro avg	0.99335	0.99334	0.99334	316840
weighted avg	0.99335	0.99334	0.99334	316840

Figura 3.9: Risultati del modello BLSTM

Per quanto riguarda la confusion matrix 3.10, il modello ha classificato quasi tutti i domini correttamente. Il modello risulta più preciso anche se di poco rispetto alle LSTM classiche. La maggior parte degli errori sono False Negative anche qui

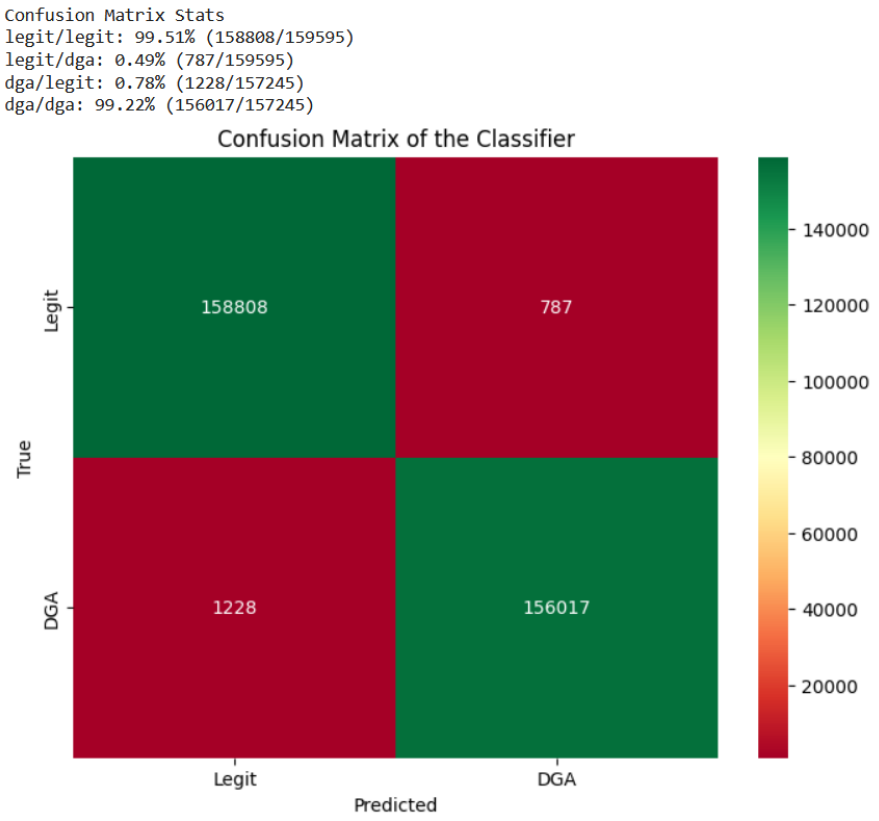

Figura 3.10: Confusion Matrix del modello BLSTM

Capitolo 4

Conclusioni

In questo progetto...

Bibliografia

[1] R. Sivaguru, C. Choudhary, B. Yu, V. Tymchenko, A. Nascimento, and M. D. Cock, "An evaluation of dga classifiers," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 5058–5067.

[2] B. Yu, J. Pan, J. Hu, A. Nascimento, and M. De Cock, "Character level based detection of dga domain names," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[3] G. Alley-Young, "Conficker worm," in *The Handbook of Homeland Security*. CRC Press, 2023, p. 175.

[4] domain.com, "Domain lifecycle," 2025, accessed: 20 May 2025. [Online]. Available: https://www.domain.com/help/article/domain-registration-the-life-cycle-of-a-domain

[5] Wikipedia contributors, "Domain name — Wikipedia, the free encyclopedia," 2025, [Online; accessed 11-June-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Domain_name&oldid=1294156712

[6] ——, "Domain name system — Wikipedia, the free encyclopedia," 2025, [Online; accessed 11-June-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Domain_Name_System&oldid=1292291287

[7] Treccani, "Machine learning," 2024. [Online]. Available: https://www.treccani.it/vocabolario/machine-learning_%28Neologismi%29/

[8] J. Namgung, S. Son, and Y.-S. Moon, "Efficient deep learning models for dga domain detection," *Security and Communication Networks*, vol. 2021, no. 1, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/8887881

[9] M. Eslahi, R. Salleh, and N. B. Anuar, "Bots and botnets: An overview of characteristics, detection and challenges," in *2012 IEEE International Conference on Control System, Computing and Engineering*, 2012, pp. 349–354.

[10] E. Ogu, N. Vrakas, C. Ogu, and A.-I. B.M., "On the internal workings of botnets: A review," *International Journal of Computer Applications*, vol. 138, pp. 975–8887, 04 2016.

[11] X. Ma, X. Guan, J. Tao, Q. Zheng, Y. Guo, L. Liu, and S. Zhao, "A novel irc botnet detection method based on packet size sequence," in *2010 IEEE International Conference on Communications*, 2010, pp. 1–5.

[12] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses," in *2009 Cybersecurity Applications and Technology Conference for Homeland Security*, 2009, pp. 299–304.

[13] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[14] J. Zou, Y. Han, and S.-S. So, "Overview of artificial neural networks," *Artificial neural networks: methods and applications*, pp. 14–22, 2009.

[15] T. O. Ayodele, "Types of machine learning algorithms," *New advances in machine learning*, vol. 3, no. 19-48, pp. 5–1, 2010.

[16] E. Grossi and M. Buscema, "Introduction to artificial neural networks," *European journal of gastroenterology & hepatology*, vol. 19, no. 12, pp. 1046–1054, 2007.

[17] W. Commons, "File:artificialneuronmodel english.png — wikimedia commons, the free media repository," 2024, [Online; accessed 13-maggio-2025]. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:ArtificialNeuronModel_english.png&oldid=840034703

[18] ——, "File:multilayerneuralnetwork english.png — wikimedia commons, the free media repository," 2020, [Online; accessed 13-May-2025]. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:MultiLayerNeuralNetwork_english.png&oldid=481159061

[19] M. Zakaria, A. Mabrouka, and S. Sarhan, "Artificial neural network: a brief overview," *neural networks*, vol. 1, p. 2, 2014.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[23] J. Brownlee, "What is the difference between a batch and an epoch in a neural network," *Machine learning mastery*, vol. 20, no. 1, pp. 1–15, 2018.

[24] B. Hamza, "The comprehensive programming language model," Ph.D. dissertation, École Nationale Supérieure d'Informatique, 07 2021.

[25] "Decision tree classification in python," 2025, accessed: 14 May 2025. [Online]. Available: https://www.datacamp.com/tutorial/decision-tree-classification-python

[26] M. Y. Khan, A. Qayoom, M. Nizami, M. S. Siddiqui, S. Wasi, and K.-U.-R. R. Syed, "Automated prediction of good dictionary examples (gdex): A comprehensive experiment with distant supervision, machine learning, and word embedding-based deep learning techniques," *Complexity*, 09 2021.

[27] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, p. 160, Mar 2021. [Online]. Available: https://doi.org/10.1007/s42979-021-00592-x

[28] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[29] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[30] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[31] H. Shahzad, A. R. Sattar, and J. Skandaraniyam, "Dga domain detection using deep learning," in *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*, 2021, pp. 139–143.

[32] "What is lstm - long short term memory?" 2025, accessed: 28 May 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

[33] Y. Y. Research, "Dga domain names dataset," 2025, accessed: 28 May 2025. [Online]. Available: https://huggingface.co/datasets/YangYang-Research/dga-detection

[34] G. Sharma, "Confusion matrix: What is it and its applications," Medium, 2025, accessed: 30 May 2025. [Online]. Available: https://geetanshsharma2018.medium.com/confusion-matrix-what-is-it-and-its-applications-4d8b0b958edb

[35] Scikit-learn, "Random forests classifier," 2025, accessed: 30 May 2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Ringraziamenti

Con questo lavoro di tesi, si conclude il mio percorso di studi sicuramente non facile ma che mi ha dato la possibilità di crescere sia a livello personale che professionale.
Ringrazio il mio relatore, il Prof. Mirko Viroli per essere stato sempre disponibile e avermi aiutato con il lavoro e Lorenzo Magi e tutta Flashstart per l'aiuto dato e per avermi dato la possibilità di lavorare su questo progetto.
Ringrazio la mia famiglia per avermi sempre supportato sia moralmente che economicamente. Senza di loro non sarei mai riuscito a raggiungere questo traguardo.
Ringrazio i miei amici in particolare l'associazione Sprite che hanno reso questo percorso più divertente. Ringrazio i bloccati e gli amici di Smash Bros per i momenti di divertimento e avermi dato un hobby per staccare dallo studio.