

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

Sommario

Max 2000 characters, strict.

Indice

Sommario	iii
1 Introduzione	1
2 Background	3
2.1 Domain Name System	3
2.2 Botnets e DGA	3
2.2.1 Botnets	3
2.2.2 Command and Control	4
2.2.3 Domain Generation Algorithm	5
2.3 Machine Learning	7
2.3.1 Reti Neurali	7
2.3.2 Sviluppo di una rete neurale	9
2.4 Algoritmi di Machine Learning adatti per DGA	10
2.4.1 Random Forest	10
2.4.2 Long Short Term Memory (LSTM)	12
3 Progetto	15
3.1 Linguaggi e librerie usate	15
3.2 Metriche di valutazione	16
3.3 Test con Random Forest	17
3.4 Test con LSTM	20
3.5 Test con BLSTM	22
3.6 Some cool topic	23
4 Contribution	25
4.1 Fancy formulas here	25
	27
Bibliografia	27

Elenco delle figure

2.1	Ciclo di vita di un botnet [6]	4
2.2	Esempio di server C&C. (a) centralizzato, (b) decentralizzato [5] . .	5
2.3	Esempio del funzionamento di un DGA [4]	6
2.4	Esempio di neurone artificiale [13]	7
2.5	Esempio di rete neurale [14]	8
2.6	Pipeline di un modello di Machine Learning [20]	10
2.7	Esempio di albero decisionale [21]	11
2.8	Struttura di un LSTM [24]	13
3.1	Esempio di Confusion Matrix [26]	17
3.2	Parte del dataset con i domini e loro features	18
3.3	Risultati del modello Random Forest	19
3.4	Confusion Matrix del modello Random Forest	19
3.5	Risultati del modello LSTM	21
3.6	Confusion Matrix del modello LSTM	21
3.7	Risultati del modello BLSTM	22
3.8	Confusion Matrix del modello BLSTM	23

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 dei Command and Control (C&C) servers. I C&C sono dei server che manipolano i computer infetti da malware, i 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 brutto 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 per nascondere i loro server chiamati Domain Generation Algorithm (DGA) servers. Negli ultimi anni, sono stati sviluppati vari metodi di Machine Learning per rilevare questi domini.

Struttura della tesi Il lavoro di tesi si dividerà in 4 capitoli. Nel primo capitolo verranno descritti concetti di base riguardanti DGA e Botnets, e Machine Learning con una breve descrizione delle reti neurali e di alcuni algoritmi usati. 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 Domain Name System

Il **Domain Name System** (DNS) è un sistema che traduce un nome di dominio in un indirizzo IP.

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 un attaccante, il 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'attaccante 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 C&C per far capire all'attaccante 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

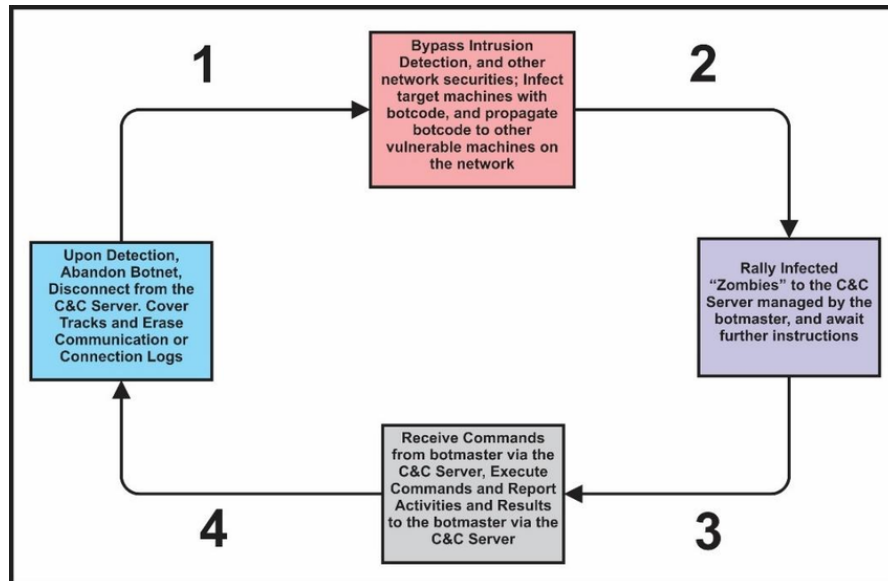


Figura 2.1: Ciclo di vita di un botnet [6]

2.2.2 Command and Control

Il meccanismo del 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:
 - **IRC** : Internet Relay Chat (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 contattare 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 C&C. Questo metodo è più difficile da rilevare ma anche più complesso da implementare [8].

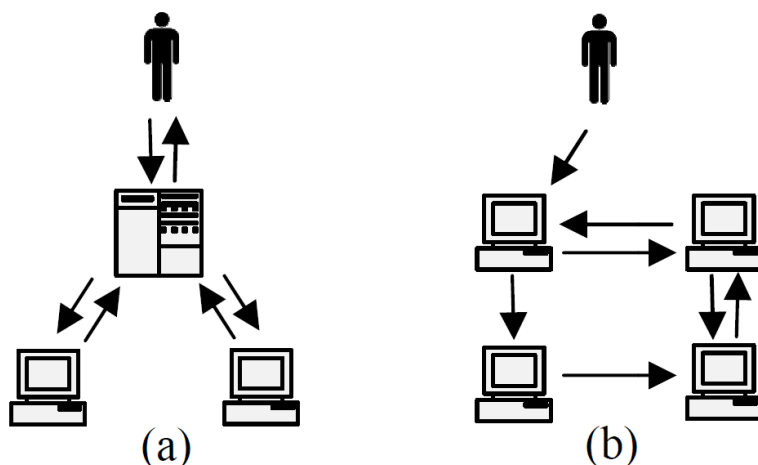


Figura 2.2: Esempio di server C&C. (a) centralizzato, (b) decentralizzato [5]

2.2.3 Domain Generation Algorithm

I DGA sono algoritmi che generano migliaia di domini in modo pseudo casuale. Prima viene scelto un seed, di solito 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 non riesce a contattarlo 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 [4].

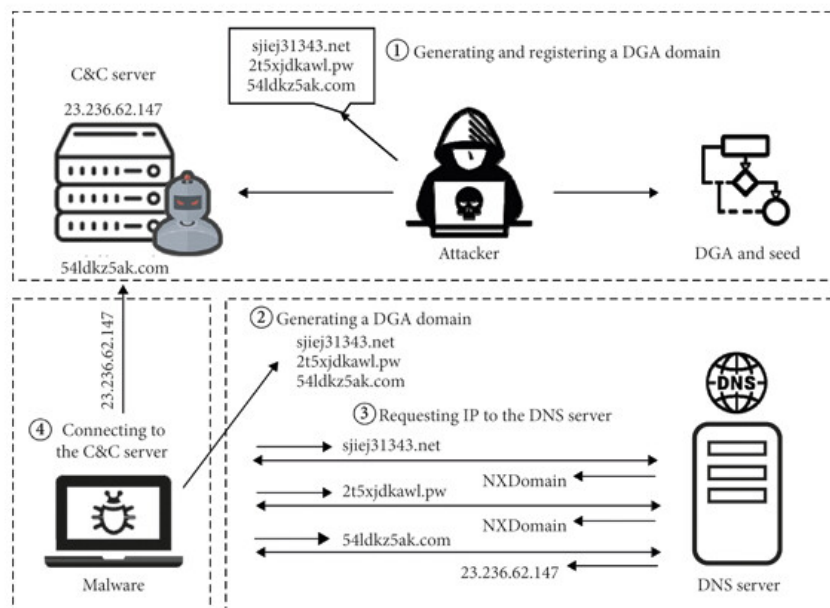


Figura 2.3: Esempio del funzionamento di un DGA [4]

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

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

2.3 Machine Learning

Il Machine Learning è una branca dell'informatica che punta a far ragionare le macchine come gli esseri umani, ovvero a svolgere compiti autonomamente senza essere programmati esplicitamente e migliorando le loro prestazioni con l'esperienza e i dati.

2.3.1 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 [10]. 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 (weight 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 in input prende 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.

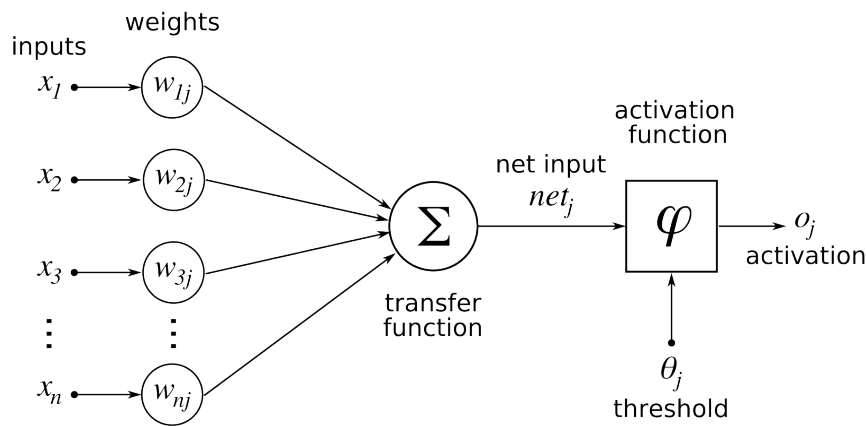


Figura 2.4: Esempio di neurone artificiale [13]

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, come nel nostro caso.
- **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 non DGA)

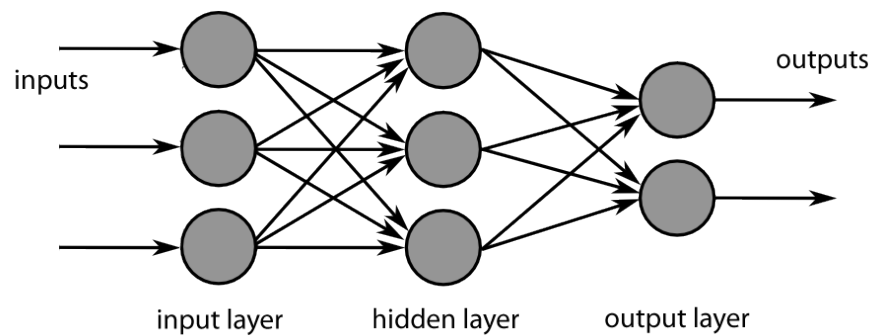


Figura 2.5: Esempio di rete neurale [14]

Abbiamo inoltre vari tipi di reti neurali:

- **Reti Neurali Feedforward:** 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 principalmente per riconoscimento di pattern o classificazioni
- **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 soprattutto nell'ambito della visione artificiale per il riconoscimento di oggetti [16].

2.3.2 Sviluppo di una rete neurale

Il primo passo per sviluppare una rete neurale 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 un etichetta, DGA o legit nel nostro caso, o no. Questi come detto in precedenza, dovranno essere convertiti in un vettore numerico. A seconda di come è strutturato il dataset possiamo avere vari tipi di allenamento:

- **Supervised Learning:** È la tecnica più comune per allenare le reti neurali [11]. 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. 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 70% training set, 15% validation set e 15% test set. Successivamente viene deciso il tipo di rete neurale da usare, il numero di layers e il numero di neuroni per ognuno di essi. Una rete troppo semplice potrebbe non essere in grado di apprendere i pattern nei dati (**Underfitting**), mentre al contrario, una rete troppo complessa potrebbe imparare troppo bene di dati e non generalizzare su dati nuovi (**Overfitting**). Come detto in precedenza, il validation set ci aiuta a capire se il modello va

in overfitting o underfitting. Successivamente vengono inizializzati i pesi di ogni neurone. Questi possono essere inizializzati in modo casuale o con altre tecniche più avanzate ad esempio la **He Initialization** [18] (opzione di default quando si inizializza con attivazione ReLU). Dopo di che si passa alla fase vera e propria di addestramento. Il modello viene iterato per un certo numero di epoche su un batch di dati. 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. Il numero di epoche è un iperparametro che indica quante volte l'algoritmo di apprendimento deve passare attraverso il dataset di training[19]. Infine viene testato il modello sul test set e, se la precisione è soddisfacente, il modello potrà essere usato.

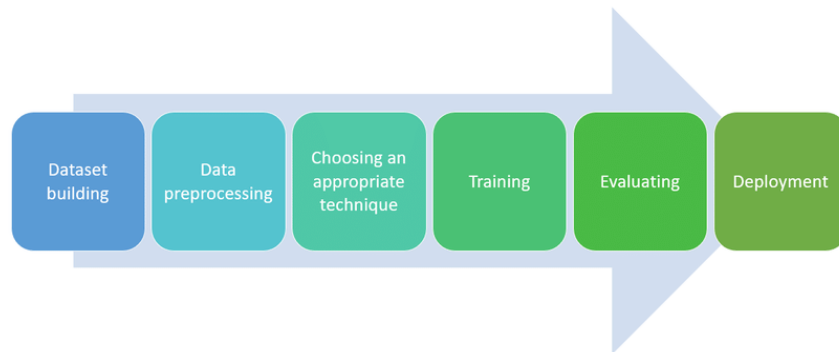


Figura 2.6: Pipeline di un modello di Machine Learning [20]

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 ensemble (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. Il random forest risolve questo problema aumentando l'accuratezza del modello sia in fase di training che in fase di testing. [22] Random Forest usa un approccio 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. L'algoritmo risulta tra i più usati per la sua semplicità e velocità e per la sua capacità di gestire grandi quantità di dati e feature senza perdere accuratezza.

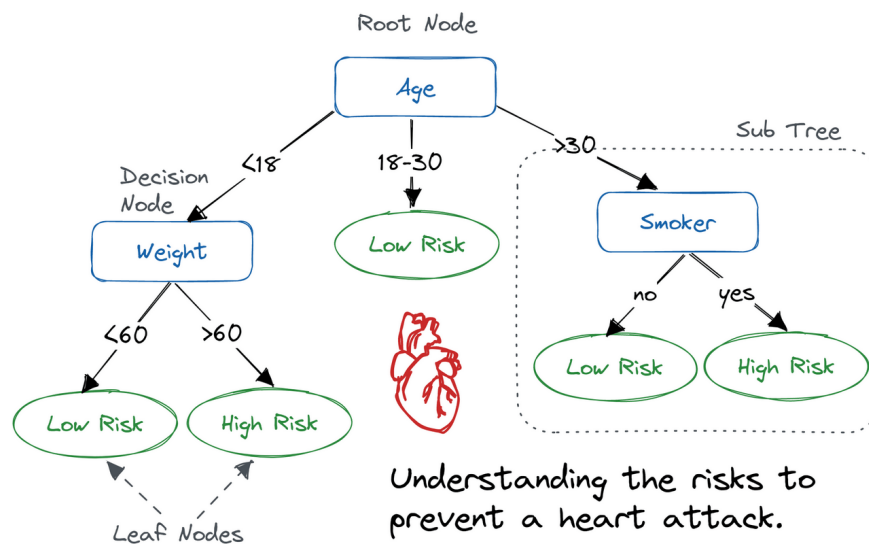


Figura 2.7: Esempio di albero decisionale [21]

2.4.2 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 [17], 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 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, le LSTM usano un approccio featureless e imparano direttamente dai dati. Alcuni usi delle LSTM sono ad esempio il riconoscimento vocale o la traduzione automatica.

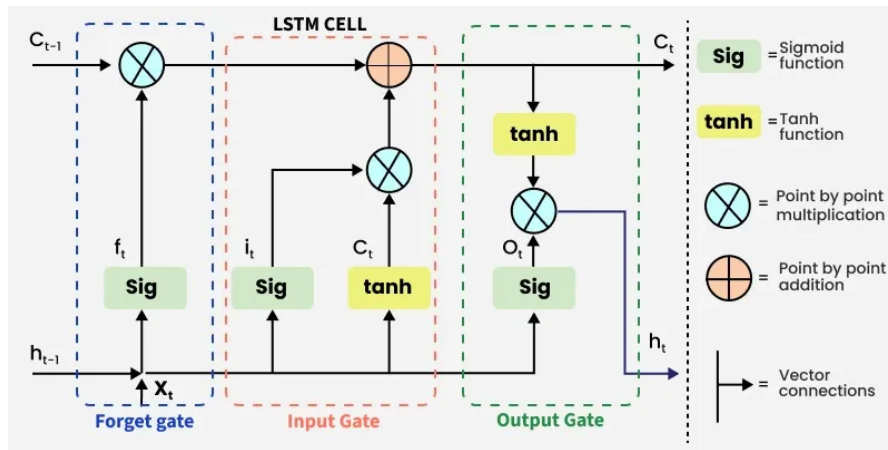


Figura 2.8: Struttura di un LSTM [24]

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 [25] 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.
- **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.
- **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, Precision, Recall, F1-Score e Confusion Matrix.

Nel nostro modello un DGA ha come label 1(positivo) mentre un dominio legittimo ha come label 0 (negativo). L'**Accuracy** è il rapporto del numero di predizioni corrette sul numero totale di predizioni effettuate. La formula è la seguente:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.1)$$

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:

$$Precision = \frac{TP}{TP + FP} \quad (3.2)$$

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

$$Recall = \frac{TP}{TP + FN} \quad (3.3)$$

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

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (3.4)$$

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.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Figura 3.1: Esempio di Confusion Matrix [26]

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 ma a parte un maggiore tempo di addestramento, non si è notato un miglioramento significativo.
- **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.

3.3. TEST CON RANDOM FOREST

- **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`, gli altri parametri sono stati lasciati con i valori di default poiché, dopo vari test non sono stati riscontrati miglioramenti. Per le features, si è notato che più sono le features, migliore è il risultato del modello. 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:

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

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 chiamato `feature_importances_` che permette di calcolare l'importanza di ogni feature nel modello. 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	nyic.org	0	8	3.000000	5	0.625000	2	0.250000	0	0	2.500000	3	0
1	heagols.com	0	12	3.251629	7	0.583333	4	0.333333	0	0	1.750000	2	0
2	comio-doujn.com	0	16	3.155639	8	0.500000	6	0.375000	0	1	1.333333	2	0
3	yleffananshan.cc	1	16	2.952820	11	0.687500	4	0.250000	0	0	2.750000	4	0
4	simplicity.com	0	15	3.323231	9	0.600000	5	0.333333	0	0	1.800000	2	0
5	1g1wuq15geug4nv4gtr1uzps2.net	1	31	4.026619	17	0.548387	5	0.161290	8	0	3.400000	4	5
6	l-fa.biz	0	8	2.750000	3	0.375000	3	0.375000	0	1	1.000000	2	0
7	xfidtesthssbiophysicalohax.com	1	30	4.015061	20	0.666667	9	0.300000	0	0	2.222222	5	0
8	caqhtkzupjejsa.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

Dopo vari test si è raggiunta un accuratezza del 91% ma con risultati peggiori nel riconoscimento dei DGA come si può vedere nella fig. 3.3.

3.3. TEST CON RANDOM FOREST

	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.

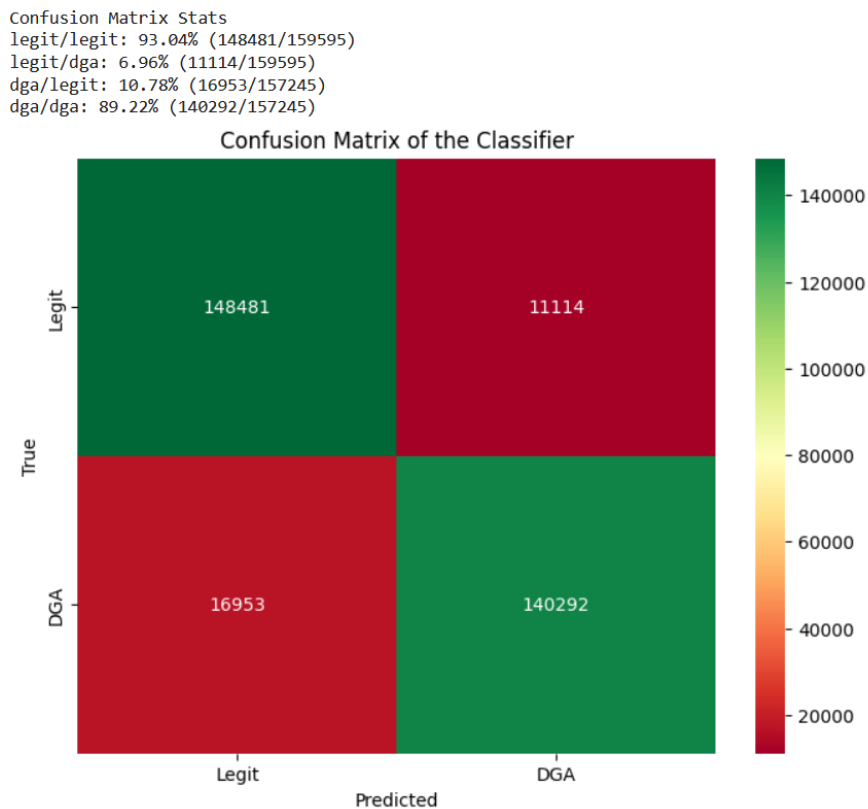


Figura 3.4: Confusion Matrix del modello Random Forest

3.4 Test con LSTM

Poiché LSTM è un tipo di rete neurale adatto per il riconoscimento di pattern ho deciso di usarlo per il progetto sia nella sua forma classica che in quella bidirezionale. 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 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 anche se di poco. 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 andava in overfitting, 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 il modello 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

3.4. TEST CON LSTM

domini, come si può vedere nella fig. 3.5.

	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 3.5: Risultati del modello LSTM

Dalla confusion matrix 3.6 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.

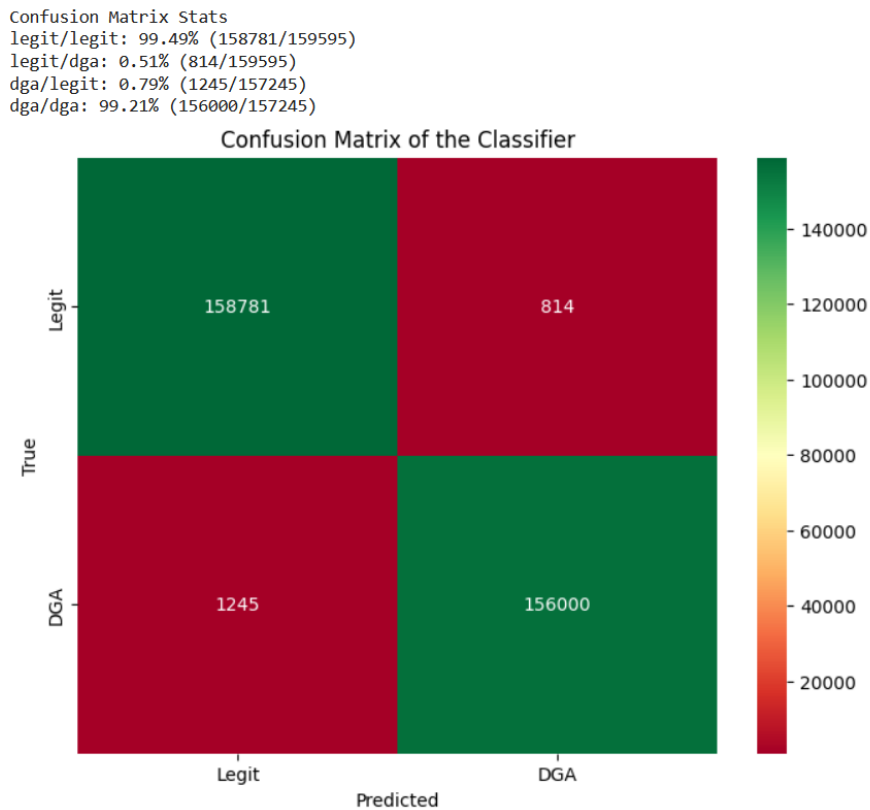


Figura 3.6: Confusion Matrix del modello LSTM

3.5 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 diminuire 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 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. 3.7.

	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 3.7: Risultati del modello BLSTM

Per quanto riguarda la confusion matrix 3.8, 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

3.6. SOME COOL TOPIC

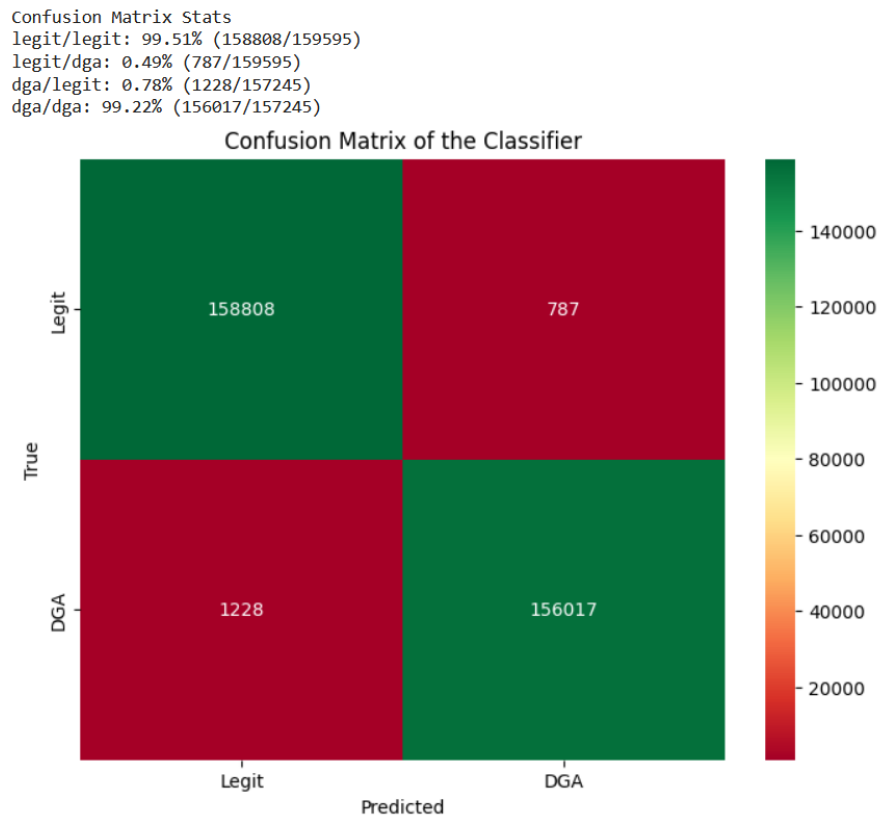


Figura 3.8: Confusion Matrix del modello BLSTM

3.6 Some cool topic

Capitolo 4

Contribution

You may also put some code snippet (which is NOT float by default), eg: capitolo 4.

4.1 Fancy formulas here

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         // Prints "Hello, World" to the terminal window.
4         System.out.println("Hello, World");
5     }
6 }
```

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] 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>
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.

- [9] 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.
- [10] J. Zou, Y. Han, and S.-S. So, “Overview of artificial neural networks,” *Artificial neural networks: methods and applications*, pp. 14–22, 2009.
- [11] T. O. Ayodele, “Types of machine learning algorithms,” *New advances in machine learning*, vol. 3, no. 19-48, pp. 5–1, 2010.
- [12] E. Grossi and M. Buscema, “Introduction to artificial neural networks,” *European journal of gastroenterology & hepatology*, vol. 19, no. 12, pp. 1046–1054, 2007.
- [13] 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
- [14] —, “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
- [15] M. Zakaria, A. Mabrouka, and S. Sarhan, “Artificial neural network: a brief overview,” *neural networks*, vol. 1, p. 2, 2014.
- [16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
- [17] 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>
- [18] 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.
- [19] 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.
- [20] B. Hamza, “The comprehensive programming language model,” Ph.D. dissertation, École Nationale Supérieure d’Informatique, 07 2021.

- [21] “Decision tree classification in python,” 2025, accessed: 14 May 2025. [Online]. Available: <https://www.datacamp.com/tutorial/decision-tree-classification-python>
- [22] 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.
- [23] 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.
- [24] “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/>
- [25] Kaggle, “Dga domain names dataset,” 2025, accessed: 28 May 2025. [Online]. Available: <https://huggingface.co/datasets/YangYang-Research/dga-detection>
- [26] 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>
- [27] 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.