Università Politecnica delle Marche

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Confronto tra classificatori ensemble per il rilevamento di malware basati su DGA

Comparison of ensemble classifiers for DGA-based malware detection

Relatore

Prof. Luca Spalazzi

Candidato

Lorenzo Sopranzetti

Correlatori

Prof. Alessandro Cucchiarelli

Prof. Christian Morbidoni

Anno Accademico 2020-2021

Indice

Elenco delle figure

1

Introduzione

Sin dalla nascita del World Wide Web la sicurezza informatica è stato uno dei temi centrali e più importanti sui quali si è investito di più.

Negli ultimi anni il WWW è sempre di più il mezzo utilizzato per registrare informazioni personali di ogni individuo andando dai conti bancari, agli acquisti, agli interessi personali.

La crescente influenza che Internet ha sulla vita di ognuno lo ha portato ad essere un parte fondamentale e quasi indispensabile della quotidianità di molte persone.

Con l'esplosione della pandemia del covid-19 questa dipendenza è stata ulteriormente esasperata con interi sistemi divenuti completamente dipendenti dalla rete, basti pensare all'apparato scolastico.

Mantenere il WWW e Internet luoghi sicuri è dunque sempre più importante. Questo lavoro si propone di fare un passo in avanti in questa direzione analizzando una delle armi emergenti nella pirateria informatica: le botnet e i DGA (domain generation algorithm).

L'obiettivo di questa tesi è applicare delle tecniche di machine learning per rilevare e classificare nomi di dominio generati dai DGA. Tra le varie tecniche di machine learning si è scelto un percorso che sfrutta un approccio supervisionato e featureless basato sulle Reti Neurali Profonde o Deep Learning. In particolare, nel contesto del deep learning, si fa uso dei "Sistemi Multi-classificatori" o "Classificatori Ensamble" al fine di confrontarli per individuare quali offrono delle performance migliori.

Nel primo capitolo verrà presentato cosa sono una botnet e un DGA quali sono i problemi da affrontare per combatterli e come il machine learning può essere un arma per questo scopo. Sempre nella prima sezione verrà dunque spiegato cos'è il machine learning e quali tecnologie in quest'ambito sono state utilizzate: deep learning, reti neurali, reti LSTM.

Nel secondo capitolo sono presentati gli strumenti per gli esperimenti: qual è la struttura delle reti neurali utilizzate, cos'è una sistema multi-classificatore (MCS) e quali MCS sono stati creati descrivendoli nel dettaglio.

Nel terzo capitolo vengono, nella prima parte, presentati gli esperimenti effettuati e i risultati ottenuti spiegando il dataset e le metriche di valutazione utilizzate. Nella seconda parte del capitolo questi risultati vengono poi confrontati e discussi per valutare i sistemi MCS costruiti.

Nel quarto e ultimo capitolo verranno tratte delle conclusioni discutendo gli obiettivi raggiunti, quelli non raggiunti e indicando possibili sviluppi futuri.

2

Botnet, DGA e Machine Learning

2.1 I malware

Con il termine malware si indica un qualsiasi programma utilizzato per disturbare le operazioni svolte da un utente di un computer [1]. Questa definizione, che risulta essere evidentemente generica, nasconde in realtà una grande diversità, infatti esistono molti malware diversi, con diverse caratteristiche e creati per scopi diversi. La particolare tipologia di malware che viene qui affrontata, diventata sempre più importante e famosa negli ultimi anni, è quella utilizzata per la costruzione di Botnet.

2.2 Botnet

Le Botnet sono definite come una rete di computer, tra loro comunicanti, controllata da un "Botmaster" e composta da dispositivi infettati da malware specializzato denominati "bot" o "zombie". In realtà, all'inizio della loro storia, le Botnet nacquero come semplici reti di calcolatori per il calcolo condiviso o per il mantenimento di servizi web. Acquisirono poi un significato negativo quando divennero un mezzo utilizzato dai pirati informatici per eseguire attacchi di diverso tipo come:

- *DDos:* attacco massivo contro uno specifico bersaglio con lo scopo interrompere un particolare servizio.
- *Spam:* compagne di spam di diverso tipo, solitamente per la promozione di prodotti.
- *Phishing:* campagne di spam con lo scopo di rubare informazioni di diverso tipo.
- *Spyware:* i malware nei computer infettati possono inviare informazioni sull'attività online della vittima.

Molto spesso servizi e attacchi vengono venduti dai Botmaster a clienti terzi garantendo guadagni importanti.

[1] Wikipedia: https://it.wikipedia.org/wiki/Malware

2.2.1 Caratteristiche principali

Le principali caratteristiche di una Botnet sono[5]:

- **Sono una rete di macchine compromesse:** le Botnet sono una rete di macchine infette che possono comunicare tra di loro o verso un elemento centrale.
- **Possono essere coordinate da remoto:** le Botnet hanno la capacità di ricevere e eseguire comandi inviati da un Botmaster da remoto agendo in maniera coordinata. Questa è la caratteristica principale che distingue le Botnet da altri tipi di malware.
- **Usate per scopi fraudolenti:** come già scritto, le Botnet nascono principalmente per svolgere attività illegali e malevole.

2.2.2 Struttura

La struttura di una Botnet è semplice e si compone dei seguenti elementi:

- **Bot o Zombie**: singoli calcolatori infettati da un malware che fanno parte della Botnet. Questo malware prende il nome di "bot agent" e il suo principale scopo è di comunicare con tutte le componenti della Botnet.
- **Botmaster**: è una persona che comanda e controlla la Botnet inviando comandi da remoto.
- **Command and Control Server (C&C)** è il nodo principale della rete dal quale il Botmaster invia comandi a tutti gli altri componenti. Questo è l'elemento più importante della rete, senza di esso la Botnet degenera in un gruppo non coordinato di macchine infette divenendo inutile.

2.2.3 Protocolli di Comando e Controllo della rete

Vengono utilizzati diversi protocolli per comandare e controllare le Botnet [15]:

- **Telnet**: utilizzando un semplice protocollo i bot si connettono a una unità di comando principale che ospita la botnet. I bot sono inseriti nella rete utilizzando un script, eseguito su un server esterno, che scansiona diversi indirizzi IP cercando un server che utilizza i protocolli telnet o SSH con credenziali di accesso di default. Quando si riesce con successo ad accedere a un server, quest'ultimo viene infettato dal server esterno via SSH con un malware, il quale segnalerà la propria presenza al server C&C principale. Quando viene eseguito il comando ssh, il server viene infettato e diventa un componente della Botnet tramite il codice malevolo che l'ha infettato. A questo punto il Botmaster, attraverso il server di comando principale, è in grado di controllare i bot.
- **IRC**: i canali di comunicazione IRC rappresentano una vecchia forma di chat online. Le reti IRC utilizzano semplici metodi di comunicazione a larghezza di banda ridotta, che li rendono validi per ospitare una Botnet. Essendo relativamente semplici nella costruzione sono state usate, con moderato successo, per coordinare attacchi DDoS oppure per inviare migliaia di mail di spam sulla rete. Il Botmaster può impartire i comandi attraverso la chatroom, riuscendo ad allertare velocemente tutti gli zombi connessi. Un problema con questo modello

è la necessità che tutti i bot conoscano l'IRC server, la porta, e il canale al quale connettersi. Le organizzazioni anti-malware possono facilmente chiudere questi server o canali per rendere la Botnet inutile.

- **P2P**: in questo modello i nodi della rete non sono gerarchizzati secondo il modello client-server ma ognuno di essi può assumere entrambi i ruoli. In questa struttura dunque, tutti i bot sono connessi tra loro e possono sia inviare che ricevere istruzioni permettendo la costruzione di una struttura decentralizzata.
- **HTTP**: molte botnet sfruttano la connessione http verso un server con uno specifico nome di dominio per la loro costruzione. Un computer zombie accede a una pagina web o a un dominio appositamente progettati per restituire l'elenco dei comandi di controllo. L'uso di pagine web o domini come C&C consente un controllo e un mantenimento della Botnet con un semplice codice e può essere facilmente aggiornato. Il protocollo http è anche in grado di evitare nella maggior parte dei casi il controllo da parte degli anti-virus. Tuttavia, l'utilizzo di questo metodo necessita di una notevole larghezza di banda, inoltre, senza alcun meccanismo di protezione, i nomi di dominio possono essere facilmente sequestrati da agenzie governative.

2.2.4 Architettura

A partire dai protocolli di comando e controllo sopra elencati si possono costruire delle botnet con diversi tipi di architetture[8].

Architettura centralizzata

All'interno della rete è presente un unico dispositivo centralizzato che funge da server di controllo C&C dal quale tutti i componenti della rete prendono comandi. Al Botmaster è sufficiente collegarsi a questo server per mandare comandi a tutti i bot della rete. I protocolli più spesso utilizzati sono quello HTTP e IRC.

Questa architettura è molto semplice e di facile implementazione, tuttavia la chiusura del C&C porterebbe alla distruzione dell'intera Botnet.

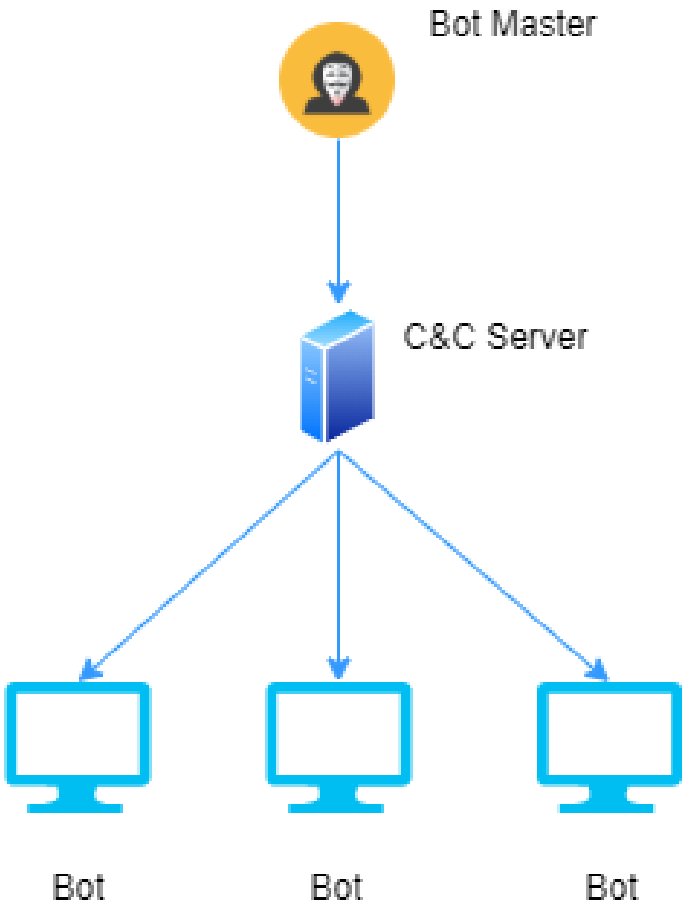

Figura 2.1. Esempi di architettura centralizzata

Architettura Multi-server

Questa architettura estende logicamente quella centralizzata utilizzando più di un server C&C per fornire istruzioni alla rete. I server comunicano costantemente tra di loro e se uno di essi dovesse essere spento gli altri continuerebbero a gestire la rete rendendo più difficile la chiusura dell'intera Botnet.

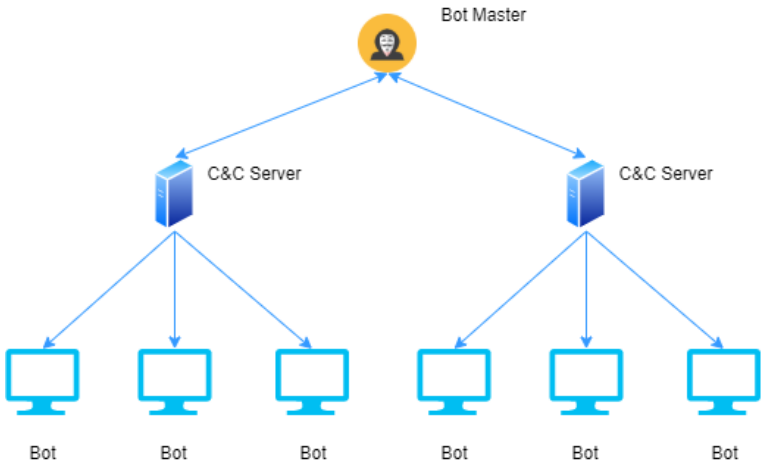

Figura 2.2. Esempio di architettura multiserver [21]

Struttura gerarchica

Una struttura di questo tipo riflette perfettamente le dinamiche della propagazione della botnet. Questa configurazione è difficile da utilizzare in quanto soffre di problemi di latenza dovuta alla propagazione dei comandi nella gerarchia.

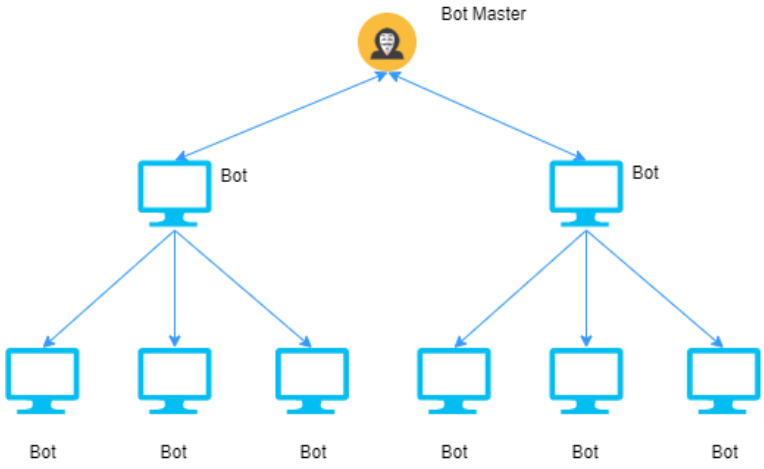

Figura 2.3. Esempio di architettura gerarchica

Struttura Peer to Peer

Ogni bot funge anche da Server C&C, dunque può inviare e ricevere comandi a e da tutti gli altri elementi della rete. In questo modo al Botmaster è sufficiente connettersi ad uno qualunque di questi elementi per poter fornire istruzioni.

Reti di questo tipo sono di difficile mantenimento e costruzione ma garantiscono grande resistenza e sicurezza per il Botmaster.

Le P2P Botnet possono essere divise in due tipi:

- **Parasite P2P botnet:** Tutti i bot si trovano all'interno di una rete P2P già esistente. In questo caso il bootstrap(processo di accesso ad una rete P2P) non è necessario dato che i bot sono già all'interno della rete.
- **Leeching P2P botnet:** I bot possono essere una qualsiasi computer infetto in Internet. I bot devono dunque seguire un processo di bootstrap per entrare a far parte di una rete P2P.

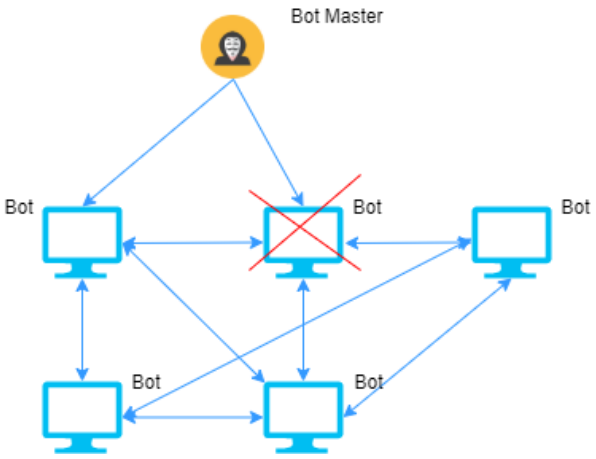

Figura 2.4. Esempio di architettura P2P

Architettura non strutturata

In questo modello ogni bot ha la possibilità di cercare in internet altri bot. Queste reti sono le più durevoli e sicure a causa della mancanza di una struttura centralizzata e delle molteplici rotte di comunicazione fra i vari elementi della botnet.

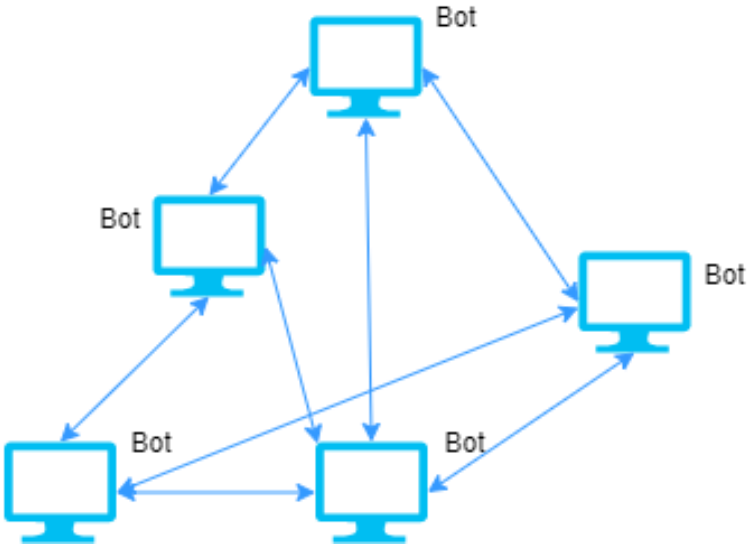

Figura 2.5. Esempio di architettura non strutturata

2.2.5 Life-Cycle

Una volta definite le caratteristiche strutturali e di base di una Botnet, ne viene ora analizzato il ciclo di vita che si compone di sei stadi[12]: Concezione, Reclutamento, Interazione, Marketing, Esecuzione dell'attacco, Conclusione dell'attacco.

Concezione

Dietro la fase di concezione di una Botnet si nascondono tre processi: *motivazione*, *design* e *implementazione*. La *motivazione* principale dietro la creazione di una Botnet è, solitamente, il profitto economico. Il *design* della Botnet può seguire una delle strutture prima elencate dipendentemente dalle capacità, risorse e volontà del Botmaster. L'ultima fase è quella dell'*implementazione* e codifica vera e propria della Botnet.

Reclutamento

Si effettua l'infezione dei sistemi vulnerabili, attraverso malware specializzati, con lo scopo di ampliare il più possibile la Botnet.

Interazione

In questa fase vengono comprese tutte le *interazioni interne* ed *esterne* alla rete che le Botnet effettuano durante il loro ciclo di vita. Le *interazioni interne* sono quelle che si effettuano tra i vari membri della Botnet (Botmaster e Bots). In questo caso i processi che entrano in gioco sono due:

- Registrazione:
  Processo grazie al quale un sistema diventa effettivamente parte della botnet. Esistono due tipi di registrazione: *statica* e *dinamica*.
  Nel primo caso tutte le informazioni (indirizzo IP del C&C server) per entrare a far parte della Botnet sono hardcoded, ovvero già presenti all'interno del codice del malware e solitamente mascherate in diversi modi per renderne più difficile l'individuazione.
  Nel secondo caso, i bot devono esplicitamente chiedere le informazioni necessarie per diventare parte della Botnet.

- Comunicazioni con il server C&C:
  Queste comunicazioni vengono raggruppate in base a due aspetti:
  - Direzione dell'informazione: i messaggi possono essere classificati come *pull* o *push*. Quando i bot richiedono informazioni esplicitamente parliamo di *pull*, quando invece ricevono informazioni in maniera passiva senza prima richiederle parliamo di *push*.
    Nel primo caso i bot periodicamente si connettono con il C&C controllando la presenza di nuovi comandi. Questa tecnica ha lo svantaggio di non fornire al Botmaster il controllo in real-time della Botnet. Nel secondo caso i bot sono connessi attivamente al C&C attendendo comandi, in questo modo si ha la possibilità di controllo real-time della rete.
  - Protocollo di Comunicazione: i protocolli di comunicazione maggiormente osservati sono IRC, HTTP, P2P.

Le *interazioni esterne* sono quelle tra un membro della botnet e un host non compromesso. Queste spesso corrispondono all'accesso a servizi Internet di diverso tipo: DNS, reti P2P.

Fase di marketing

Una volta costituita una solida botnet il Botmaster deve trovare un modo per trarre guadagno da essa. Le principali soluzioni per ottenere guadagno sono: vendere il codice della botnet, affittare il codice della botnet o fornire diversi servizi.

Esecuzione dell'attacco

Gli attacchi che possono essere portati avanti grazie ad una botnet, come gia scritto, sono diversi, come ad esempio: DDoS, Spamming, phishing.

2.2.6 Meccanismi di protezione

La struttura basata sul C&C è il punto di forza ma anche quello di debolezza delle Botnet. Come scritto sopra, l'esclusione del C&C dal resto della rete renderebbe inutile la Botnet. Dunque, è nell'interesse del Botmaster proteggere il server di comando e controllo. I principali meccanismi di protezione sono quattro[5]:

- **Bulletproof hosting**: è un servizio di hosting fornito da compagnie che ne permettono l'utilizzo fraudolento.

- **DNS dinamico**: è un servizio che lega un nome di dominio a indirizzi IP che cambiano dinamicamente. Questo fa si che il nome di dominio continuerà a puntare allo stesso host nonostante il cambiamento continuo di indirizzo IP. I malware oggi includono spesso nei loro codici binari i domain name assegnati dai provider che forniscono DNS dinamici. Il vantaggio di questo metodo è che se un C&C server viene bloccato il Botmaster può facilmente riprendere il controllo della Botnet semplicemente creando un nuovo server di controllo da qualche altra parte e aggiornando l'IP corrispondente al Domain Name. Quando le connessioni al vecchio C&C server falliscono, allora i bot fanno una query al DNS server e vengono ridiretti verso il nuovo C&C server.
- **Fast Fluxing**: si riferisce alla capacità di avere un singolo nome di dominio collegato a indirizzi IP che cambiano frequentemente. Ne deriva che un singolo domain name è collegato a molteplici indirizzi IP.
- **Domain Fluxing**: In molte tipologie di Botnet i nomi di dominio necessari per la connessione alla rete sono distribuiti direttamente con il bot agent. Possono essere hard-coded nel malware o scritti in file di configurazione. Questi nomi rimangono sempre gli stessi e possono essere aggiornati solo dal Botmaster attraverso l'uso del C&C. Se le comunicazioni con il C&C venissero in qualsiasi modo bloccate i bot diventerebbero inutili poiché il Botmaster non avrebbe alcun modo per cambiare il nome di dominio per permettere al bot di connettersi ad un altro server di controllo.
  Per questo motivo si iniziarono a creare dei bot agent con la capacità di generare in maniera autonoma nomi di dominio unici. Questa capacità viene chiamata "domain fluxing". Il software che permette questo processo prende il nome di "Domain generation algorithm" o DGA.

2.3 DGA

IL DGA è un codice inserito direttamente all'interno del malware che genera, a partire da un seed, nomi di dominio utilizzati dal bot agent per connettersi a un server C&C. I meccanismo di domain fluxing basato su DGA ha diversi vantaggi[5]:

- Evade il meccanismo di blacklisting: il numero di nomi di dominio generati dai DGA è troppo grande da renderne inutile il blacklisting.
- Per il Botmaster è sufficiente registrare un nome di dominio che sicuramente verrà, prima o poi, generato dal DGA in futuro.
- I nomi di dominio generati vengono usati per un breve periodo di tempo rendendo il sistema del "domain reputation" inutile.

Nonostante tutti questi vantaggi il meccanismo dei DGA è ben lontano da essere perfetto infatti[5]:

- Alcuni DGA generano un elevato numero di nomi di dominio e talvolta possono generare dei nomi utilizzati da enti legittimi.
- Essendo il codice del DGA direttamente integrato nel malware, i ricercatori hanno la possibilità di capirne il funzionamento interno prevedendo in nomi di dominio che andrà a generare. Questo lavoro è però difficile, lungo e molto costoso.

2.3.1 Classificazione dei DGA

Benché le caratteristiche di base siano sempre le stesse esistono alcune diversità fra le varie famiglie di DGA. Di seguito si propone una classificazione basata su due caratteristiche[9]: **tipologia del seed**, **schema di generazione del nome di dominio**.

Seed

Il seed di un "domain generation algorithm" è definito come l'insieme dei parametri necessari per l'esecuzione dell'algoritmo (come costanti numeriche o stringhe).

Le caratteristiche principali che distinguono i seed tra di loro sono due:

- **Tempo-Dipendenza**: il seed può o meno derivare da una risorsa tempo variante.
- **Determinismo**: il determinismo fa riferimento all'osservabilità e disponibilità dei parametri. Per la maggior parte dei DGA conosciuti tutti i parametri necessari per l'esecuzione dono conosciuti ad un livello tale da poter calcolare tutti i possibili domini che verrebbero generati.

Queste due elementi permettono 4 combinazioni:

- Tempo-indipendente e deterministico (TID)
- Tempo-dipendente e deterministico (TDD)
- Tempo-dipendente e non deterministico (TDN)
- Tempo-indipendente e non deterministico (TIN)

Schemi di generazione dei nomi di dominio

Secondo alcune analisi[9] sono emersi quattro tipologie di schemi di generazione:

- **Arithmetic-based DGA:** calcolano una sequenza di valori che o hanno una diretta rappresentazione ASCII utilizzabile per un nome di dominio oppure designano uno scostamento in uno o più array hard-coded nell'algoritmo.
- **Hash-based DGA:** utilizzato la rappresentazione esadecimale di una hash per produrre il nome di dominio.
- **Wordlidst-based DGA:** concatenano una serie di parole prese da una o più liste, costruendo nomi di dominio che appaiono meno randomici e più credibili. Le liste di parole utilizzate possono essere hard-coded o prese da risorse online pubblicamente accessibili.
- **Permutation-based DGA:** derivano tutti i possibili nomi di dominio da una permutazione di un nome di dominio iniziale.

2.3.2 Una soluzione al problema dei DGA

Come già descritto non è possibile combattere le Botnet basate su DGA ricorrendo a meccanismi classici come blacklisting. Altri approcci come il reverse engineering del malware per scoprire il funzionamento del DGA e predire così i nomi di domino che andrà a generare risultano nella pratica non utilizzabili perché troppo spesso

lunghi e complicati e dunque incapaci di combattere tutte le nuove famiglie di DGA che nascono e vengono diffuse costantemente sul web.

Un altro problema è il forte squilibrio che esiste fra Botmaster e chi tenta di bloccare una botnet infatti, se il Botmaster deve mantenere attivo un solo nome di dominio, chi si occupa di cybesecurity deve bloccare tutti i migliaia nomi di dominio generati costantemente dal DGA in quanto è impossibile sapere quali di essi è quello utilizzato dal Botmaster.

Nonostante queste problematiche esistono modi per contrastare la diffusione dei DGA. Uno valido è quello di controllare le richieste fatte ai servizi DNS per capire quali di esse sono fatte da DGA e bloccarle. Per far questo è possibile costruire un classificatore in grado di distinguere quali query ai DNS server sono malevoli e quali sono legittime.

2.4 Machine Learning

Come descritto nella sezione precedente un modo per chiudere una Botnet basata su DGA è andare a intercettare le richieste fatte ai server DNS.

Per far ciò si necessita di un metodo capace di individuare nuovi domain name malevoli a partire da una lista già esistente. Una possibile soluzione a questo problema è il Machine Learning.

Il Machine Learning (ML) è una branca dell'intelligenza artificiale che studia algoritmi che possono migliorare automaticamente attraverso l'esperienza grazie all'utilizzo di dati dai quali possano apprendere.

La grande potenza di questi algoritmi sta nella loro capacità di fare delle predizioni su dati totalmente nuovi a partire da dati di esempio senza essere esplicitamente programmati per farlo.

Questa caratteristica è fondamentale nell'individuazione di domain name generati da DGA in quanto permette la classificazione di nomi di dominio nuovi a partire da una lista precostituita.

2.4.1 Metodi di addestramento

Tutti gli algoritmi di Machine Learning necessitano, prima di poter essere utilizzati, di essere addestrati. I metodi di addestramento di dividono tradizionalmente in tre categorie:

- **Approccio supervisionato**: all'algoritmo vengono presentati dei dati dove ogni elemento è costituito dalla coppia dato di input (tipicamente un vettore) e il corrispettivo output desiderato. In questo approccio i dati vengono dunque "etichettati", ovvero ogni elemento viene già classificato prima di essere fornito all'algoritmo.
- **Approccio non supervisionato**: all'algoritmo non viene fornita alcuna etichetta è esso stesso a dover individuare una struttura a partire dai dati di input.

- **Reinforcement learning**: un programma interagisce con un ambiente dinamico nel quale deve perseguire un certo obiettivo. Mentre naviga questo spazio il programma riceve dei feedback simili a delle ricompense che deve cercare di massimizzare.

2.4.2 Features

Nel machine learning le ”features” sono delle variabili indipendenti che vengono utilizzate nei modelli e vengono estratte dai dati di ingresso. In altre parole sono delle caratteristiche o proprietà misurabili di un fenomeno che i modelli prendono in considerazione quando fanno delle predizioni. In base alle features utilizzate si può fare una ulteriore classificazione dei modelli. Gli approcci più utilizzati per il rilevamento di DGA sono tre[20]:

- **Context-Aware Feature.** Sono le feature che tengono conto del contesto nel quale vengono applicate. Sono dunque dipendenti dall'esecuzione dello specifico malware realizzata in uno specifico ambiente, con una specifica configurazione e in un particolare intervallo di tempo.
- **Context-Free Feature.** Feature legata solamente ad un FQDN (nome di dominio completo), indipendente da ogni informazione di contesto.
- **Featureless.** I modelli featureless non necessitano di feature per il training, lavorano direttamente sui dati ”grezzi”.

I modelli con feature sono potenzialmente più affidabili e hanno diversi vantaggi come: trasparenza, efficienza, scalabilità. Tuttavia la creazione dell'insieme delle feature che permetta un funzionamento ottimale del modello è un processo lungo e complicato.

Nel nostro caso dunque si è seguito un approccio featureless, più semplice ma comunque in grado di fornire buoni risultati, e supervisionato, utilizzando dunque un dataset già etichettato. Gli algoritmi che hanno queste due caratteristiche si basano sulle Reti Neurali Profonde o Deep Learning.

2.5 Deep Learning

Il deep learning è una branca del machine learning che si basa su reti neurali artificiali (ANN) in grado di individuare da sole le feature necessarie per la classificazione. Queste reti neurali sono composte da molteplici strati (da questo deriva il nome ”Deep”) che permettono di scoprire pattern e astrarre caratteristiche generali da un grande insieme di dati.

Ogni strato ”impara”, dai dati, un concetto sul quale si baserà lo strato successivo. Più alti sono i livelli più i concetti imparati sono astratti.

2.5.1 Reti neurali artificiali (ANN)

Le reti neurali artificiali nascono come una generalizzazione di modelli matematici che rappresentano il cervello umano. Proprio come quest'ultimo è composto da una

serie di neuroni collegati tra loro, anche le ANN sono composte da un insieme di elementi funzionali di base chiamati neuroni artificiali o nodi.

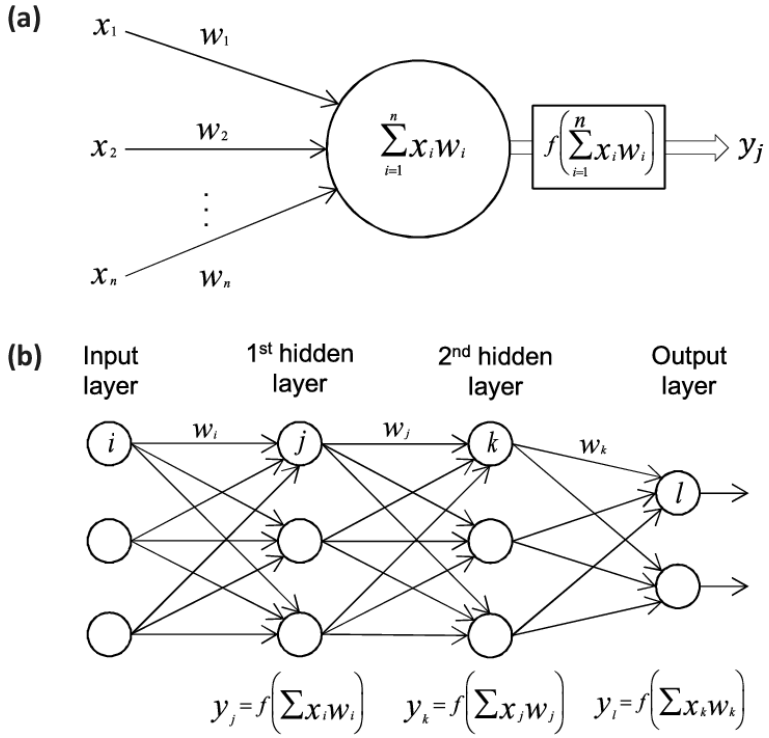

Figura 2.6. Dettaglio di un singolo neurone (a) e esempio di rete neurale (b) [7]

Ad ogni nodo arrivano n input e ad ogni input viene associato un peso, i contributi di tutti questi input, moltiplicati per i pesi, vengono sommati e forniti ad una funzione di attivazione. Se la somma totale supera una certa soglia di threshold il nodo diventa ”attivo” e viene fornito un output che verrà passato al nodo successivo.

Addestrare una rete neurale significa aggiustare i pesi che moltiplicano gli input di ogni neurone.

Architettura di una rete neurale artificiale

In generale una rete neurale si costituisce di tre principali strati di neuroni: input layer, hidden layer, output layer.

- **Input layer.** Si occupa di prendere i dati in input e passarli agli strati successivi che li processeranno.
- **Hidden layer.** Questo livello può essere composto da uno o più strati ed è il cuore della rete dove avvengono tutte le elaborazioni.
- **Output layer.** Si prendono i risultati dei calcoli dei layer precedenti per produrre il risultato finale.

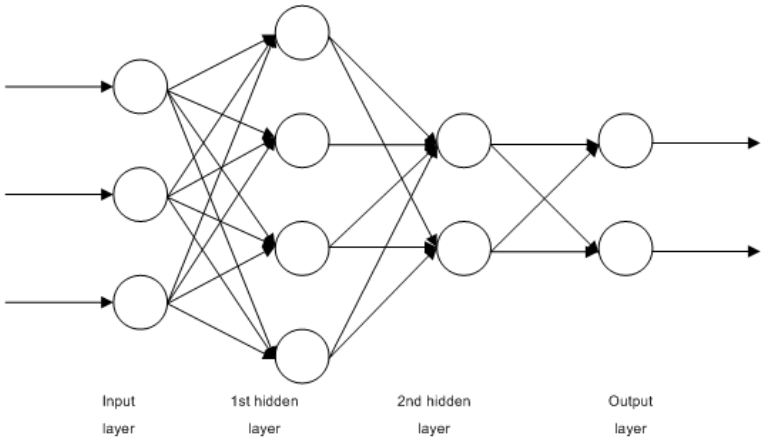

Figura 2.7. Strutturazione a strati di una rete neurale [4]

In ogni rete neurale la struttura di base, composta dai tre strati sopra descritti, rimane sempre la stessa, ma può cambiare come le informazioni passano da uno strato all'altro. Si dividono in reti neurali ricorrenti e reti neurali non ricorrenti.

- **Reti neurali ricorrenti.** Le informazioni viaggiano tra gli strati della rete in entrambe le direzioni introducendo dei cicli nella rete.
- **Reti neurali non ricorrenti.** Le informazioni possono viaggiare in una sola direzione.

Reti neurali ricorrenti (RNN)

La strutturazione a cicli di queste reti permette di conservare una memoria dei dati già analizzati consentendo di svolgere compiti che richiedono di mantenere informazioni nel tempo. Questa peculiarità le rende molto performanti per il riconoscimento di testo, di scrittura a mano o per il processamento del linguaggio naturale.

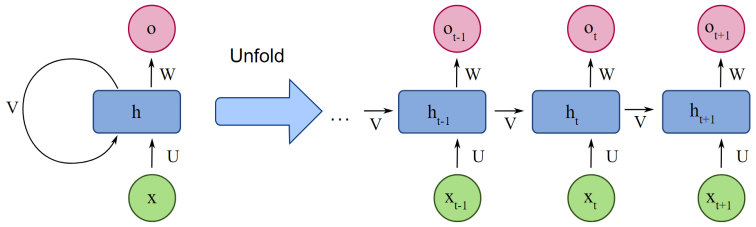

Figura 2.8. Rete neurale ricorrente [16]

Le RNN hanno però il difetto di poter mantenere in memoria solo una piccola quantità di informazioni, ciò ne limita l'applicazione. Molto spesso i nomi di dominio generati dai DGA sono molto lunghi, dunque le reti RNN, talvolta, non hanno abbastanza memoria temporale per poterli analizzare per intero[11].

Per questo, nel nostro caso, verrà utilizzata una tipologia di rete più performante che nasce direttamente dalle RNN e prende il nome di Long-Short Term Memory o LSTM.

Long-Short Term Memory (LSTM)

Le LSTM sono particolari reti RNN usate nel campo del deep learning. Le LSTM sono più efficaci nella risoluzione di problemi che dipendono da intervalli di tempo molto lunghi.

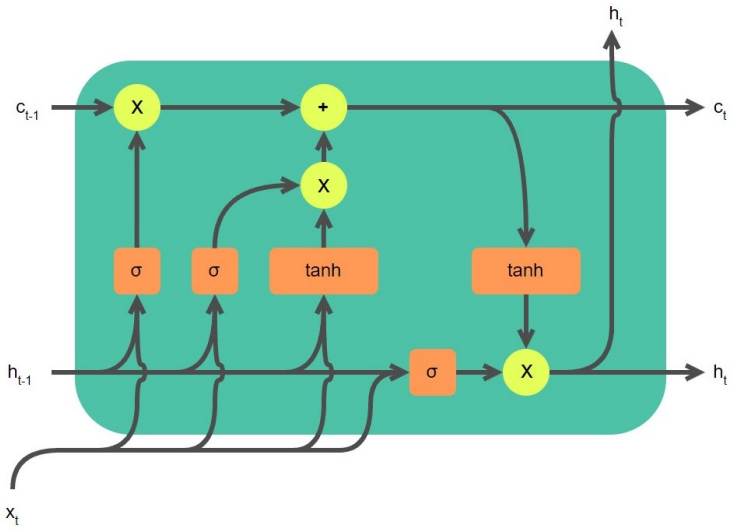

Figura 2.9. Struttura interna di una rete LSTM [3]

Ogni cella di memoria delle reti LSTM include quattro elementi principali chiamati: input gate, forgetting gate, output gate, self-looping connected units.

Ogni cella ha uno stato con una connessione ricorrente a se stesso che le permette di conservare lo stato ad ogni ciclo[18]. Questo stato può essere modulato attraverso l'input gate, che moltiplica il segnale di input per un numero compreso fra 0 e 1 oppure -1 e 1 dipendentemente dalla funzione di attivazione. Allo stesso modo il forget gate regola la connessione ricorrente dello stato con se stesso moltiplicandolo per un numero compreso fra 0 e 1. Dunque, ad esempio, se l'input gate modula l'input con 0 e il forget state modula lo stato con 1, la cella ignora l'input e mantiene lo stato corrente. Infine l'output gate controlla il contributo dello stato della cella sull'output che si propagherà nei layer successivi.

Questa struttura permette alle reti LSTM di mantenere memoria dei dati analizzati per un periodo lungo con prestazioni migliori rispetto alle semplici RNN.

Multiclass imbalance

Per poter funzionare al meglio le reti neurali artificiali necessitano di un dataset ampio per l'addestramento. Costruire una dataset per la classificazione dei DGA è molto complicato perché risulta difficile collezionare dati a sufficienza causando un forte squilibrio tra la quantità di nomi di dominio benevoli e nomi di dominio generati da DGA[14]. Questo squilibrio può portare le reti LSTM ad avere dei bias nei confronti della classe dominante, nel nostro caso quella dei benevoli, rendendo alcune famiglie di DGA praticamente non rilevabili.

Si può dunque utilizzare un algoritmo chiamato LSTM.MI più robusto delle semplici reti LSTM di fronte a problemi con dataset fortemente sbilanciati. Secondo gli autori di [14] le reti LSTM.MI possono portare ad un miglioramento fino al 7% su tutte le principali metriche rispetto alle LSTM in caso di dataset sbilanciati.

3

Modelli e Classificatori Ensamble

Lo scopo di questo lavoro è la costruzione e il confronto di sistemi multi-classificatori (MCS) per individuare quali di essi è il migliore per la classificazione di nomi di dominio generati dai DGA.

Dopo aver analizzato nel capitolo precedente il background teorico verranno ora introdotti i modelli di base utilizzati, i sistemi MCS e, infine, i modelli utilizzati per gli esperimenti.

3.1 Introduzione generale ai modelli

I modelli o classificatori che sono stati costruiti (tutti basati su una rete LSTM) sono sei e si distinguono in base a due caratteristiche: modalità di classificazione (binaria o multiclasse) e strato di embedding.

- **Modalità di classificazione:** con modalità di classificazione si vuole indicare la possibilità di classificare un nome di dominio distinguendolo fra due sole classi (benevolo o malevolo), in questo caso si parla ovviamente di classificazione binaria, o fra più classi (nel nostro caso sono 26 e le discuteremo in seguito), si parla di classificazione multi-classe.
- **Strato di embedding:** è il primo strato di una rete neurale, è quello che converte il dato in input, ad esempio un nome di dominio, in un array di numeri creando uno spazio vettoriale utilizzato dalla rete neurale per la classificazione. Da lavori precedenti [2] troviamo che nei modelli creati lo strato di embedding può essere di tre tipi: randomico, basato su ELMo, basato su FastText.

3.1.1 ELMo e FastText

Prima di procedere nella discussione dei modelli è necessario riportare una breve spiegazione di quello che sono ELMo e FastText.

ELMo è una rete neurale pre-addestrata capace di riconoscere il testo delle parole all'interno di una frase. Si basa su una rete bi-LSTM ovvero LSTM bidirezionale. Questo permette di confrontare una parola all'interno di una frase sia con le parole che la precedono sia con quelle che la seguono.

Nel nostro contesto ELMo è inserito nello strato di embedding del modello per capire se una parola ha senso di trovarsi all'interno di un domain name.

FastText è una libreria che lavora per rappresentazioni di testo e classificatori di testo con modelli vettoriali di parole pre-formati.

La caratteristica principale di FastText è di utilizzare la suddivisione in monogrammi, bigrammi, trigrammi delle parole permettendo una classificazione efficace anche di parole al di fuori del dataset originale.

3.1.2 Struttura dei classificatori

Di seguito verranno presentate nel dettaglio le strutture dei classificatori mostrando i vari strati che li compongono. Nelle immagini riportate ogni rettangolo rappresenta uno strato della rete, dentro ognuno di essi vengono indicati:

- Nome dello strato
- Tipo dello strato
- Informazioni sulla struttura dei dati in input.
- Informazione sulla struttura dei dati in output.

Sia per le informazioni di input sia per quelle di output viene indicata una tupla che mostra il numero di elementi che il vettore di input o di output ha per ogni sua dimensione. Il primo elemento di ogni tupla, come è possibile vedere nelle figure seguenti, è indicato con il temine "none". Esso identifica che quella dimensione del vettore può avere un numero di elementi qualsiasi. Ciò succede perché il primo posto della tupla indica il batch size ovvero il numero di elementi che vengono propagati nella rete durante l'addestramento. Nel nostro caso, non essendo esplicitamente definito, è completamente dipendente dal numero degli elementi dati in input durante il training e quindi risulta essere variabile.

- **Classificatore binario con strato di embedding randomico**

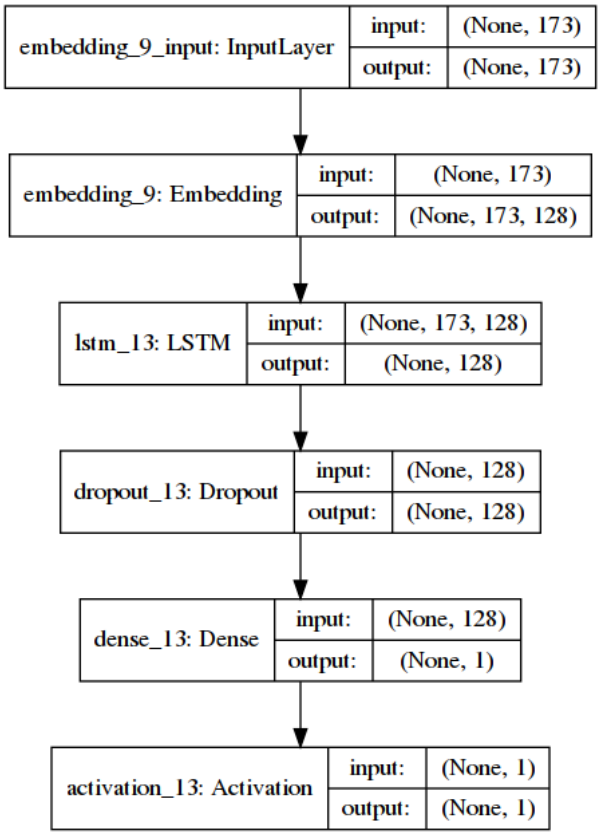

Figura 3.1. Layer che compongono Random-Binario

- **Classificatore binario con strato di embedding basato su FastText**

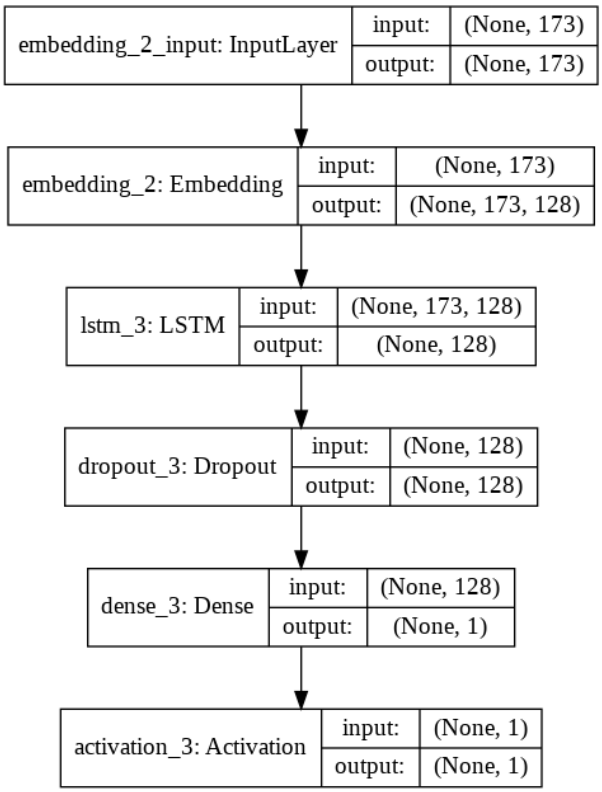

Figura 3.2. Layer che compongono FasText-Binario

- **Classificatore binario con strato di embedding basato su ELMo**

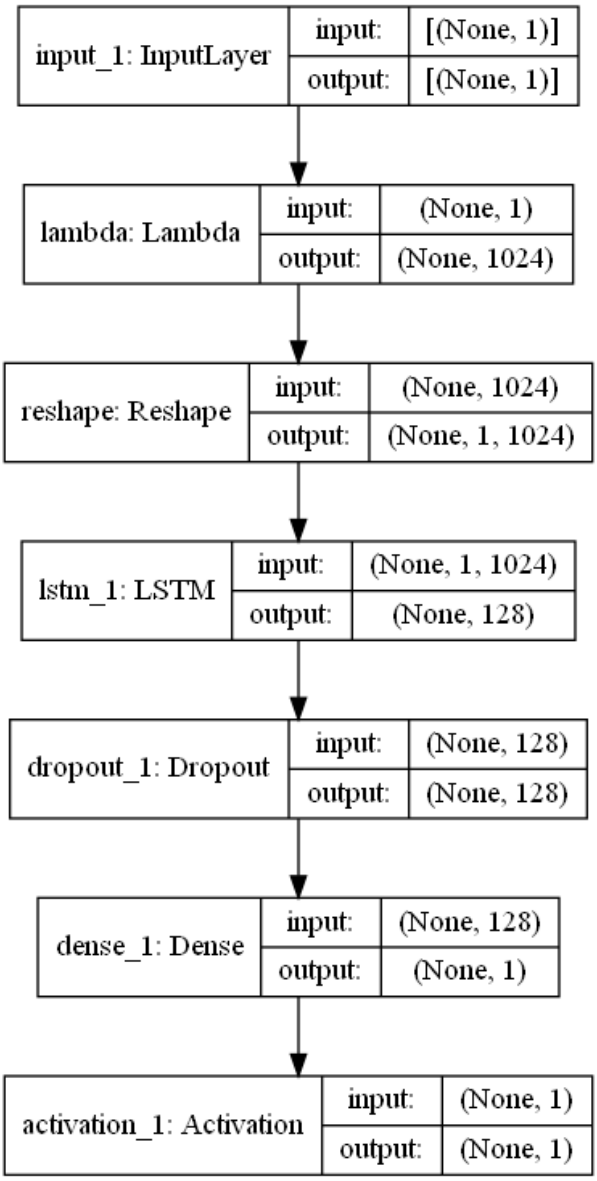

Figura 3.3. Layer che compongono ELMo-Binario

- **Classificatore multi-classe con strato di embedding randomico**

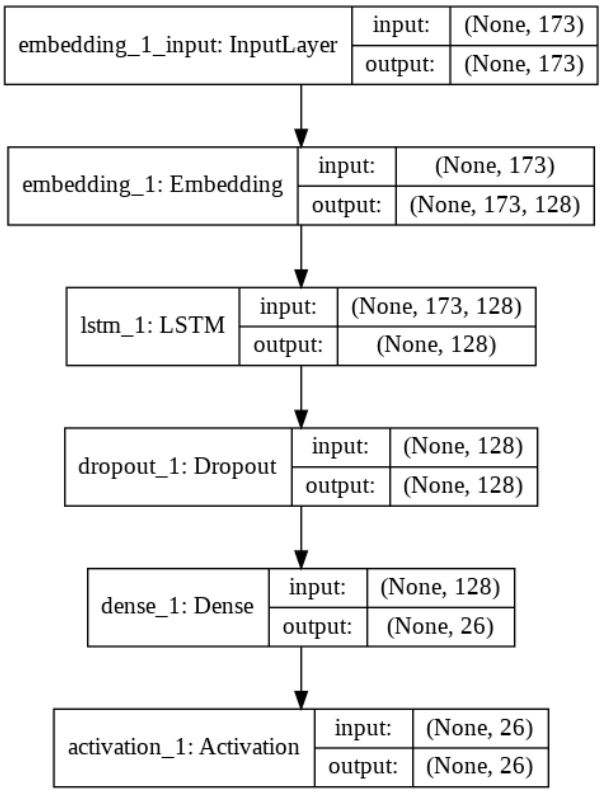

Figura 3.4. Layer che compongono Random-Multiclasse

- **Classificatore multi-classe con strato di embedding basato su FastText**

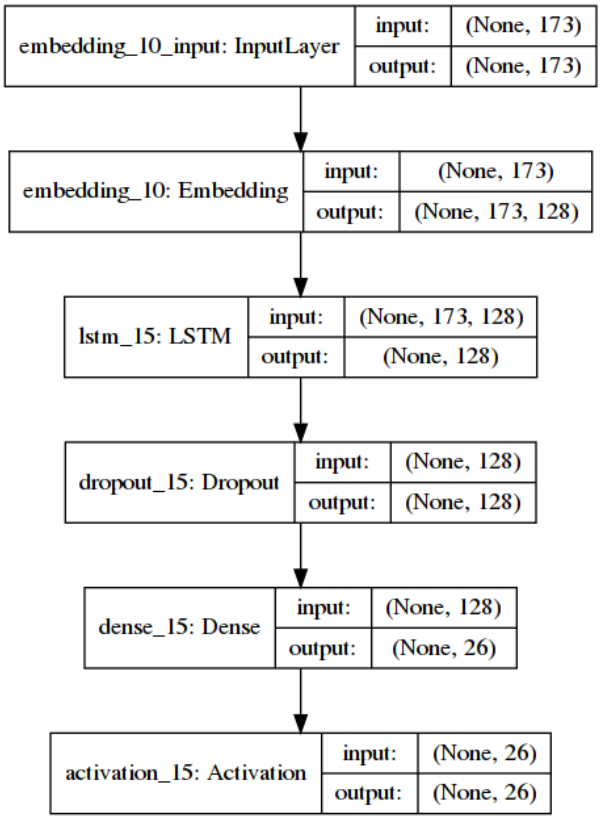

Figura 3.5. Layer che compongono FastText-Multiclasse

- **Classificatore multi-classe con strato di embedding basato su ELMo**

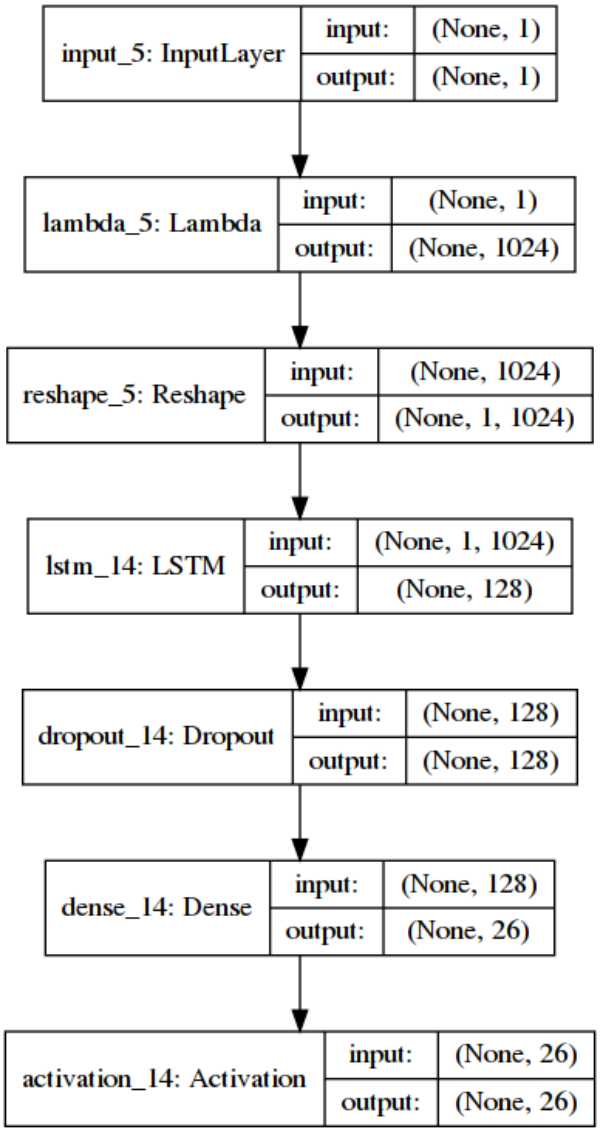

Figura 3.6. Layer che compongono ELMo-Multiclasse

Da qui in avanti, per comodità e chiarezza, si farà riferimento ai modelli sulla base dello strato di embedding e della modalità di classificazione. Ad esempio, il classificatore binario con strato di embedding randomico verrà chiamato ”random-binario”.

I layer utilizzati per la costruzione dei modelli con strato di embedding randomico e basato su FastText sono gli stessi mentre differiscono quelli con strato di embedding basato su ELMo. Nel primo caso troviamo:

1. *InputLayer:* è lo strato che prende i dati di input e li ordina in una array.
2. *Embedding:* come già spiegato è lo strato che si occupa di trasformare un input in un vettore di numeri. Nel caso di embedding randomico i vettori vengono costruiti associando ad ogni carattere che si trova nei nomi di dominio del dataset dei valori interi randomici. Nel caso di FastText invece, vengono utilizzati dei vettori di numeri interi associati a monogrammi, bigrammi e trigrammi derivati da un pre-addestramento di FastText.
3. *LSTM:* è il layer che contiene la rete LSTM.
4. *Dropout:* annulla randomicamente alcuni dati di input con un frequenza fornita dall'utente allo scopo di evitare l'overfitting della rete.
5. *Dense:* in questo strato ogni input è connesso ad ogni output. Se il modello è binario allora questo strato ha un solo nodo e fornisce un solo output, se multi-classe ne fornisce 26, ovvero il numero delle classi prese in considerazione.
6. *ActivationLayer:* è lo strato che contiene la funzione di attivazione che segue l'ouput del dense layer.

Nel secondo caso alcuni layer sono differenti (fig 3.6):

1. *Lambda:* è un layer wrap che permette di inserire ELMo embedding nel modello, inoltre trasforma l'input in maniera adeguata per poter essere utilizzato dalla rete.
2. *Reshape:* serve a ristrutturare la matrice che deriva dallo strato di embedding trasformandola da una matrice a due dimensioni in una a tre dimensioni. Questa operazione è necessaria perché la rete LSTM accetta solo matrici a tre dimensioni come input.

Dei sei modelli presentati sopra "ELMo-multiclasse", "FastText-multiclasse", "random-binario" erano già presenti in lavori precedenti [2]. Ad essi si sono aggiunti i restanti tre modelli ovvero: "ELMo-binario", "FastText-binario", "random-multiclasse".

Si vuole far notare che a parità di layer di embedding ciò che differenzia un modello multiclasse da un modello binario è principalmente il *Dense layer*: nel caso binario è composto da un singolo nodo con funzione di attivazione "sigmoide", nel caso multiclasse è composto da tanti nodi quante sono le classi da analizzare e utilizza una funzione di attivazione "softmax".

Funzioni di attivazione

Nelle reti neurali artificiali, la funzione di attivazione di un nodo definisce l'output di quel nodo dato un input o un insieme di input. Nel nostro caso:

- SOFTMAX
  La funzione "softmax" converte un vettore di numeri reali in un vettore di probabilità, ogni elemento di quest'ultimo ha un valore compreso fra 0 e 1 e la somma dei valori di tutti i suoi elementi è 1. Nel caso specifico il vettore avrà

tanti elementi quante sono le classi analizzate (ovvero 26) e assocerà ad ogni elemento, e quindi ad ogni classe, la probabilità che il nome di dominio analizzato appartenga a quella classe.
- SIGMOID
  La funzione "sigmoid" restituisce in output un valore compreso fra 0 e 1. Tanto più il valore restituito si avvicina allo 0 tanto maggiore è la probabilità che l'elemento analizzato appartenga alla categoria individuata dal numero 0, stessa cosa accade con la categoria opposta individuata dal numero 1. Nel caso specifico la categoria individuata dallo 0 indica i nomi di dominio benevoli mentre l'1 indica i nomi di dominio generati dai malware.

3.2 Sistemi multi-classificatori (MCS)

Gli MCS sono combinazioni di classificatori più o meno omogenei che mirano a far emergere i punti di forza di ogni singolo classificatore creando un modello capace di performare meglio di quelli che lo compongono. Similmente a ciò che accade con gli altri sistemi nel campo dell'intelligenza artificiale [13], i sistemi multi-classificatori rispondono a un tentativo di emulazione del comportamento umano. In particolare, questi sistemi cercano di replicare le azioni di un essere umano quando affronta una decisione importante. Ad esempio, è comune a chiedere il parere di diversi medici prima di avere un intervento chirurgico, o di leggere le recensioni prima di acquistare a prodotto. In altre parole, una decisione è considerata più affidabile se è fatta sulla base del parere di diversi esperti. Estrapolazione di questa proposizione applicata al machine learning porta allo sviluppo di sistemi composti da diversi classificatori, in cui la decisione finale è presa collettivamente.

I sistemi MCS hanno diversi vantaggi tra cui [19]:

- Danno dei buoni risultati in due casi: quando si ha una scarsa quantità di dati da analizzare o quando se ne ha una grande quantità.
- Possono dare risultati migliori rispetto al miglior classificatore singolo.
- Molti algoritmi di machine learning sono euristici, mentre gli MCS permettono di iniziare la ricerca da diversi punti nello spazio aumentando la possibilità di trovare un modello ottimale.
- Possono essere implementati in maniera concorrenziale o in parallelo aumentando l'efficienza computazionale del modello.
- Gli MCS provano a "selezionare" sempre il modello ottimale per risolvere il problema.

Per ottenere tutti i vantaggi sopra descritti è necessario progettare in maniera adeguata i sistemi MCS. I problemi da affrontare durante questa fase sono tre:

- **Topologia del sistema**: come interconnettere i classificatori di base.
- **Ensemble design**: come creare e scegliere i classificatori di base da utilizzare.
- **Design del fusore**: come costruire un fusore che combini le decisioni dei modelli di base nel miglior modo possibile.

3.2.1 Topologia del sistema

I modelli di base possono essere combinati in diversi modi [19]:

- **In serie**: i classificatori sono disposti secondo una struttura sequenziale. Quando il risultato di un classificatore non supera una soglia (threshold) di attendibilità (prestabilita in maniera empirica) il dato viene passato al modello successivo fino ad arrivare eventualmente all'ultimo classificatore il cui risultato viene sempre considerato valido.

Figura 3.7. Classificatori in serie

- **In parallelo**: ad ogni classificatore vengono passati gli stessi dati da analizzare per poi combinare le decisioni dei singoli classificatori in una decisione finale.

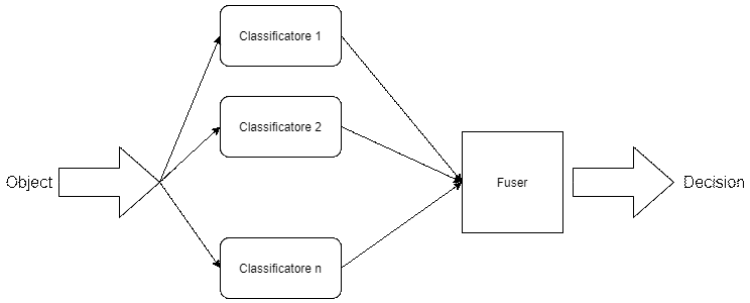

Figura 3.8. Classificatori in parallelo

- **Ibrido**: questi sistemi sono frutto della combinazione della topologia "in serie" e della topologia "in parallelo"

3.2.2 Ensemble design

L'obiettivo di base quando si progetta un MCS è garantire maggior diversità possibile aumentando la generalità del sistema e quindi le prestazioni.

Alcune tecniche per aumentare la diversità di un MCS [19]:

- Diversificare gli input. Ci sono tre strategie generali: usare differenti partizioni dei dati, usare differenti insiemi di features, prendere in considerazione la particolare specializzazione di ogni classificatore.
- Diversificare gli output. Ogni classificatore è addestrato a riconoscere solo un sottoinsieme del problema.
- Diversificare i modelli. Utilizzare modelli diversi permette al sistema MCS di prendere vantaggio dei bias che ogni modello possiede.

In questo lavoro si è scelta la terza opzione utilizzando classificatori diversi o diverse versioni dello stesso classificatore.

3.2.3 Design del fusore

Il fusore è quella componente del MCS che si occupa di unire i risultati dei modelli di base e da essi prendere une decisione finale. La scelta di un fusore adeguato è uno dei problemi più importanti nella costruzione di un MCS. Esistono diverse tipologie di fusori:

- Class label fusion: è la categoria che include tutti gli algoritmi che lavorano al livello delle risposte dei modelli di base. Un esempio è il voto di maggioranza di cui esistono tre versioni principali: tutti i classificatori sono d'accordo, almeno metà dei classificatori è d'accordo, la classe che riceve il voto più alto vince. Un'altra possibilità è il voto pesato: si possono assegnare dei pesi ai classificatori, ai classificatori e alle classi oppure si possono assegnare dei pesi basati sulle feature sia ai classificatori che alle classi.
- Support fusion: si basa sull'utilizzo di funzioni chiamate "support functions". Queste funzioni danno un voto alle decisioni dei classificatori di base, questi voti vengono poi utilizzati per prendere la decisione finale.
- Trainable fuser: è l'insieme dei fusori che possono essere addestrati a prendere decisioni partendo dalle predizioni dei modelli di base. Un esempio è lo "Stacking" o "Stacked Generalization".

3.3 Costruzione dei modelli MCS

Nella costruzione dei classificatori ensamble si sono seguite le seguenti fasi:

1. Scelta dei classificatori di base. Non sono mai stati utilizzati due classificatori di base identici nello stesso MCS in modo da garantire maggior diversificazione possibile.
2. Scelta della topologia del sistema.
3. Design del fusore.

Ogni MCS si distingue dagli altri per come una o più di queste fasi sono state eseguite. L'obiettivo è stato quello di creare sistemi multi-classificatori più variegati possibile per avere poi un valido confronto.

3.3.1 Lavori precedenti

Si è deciso di inserire in questo confronto anche MCS costruiti in lavori precedenti [2] per valutare eventuale miglioramenti o peggioramenti. Questi MCS sono i seguenti:

- La prima architettura presenta Random-binario, ELMo-multiclasse, FastText-multiclasse in serie. La logica implementativa segue quella dei modelli in serie con una soglia di threshold per i modelli ottenuta empiricamente.

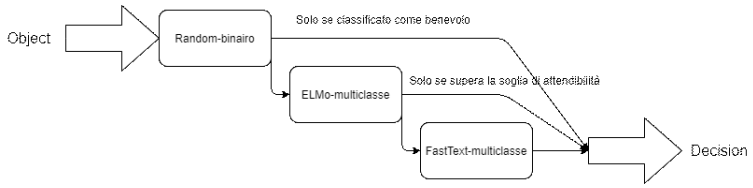

Figura 3.9. In serie: Random-binario – ELMo-multiclasse – FastText-multiclasse

- La seconda architettura è molto simile alla prima con una inversione di FastText-multiclasse e ELMo-multiclasse.

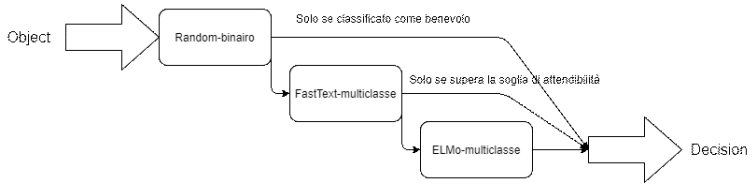

Figura 3.10. In serie: Random-binario – FastText-multiclasse – ELMo-multiclasse

- L'ultimo MCS utilizza una struttura diversa mettendo FastText-multiclasse e ELMo-multiclasse in parallelo. Ognuno di questi due modelli restituisce in output 26 valori di probabilità (uno per ogni classe), tra i 52 output disponibili il fusore prende la classe che ha la probabilità maggiore di essere quella giusta.

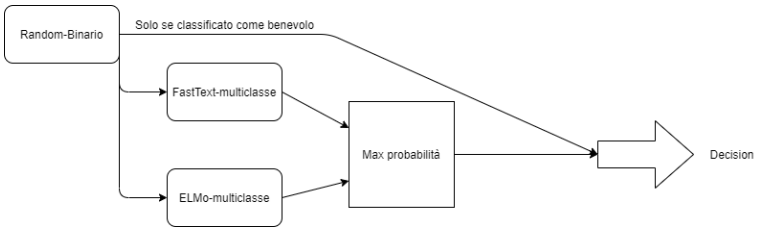

Figura 3.11. In parallelo: FastText-multiclasse – ELMo-multiclasse in serie con Random-Multiclasse

3.3.2 Nuovi modelli

I nuovi classificatori ensamble inseriti appositamente per questo lavoro sono:

1. ”ELMo-binario”, ”FastText(bigrammi)-binario, ”Random-multiclasse” posti in serie.
   La logica implementativa di questo modello segue quasi nella sua totalità quella generale tipica dei sistemi in serie: quando il risultato di un classificatore non supera una soglia (threshold) di attendibilità (prestabilita in maniera empirica) il dato viene passato al modello successivo fino ad arrivare eventualmente all'ultimo classificatore il cui risultato viene sempre considerato valido. Per avere omogeneità di output, tuttavia, i classificatori binari contribuiscono alla decisione finale solo quando classificano i domini come benevoli, in caso contrario l'analisi del dominio è demandata al classificatore multiclasse in modo da garantire che ogni dominio sia ”posizionato” all'interno di una delle classi prese in esame. In particolare la logica utilizzata è: ELMo-binario analizza un dominio, se la probabilità che appartiene alla classe dei benevoli supera una certa soglia la rete restituisce il risultato classificandolo come benevolo, altrimenti viene passato al FastText-binario che compie la medesima operazione e infine viene eventualmente passato a random-multiclasse che restituirà in ogni caso un risultato.

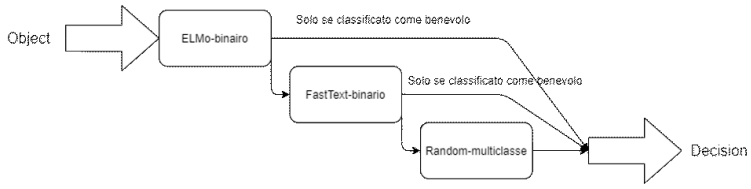

Figura 3.12. In serie: ELMo-binairo – FastText-binario – Random-multiclasse

2. ”ELMo-binario”, ”FastText(bigrammi)-binario” tra loro in parallelo e posti in serie con ”Random-multiclasse”.

   Questo modello pone due classificatori binari in parallelo tra loro e poi in serie con il classificatore multiclasse. Data la presenza di classificatori in parallelo è necessario creare un fusore che possa, a partire dai risultati dei singoli modelli, estrarre un risultato comune. Il fusore è costruito utilizzando un tecnica che sfrutta i valori delle probabilità restituite in output dai due modelli: si fa la media dei valori, se essa supera una certa soglia il nome di dominio in esame verrà classificato come benevolo, altrimenti viene passato al classificatore multiclasse il quale restituirà in ogni caso un risultato associando al dominio una classe.
   Si ribadisce che i due modelli binari, avendo un funzione di attivazione ”sigmoid”, restituiscono in output un valore compreso fra 0 e 1. Tanto più il valore è vicino allo 0 tanto più la probabilità che il nome di dominio appartenga alla categoria indicata dal valore 0 (nel nostro caso è la categoria dei nomi di dominio benevoli) è maggiore.

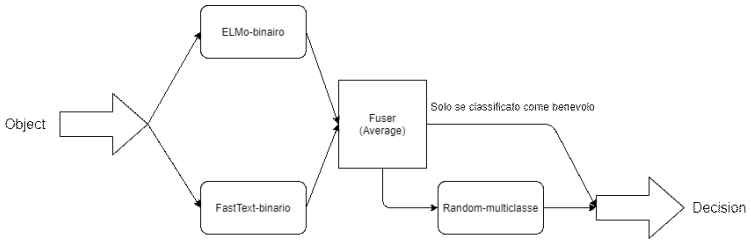

Figura 3.13. ELMo-binario in parallelo con FastText-binario, entrambi in serie con Random-multiclasse.

3. In parallelo: ”ELMo-multiclasse”, ”FastText(mogorammi)-multiclasse, ”FastText(bigrammi)-multiclasse, ”FastText(trigrammi)-multiclasse”, ”Random-multiclasse”.

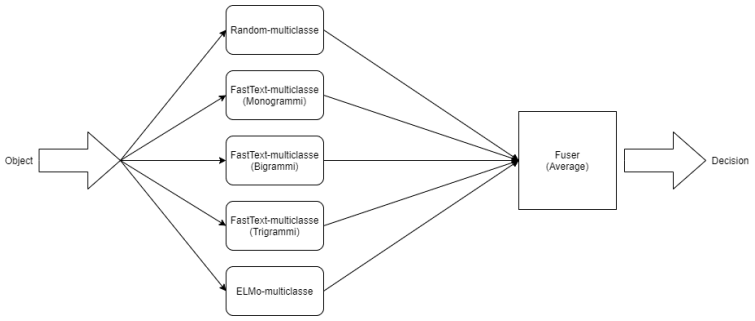

Figura 3.14. In parallelo: ”ELMo-multiclasse”, ””FastText(mogorammi)-multiclasse, ”FastText(bigrammi)-multiclasse, ”FastText(trigrammi)-multiclasse”, ”Random-multiclasse”.

I classificatori hanno tutti una funzione di attivazione ”softmax” dunque ognuno di essi associa ad ogni classe una probabilità compresa fra 0 e 1, perciò dall’output dei modelli si ricavano cinque probabilità distinte per ognuna delle 26 classi. Il fusore compie due operazioni:

a) Per ogni classe viene fatta la media dei cinque valori delle probabilità ad essa associate.
b) Tra le 26 classi viene scelta quella la cui media delle probabilità risulta maggiore.

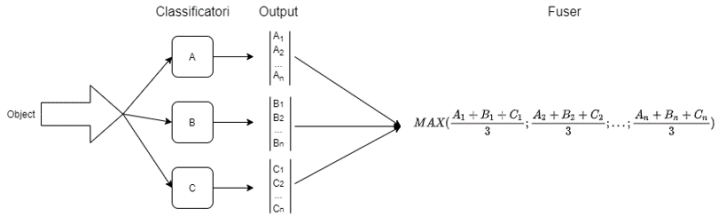

Figura 3.15. Funzionamento del fusore per il modello basato su "Average".

4. In parallelo: "ELMo-multiclasse", ""FastText(mogorammi)-multiclasse, "FastText(bigrammi)-multiclasse, "FastText(trigrammi)-multiclasse", "Random-multiclasse".
   Questo modello si differenzia molto da quelli presentati finora in quanto utilizza un fusore non più statico ma "addestrabile"(trainable). In questa tecnica, che prende il nome di "Stacking" o "Stacked Generalization", si utilizza un ulteriore classificatore, chiamato meta-model o level1-model, come fusore: esso prende come input gli output dei modelli da combinare, chiamati base-models o level0-models, e da essi prende una decisione. Nel nostro caso i modelli base sono "ELMo-multiclasse", "FastText(mogorammi)-multiclasse, "FastText(bigrammi)-multiclasse, "FastText(trigrammi)-multiclasse", "Random-multiclasse" mentre il fusore è un classificatore multiclasse chiamato "LogisticRegression" che viene fornito dalla libreria "sklearn" di python. Questo modello nasce sia per classificazioni binarie che multiclasse. Dato che nel nostro esperimento ci si trova nel secondo caso il modello utilizza la regressione logistica multinomiale: un metodo di classificazione che generalizza la regressione logistica a problemi multiclasse.

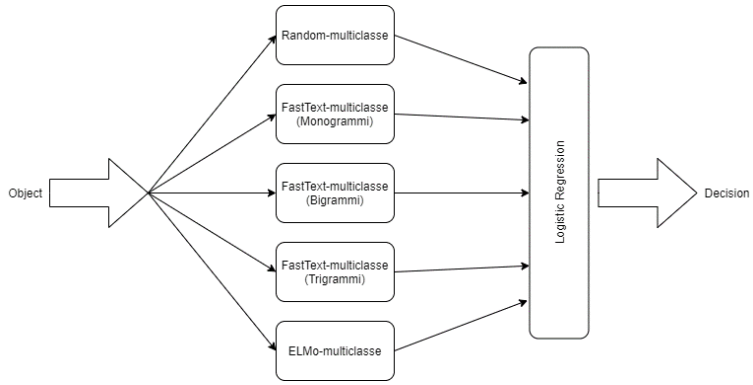

Figura 3.16. Stacking in parallelo: "ELMo-multiclasse", "FastText(mogorammi)-multiclasse, "FastText(bigrammi)-multiclasse, "FastText(trigrammi)-multiclasse", "Random-multiclasse".

3.3.3 Stacked Generalization

Lo ”Stacking” o ”Stacked generalization” è una tecnica, introdotta da Wolpert nel 1992 [17], il cui scopo è di acquisire una accuratezza di generalizzazione il più alta possibile. Wolpert afferma che la stacked generalization funziona deducendo i bias dei ”generalizer” rispetto a un determinato dataset di addestramento. Questa deduzione viene fatta generalizzando in una secondo spazio i cui input sono le supposizioni dei ”generalizer” di base addestrati su una parte del dataset di addestramento e testati sulla restante parte, mentre l'output è, per esempio, la supposizione corretta.

Per capire al meglio questa affermazione bisogna capire cos'è un ”generalizer” secondo Wolpert. Un ”generalizer” è un algoritmo che trasforma un insieme di addestramento composto da m coppie $\{x_k \in \mathbf{R}^n, y_k \in \mathbf{R}\}$, $1 \leq k \leq m$, insieme a una domanda $\in \mathbf{R}^n$ in una risposta $\in \mathbf{R}$ o più in generale $\in \mathbf{R}^n$.

Secondo questa definizione anche una rete neurale è un ”generalizer”. In questo caso le m coppie sono formate da un elemento del dataset di addestramento e dall'etichetta associata a quel elemento; la domanda è l'elemento che deve essere classificato, mentre la risposta è l'output fornito dal modello.

La principale implementazione della stacked generalization, afferma Wolpert, è come combinatore di generalizer, dunque può essere utilizzata per combinare anche reti neurali come fatto in questo lavoro.

Quando questa tecnica venne introdotta si distinse profondamente nella risoluzione di problemi che prevedevano la presenza di più classificatori. La maggior parte delle altre tecniche come la cross-validation o il bootstrapping erano tecniche di tipo ”winner-takes-all”, la stacked generalization invece si proponeva di unire i ”generalizer” e non di scegliere fra uno di essi.

Implementazione

I passaggi per implementare lo stacking sono i seguenti:

1. Si divide il dataset in due sotto insiemi: uno per il testing e uno per il training
2. Si addestra il meta-model applicando un tecnica simile al ”k-fold cross validation” al training set. Questa tecnica consiste nel partizionare il training set in ”k” sottoinsiemi per poi ripetere per k volte i seguenti passaggi:
   - Si scelgono k-1 sottoinsiemi per il training e un insieme per il testing.
   - Si creano i base-models e li si addestrano sui k-1 sottoinsiemi.
   - I base-models addestrati vengono testati sull'insieme rimanente.
   - I risultati del testing vengono salvati e i base-models distrutti.
   - Si ricomincia il ciclo successivo scegliendo come insieme di testing un insieme mai scelto nei cicli precedenti.

   Dagli output delle k operazioni di testing sui base models si ricava un nuovo dataset sul quale viene addestrato il meta-model.
3. Si addestrano i base model sull'intero training set.
4. Una volta che i base-models e il meta-model sono addestrati è possibile testare l'intero sistema sul test set fornendo come input al meta-model l'output dei base-models.

4

Esperimenti e risultati

4.1 Esperimenti

In questa sezione, prima di procedere alla presentazione degli esperimenti, verranno descritti il dataset utilizzato e le metriche di riferimento. Si precisa che il dataset è, ovviamente, sempre lo stesso per ogni esperimento e uguale a quello dei lavori precedenti [2] in modo tale da avere un confronto affidabile dei risultati.

4.1.1 Dataset

È composto da 33750 elementi ed ogni modello è stato addestrato sul 80% del dataset e testato sul restante 20% per 5 volte mescolando ad ogni iterazione l'ordine degli elementi del dataset. I dati finali degli esperimenti sono poi ottenuti facendo la media dei risultati delle 5 iterazioni. La composizione delle classi del dataset è la seguente:

- alexa: 16875;
- conficker: 675;
- corebot: 675;
- cryptolocker: 675;
- dircrypt: 675;
- emotet: 675;
- fobber: 675;
- gozi: 675;
- kraken: 675;
- matsnu: 675;
- murofet: 675;
- necurs: 675;
- nymain: 675;
- padcrypt: 675;
- pushdo: 675;
- pykspa: 675;
- qadars: 675;

- ramdo: 675;
- ramnit: 675;
- ranbyus: 675;
- rovnix: 675;
- simda: 675;
- suppobox: 675;
- symmi: 675;
- tinba: 675;
- vawtrak: 675;

Per una classificazione binaria il dataset risulta bilanciato, mentre risulta sbilanciato per una classificazione multiclasse. In particolare:

- Il 50% degli elementi appartengono alla classe Alexa, ovvero sono benevoli.
- Il restante 50% sono malevoli suddivisi in 25 classi, dunque ogni classe ha il 2% del totale del dataset.

Ogni elemento del dataset ha la medesima forma, per esempio:

- legit/dga: ”dga”
- classe: ”vawtrak”
- dominio senza caratteri speciali: ”unaflurnifdcom”
- dominio normale: ”unaflurnifd.com”
- divisione in caratteri: ”u n a f l u r n i f d c o m”
- divisione in bigrammi: ”un na af fl lu ur rn ni if fd dc co om”
- divisione in trigrammi: ”una naf afl flu lur urn rni nif ifd fdc dco com”
- divisione in parole: ”un afl urn if d com”

Di queste caratteristiche sono state utilizzate: ”divisione in caratteri”, ”divisione in bigrammi”, ”divisione in trigrammi” per i modelli con strato di embedding basato su FastText (utilizzato nelle sue tre declinazioni), ”divisione in parole” per il modelli con strato di embedding basato su ELMo, ”divisione in caratteri” per i modelli con strato di embedding randomico.

4.1.2 Metriche

Per valutare i risultati verranno utilizzate le seguenti metriche: *accuracy*, *precision*, *recall*, *F1-Score*.

Precision

Indica la porzione di veri positivi sul totale degli elementi classificati come positivi.

La precision di un modello è molto importante in problemi in cui si desidera avere pochi falsi positivi.

$$\frac{TP}{FP + TP} \tag{4.1}$$

Recall

Viene calcolata come il numero di veri positivi sul totale degli elementi effettivamente positivi.

La recall è importante in problemi in cui si vuole essere sicuri di includere tutti i casi positivi, ovvero problemi in cui non includere un positivo è più problematico di e costoso di includere un negativo.

$$\frac{TP}{TP + FN} \tag{4.2}$$

F1-Score

Si calcola come la media armonica tra precision e recall. É un parametro più generale rispetto a precision e recall in quanto fornisce in un solo dato informazioni su entrambe queste metriche.

$$2 * \frac{precision * recall}{precision + recall} \tag{4.3}$$

Accuracy

Indica la porzione di predizioni corrette sul totale delle predizioni.

4.1.3 Micro e Macro average

Precision, recall e F1-score sono calcolate in due modi:

- **Macro-Average**: si calcola ogni metrica in maniera indipendente per ogni classe, poi si fa la media aritmetica tra tutte le classi.
  In questo tipo di calcolo ogni classe ha lo stesso peso indipendentemente dal numero degli elementi che la compongono, dunque è raccomandato per dataset bilanciati.
- **Micro-Average**: si calcola ogni metrica in maniera indipendente per ogni classe, come nel Macro-Average, poi si calcola la media pesata tra tutte le classi. Il peso associato ad una classe varia in base al numero degli elementi che la compongono. Questo tipo di media è indicata per dataset sbilanciati, nella classificazione multi-classe è dunque preferibile fare rifermento al micro-average.

4.2 Risultati

4.2.1 ELMo"Bin"-FastText"Bin"-Random"Multi"

Class report:

	precision	recall	f1-score	support
conficker	0.857354	0.608889	0.710715	135.000000
corebot	0.995533	0.989630	0.992559	135.000000
cryptolocker	0.722746	0.644444	0.679668	135.000000
dircrypt	0.540123	0.444444	0.485958	135.000000
emotet	0.988300	0.997037	0.992642	135.000000
fobber	0.774164	0.881481	0.823734	135.000000
gozi	0.886690	0.675556	0.765433	135.000000
kraken	0.883243	0.754074	0.812812	135.000000
matsnu	0.963069	0.804444	0.875186	135.000000
murofet	0.859085	0.797037	0.825093	135.000000
necurs	0.813300	0.635556	0.713372	135.000000
nymaim	0.925059	0.429630	0.584945	135.000000
padcrypt	0.998519	0.986667	0.992531	135.000000
pushdo	0.955011	0.906667	0.930013	135.000000
pykspa	0.930366	0.902222	0.915474	135.000000
qadars	0.998496	0.970370	0.984178	135.000000
ramdo	0.983974	0.982222	0.982928	135.000000
ramnit	0.418913	0.334815	0.364618	135.000000
ranbyus	0.825389	0.854815	0.837254	135.000000
rovnix	0.992491	0.952593	0.971648	135.000000
simda	0.952570	0.921481	0.936726	135.000000
suppobox	0.882406	0.754074	0.811366	135.000000
symmi	0.986893	0.997037	0.991914	135.000000
tinba	0.656653	0.752593	0.698856	135.000000
vawtrak	0.984989	0.823704	0.894209	135.000000
alexa	0.903014	0.985244	0.942303	3375.000000

	precision	recall	f1-score	support
accuracy	0.000000	0.000000	0.888652	6750.000000
macro avg	0.872244	0.799489	0.827544	6750.000000
weighted avg	0.887014	0.888652	0.882628	6750.000000

Overall accuracy = 0.8886518518518519

True Positives: 5998.4

Figura 4.1. Risultati del primo modello figura: 3.12.

Si può notare come "ramnit" sia la classe che performa peggio con valori di precision, recall ed F1-score largamente minori di tutte le altre classi. All'opposto invece si trovano: padcrypt e qadars che hanno valori di precision molto vicini al 100%, emotet

e symmi offrono invece i valori di recall maggiori, mentre sono emotet e corebot a primeggiare per F1-score.

4.2.2 (ELMo"Bin"+FastText"Bin")-Random"Multi"

Class report:

	precision	recall	f1-score	support
conficker	0.832564	0.640000	0.723015	135.000000
corebot	0.995544	0.991111	0.993308	135.000000
cryptolocker	0.719382	0.645926	0.679070	135.000000
dircrypt	0.534535	0.445926	0.484651	135.000000
emotet	0.988300	0.997037	0.992642	135.000000
fobber	0.766911	0.911111	0.831787	135.000000
gozi	0.808340	0.714074	0.756511	135.000000
kraken	0.866433	0.767407	0.812784	135.000000
matsnu	0.940289	0.832593	0.881950	135.000000
murofet	0.855008	0.800000	0.824911	135.000000
necurs	0.799453	0.641481	0.711678	135.000000
nymaim	0.873261	0.484444	0.621102	135.000000
padcrypt	0.998519	0.988148	0.993275	135.000000
pushdo	0.914308	0.960000	0.935852	135.000000
pykspa	0.924658	0.920000	0.921832	135.000000
qadars	0.998507	0.977778	0.987986	135.000000
ramdo	0.984125	0.992593	0.988244	135.000000
ramnit	0.407622	0.339259	0.362345	135.000000
ranbyus	0.823202	0.856296	0.836790	135.000000
rovnix	0.983454	0.957037	0.969645	135.000000
simda	0.932868	0.945185	0.938920	135.000000
suppobox	0.804551	0.794074	0.796920	135.000000
symmi	0.986935	0.998519	0.992658	135.000000
tinba	0.645041	0.760000	0.695497	135.000000
vawtrak	0.960159	0.885926	0.917249	135.000000
alexa	0.918409	0.966933	0.941996	3375.000000

	precision	recall	f1-score	support
accuracy	0.000000	0.000000	0.888385	6750.000000
macro avg	0.856245	0.815879	0.830485	6750.000000
weighted avg	0.886084	0.888385	0.884011	6750.000000

Overall accuracy = 0.8883851851851852

True Positives: 5996.6

Figura 4.2. Risultati del secondo modello figura: 3.13.

Anche in questo MCS i risultati designano come classe peggiore "ramnit", la quale ha i numeri peggiori in tutte le metriche.

Come è accaduto per il classificatore precedente (4.1) le classi che hanno ottenuto il valore di precision migliore sono ”padcrypt” e ”qadars”, la recall è primeggiata da ”symmi”, mentre i valori migliori di F1-score sono ottenuti da ”corebot” e ”pacrypt”.

4.2.3 ELMo”Multi” + FastText(Mono/Bi/Tri)”Multi” + Random”Multi”

Class report:

	precision	recall	f1-score	support
conficker	0.871820	0.717037	0.786833	135.000000
corebot	0.998507	0.994074	0.996283	135.000000
cryptolocker	0.723997	0.764444	0.742893	135.000000
dircrypt	0.521703	0.477037	0.496061	135.000000
emotet	0.986913	1.000000	0.993396	135.000000
fobber	0.716804	0.874074	0.787130	135.000000
gozi	0.937345	0.819259	0.874290	135.000000
kraken	0.899146	0.789630	0.840176	135.000000
matsnu	0.952204	0.937778	0.944587	135.000000
murofet	0.947118	0.817778	0.877415	135.000000
necurs	0.885205	0.703704	0.783710	135.000000
nymaim	0.925531	0.656296	0.767497	135.000000
padcrypt	0.997048	0.998519	0.997781	135.000000
pushdo	0.956226	0.964444	0.960241	135.000000
pykspa	0.964734	0.931852	0.947805	135.000000
qadars	1.000000	0.976296	0.987918	135.000000
ramdo	0.992669	1.000000	0.996315	135.000000
ramnit	0.418787	0.362963	0.384797	135.000000
ranbyus	0.858437	0.863704	0.860254	135.000000
rovnix	0.990965	0.974815	0.982788	135.000000
simda	0.971668	0.965926	0.968727	135.000000
suppobox	0.896093	0.960000	0.926576	135.000000
symmi	0.991219	1.000000	0.995583	135.000000
tinba	0.784088	0.783704	0.783337	135.000000
vawtrak	0.987508	0.933333	0.959559	135.000000
alexa	0.944197	0.982459	0.962944	3375.000000

	Precision	recall	f1-score	support
accuracy	0.000000	0.000000	0.916563	6750.000000
macro avg	0.889228	0.855736	0.869419	6750.000000
weighted avg	0.915613	0.916563	0.914311	6750.000000

Overall accuracy = 0.9165629629629629

True Positives: 6186.8

Figura 4.3. Risultati del terzo modello figura: 3.14.

La classe che performa peggio in tutte le metriche è ancora ”ramnit”. Al contrario, tra le classi con i risultati migliori, se ne possono trovare diverse che hanno delle metriche con valori pari a 1. Ad esempio, per precision, la classe migliore è ”qadars”. Per recall troviamo diverse famiglie con risultati perfetti: ”emotet”, ”ramdo”, ”symmi”. Infine, a performare meglio per F1-score è padcrypt.

4.2.4 Stacked Generalization o Stacking

Class report:

	precision	recall	f1-score	support
conficker	0.862581	0.754074	0.804663	135.000000
corebot	0.998507	0.994074	0.996283	135.000000
cryptolocker	0.731748	0.712593	0.721800	135.000000
dircrypt	0.578270	0.496296	0.524371	135.000000
emotet	0.991219	1.000000	0.995583	135.000000
fobber	0.791599	0.896296	0.838667	135.000000
gozi	0.927141	0.859259	0.891810	135.000000
kraken	0.897723	0.808889	0.850527	135.000000
matsnu	0.981702	0.946667	0.963758	135.000000
murofet	0.878624	0.837037	0.856726	135.000000
necurs	0.835347	0.771852	0.801728	135.000000
nymaim	0.902613	0.743704	0.815210	135.000000
padcrypt	0.997048	0.998519	0.997781	135.000000
pushdo	0.962948	0.960000	0.961383	135.000000
pykspa	0.965309	0.943704	0.954147	135.000000
qadars	1.000000	0.979259	0.989456	135.000000
ramdo	0.994118	0.998519	0.996310	135.000000
ramnit	0.422138	0.425185	0.416316	135.000000
ranbyus	0.879904	0.863704	0.871328	135.000000
rovnix	0.995500	0.977778	0.986545	135.000000
simda	0.974547	0.962963	0.968626	135.000000
suppobox	0.928769	0.960000	0.943956	135.000000
symmi	0.992690	1.000000	0.996321	135.000000
tinba	0.777836	0.807407	0.791210	135.000000
vawtrak	0.989216	0.948148	0.968180	135.000000
alexa	0.956312	0.981867	0.968920	3375.000000
	Precision	recall	f1-score	support
accuracy	0.000000	0.000000	0.923852	6750.000000
macro avg	0.892823	0.870300	0.879677	6750.000000
weighted avg	0.923298	0.923852	0.922514	6750.000000

Overall accuracy = 0.9238518518518518

True Positives: 6236

Figura 4.4. Risultati del quarto modello figura: 3.16.

Questo MCS, come verrà discusso nella sezione seguente, è quello con i risultati più convincenti. Anche qui si possono individuare le classi migliori e peggiori per ogni metrica. Quella peggiore è sempre ”ramnit”, mentre tra le migliori si possono individuare: ”qadars” per la precision, ”emotet” e ”symmi” per la recall e ”padcrypt” per f1-score.

4.3 Discussione e confronto dei risultati

Dai risultati degli esperimenti è possibile notare come i classificatori che performano meglio lo fanno per tutte le metriche prese in considerazione superando in maniera totale i classificatori peggiori. Per aiutare il confronto dei dati sono presentati alcuni istogrammi:

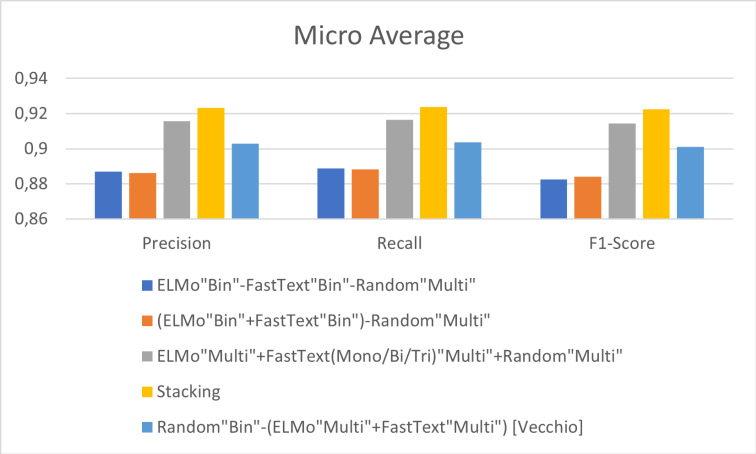

Figura 4.5. Confronto tra esperimenti. (micro average).

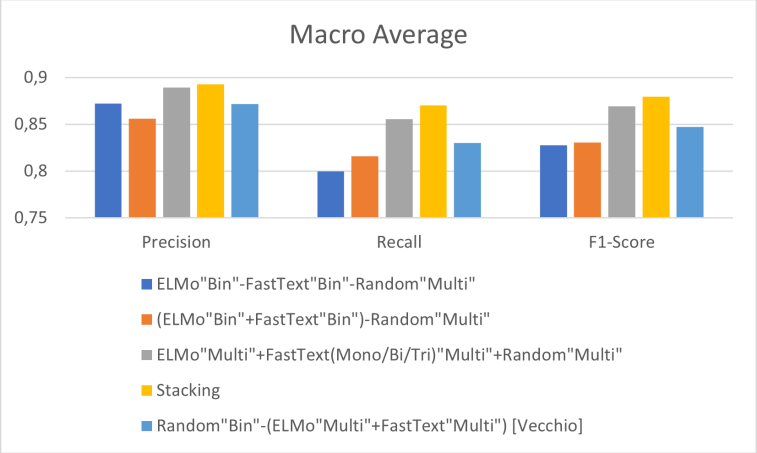

Figura 4.6. Confronto tra esperimenti. (macro average).

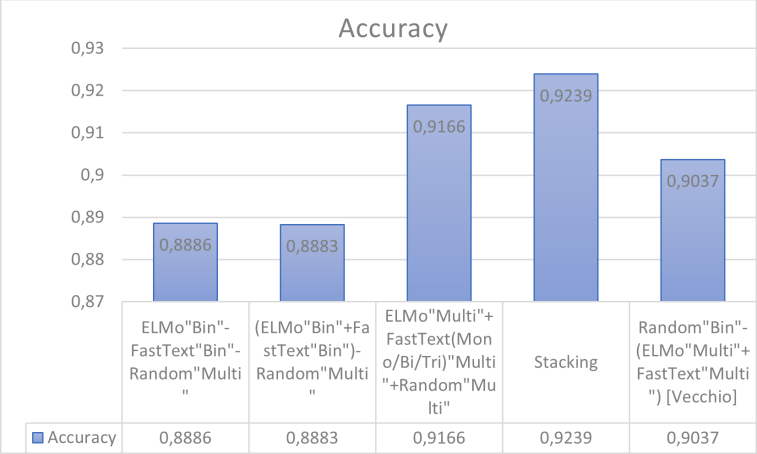

Figura 4.7. Confronto tra esperimenti. (Accuracy).

I primi due modelli (ELMo-Binario – FastText-Binario – Random-Multiclasse, ELMo-Binario – FastText-Binario tra loro in parallelo e posti in serie con Random-Multiclasse) forniscono delle prestazioni sostanzialmente sovrapponibili e inferiori rispetto agli altri modelli presentati. La loro debolezza principale sta, probabilmente, nell'utilizzo di un solo classificatore multiclasse (Random-Multiclasse) come ultimo della serie. Come è facile intuire quest'ultimo è quello più importante dell'intero sistema perché ad esso vengono demandate la maggior parte delle decisioni. Infatti, visionando i risultati in figura 4.8 ottenuti dalla classificatore random-multiclasse da solo, si può notare come non ci sia una sostanziale differenza tra questi risultati e quelli ottenuti con gli MCS sopra descritti.

```
Class report:
              precision    recall  f1-score      support
conficker      0.817061  0.640000  0.717032   135.000000
corebot        0.995544  0.991111  0.993308   135.000000
cryptolocker   0.711270  0.645926  0.675757   135.000000
dircrypt       0.529696  0.445926  0.482704   135.000000
emotet         0.988300  0.997037  0.992642   135.000000
fobber         0.748752  0.911111  0.820803   135.000000
gozi           0.758006  0.720000  0.736818   135.000000
kraken         0.860869  0.767407  0.810240   135.000000
matsnu         0.914776  0.832593  0.870398   135.000000
murofet        0.855008  0.800000  0.824911   135.000000
necurs         0.793930  0.642963  0.710372   135.000000
nymaim         0.809359  0.494815  0.609968   135.000000
padcrypt       0.997003  0.989630  0.993286   135.000000
pushdo         0.900165  0.965926  0.930935   135.000000
pykspa         0.916544  0.921481  0.918537   135.000000
qadars         0.995499  0.977778  0.986516   135.000000
ramdo          0.977115  0.992593  0.984656   135.000000
ramnit         0.401089  0.339259  0.359104   135.000000
ranbyus        0.821866  0.856296  0.836150   135.000000
rovnix         0.979064  0.957037  0.967464   135.000000
simda          0.922234  0.945185  0.933481   135.000000
suppobox       0.758671  0.794074  0.773469   135.000000
symmi          0.986935  0.998519  0.992658   135.000000
tinba          0.642699  0.760000  0.694125   135.000000
vawtrak        0.936146  0.897778  0.911030   135.000000
alexa          0.918830  0.952119  0.935110  3375.000000

              precision   recall   f1-score   support
accuracy       0.000000  0.000000  0.881748  6750.000000
macro avg      0.843709  0.816791  0.825441  6750.000000
weighted avg   0.879767  0.881748  0.878082  6750.000000

Overall accuracy = 0.8817481481481482

True Positives: 5951.8
```

Figura 4.8. Risultati ottenuti con random-multiclasse da solo.

Si ottiene invece un notevole risultato con il terzo esperimento migliore anche del miglior modello costruito nei lavori precedenti. Tuttavia, pecca molto nella classificazione di specifici tipi di malware come i «dircrypt» e i «ramnit» come è possibile vedere in figura 4.3.

In questi casi è dunque interessante domandarsi come queste famiglie di malware vengano effettivamente classificate, nello specifico capire se vengano classificate sempre come malware o se vengano confusi per nomi di dominio benevoli. Nel secondo caso l'errore sarebbe, ovviamente, molto più grave.

Per cercare di rispondere a questa domanda si può far riferimento alle matrici di confusione.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	92	0	0	...	0	0	33
corebot	0	135	0	...	0	0	0
cryptolocker	0	0	103	...	13	0	4
dircrypt	0	0	3	...	2	0	11
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	9
gozi	0	0	0	...	0	0	21
kraken	0	0	0	...	1	0	14
matsnu	0	0	0	...	0	0	7
murofet	0	0	4	...	0	0	2
necurs	6	0	3	...	3	0	8
nymaim	0	0	0	...	0	0	43
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	3
pykspa	0	0	0	...	1	0	8
qadars	0	0	0	...	1	0	3
ramdo	0	0	0	...	0	0	0
ramnit	0	0	3	...	8	0	15
ranbyus	0	0	1	...	1	0	0
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	1
suppobox	0	0	0	...	0	0	8
symmi	0	0	0	...	0	0	0
tinba	0	0	10	...	115	0	3
vawtrak	1	0	0	...	0	130	3
alexa	10	0	0	...	0	1	3334

Figura 4.9. Matrice di confusione del terzo modello (figura: 3.14) nel primo fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	99	0	1	...	0	0	32
corebot	0	134	0	...	0	0	0
cryptolocker	0	0	103	...	6	0	3
dircrypt	0	0	2	...	3	0	9
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	9
gozi	0	0	0	...	0	0	26
kraken	0	0	1	...	2	0	12
matsnu	0	0	0	...	0	0	4
murofet	0	0	10	...	0	0	1
necurs	10	0	5	...	0	0	4
nymaim	0	0	0	...	1	0	50
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	3
pykspa	0	0	0	...	0	0	5
qadars	0	0	0	...	0	0	3
ramdo	0	0	0	...	0	0	0
ramnit	0	0	6	...	11	0	10
ranbyus	0	0	1	...	0	0	2
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	3
suppobox	0	0	0	...	0	0	5
symmi	0	0	0	...	0	0	0
tinba	0	0	16	...	100	0	7
vawtrak	0	0	0	...	0	123	11
alexa	5	0	2	...	0	1	3313

Figura 4.10. Matrice di confusione del terzo modello (figura: 3.14) nel secondo fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	92	0	0	...	0	0	34
corebot	0	135	0	...	0	0	0
cryptolocker	0	0	101	...	6	0	1
dircrypt	0	0	4	...	6	0	10
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	10
gozi	0	0	0	...	0	0	24
kraken	0	0	0	...	3	0	13
matsnu	0	0	0	...	0	0	5
murofet	0	0	12	...	0	0	0
necurs	7	0	4	...	5	0	5
nymaim	1	0	0	...	0	0	43
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	7
pykspa	0	0	0	...	0	0	11
qadars	0	0	0	...	0	0	0
ramdo	0	0	0	...	0	0	0
ramnit	0	0	6	...	12	0	14
ranbyus	0	0	0	...	0	0	1
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	8
suppobox	0	0	0	...	0	0	5
symmi	0	0	0	...	0	0	0
tinba	0	0	22	...	102	0	4
vawtrak	0	0	0	...	0	125	10
alexa	9	0	0	...	0	2	3313

Figura 4.11. Matrice di confusione del terzo modello (figura: 3.14) nel terzo fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	102	0	0	...	0	0	25
corebot	0	134	0	...	0	0	0
cryptolocker	0	0	101	...	6	0	7
dircrypt	0	0	2	...	5	0	7
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	5
gozi	0	0	0	...	0	0	20
kraken	0	0	1	...	3	0	17
matsnu	0	0	0	...	0	0	11
murofet	0	0	13	...	0	0	0
necurs	5	0	5	...	2	0	6
nymaim	1	0	1	...	0	0	36
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	5
pykspa	0	0	0	...	0	0	4
qadars	0	0	1	...	0	0	2
ramdo	0	0	0	...	0	0	0
ramnit	0	0	5	...	14	0	9
ranbyus	0	0	0	...	0	0	1
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	6
suppobox	0	0	0	...	0	0	4
symmi	0	0	0	...	0	0	0
tinba	0	0	22	...	106	0	2
vawtrak	0	0	0	...	0	125	10
alexa	4	0	1	...	1	1	3306

Figura 4.12. Matrice di confusione del terzo modello (figura: 3.14) nel quarto fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	99	0	0	...	0	0	31
corebot	0	133	0	...	0	0	0
cryptolocker	0	0	108	...	8	0	3
dircrypt	0	0	3	...	3	0	9
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	7
gozi	0	0	0	...	0	0	28
kraken	1	0	0	...	1	0	15
matsnu	0	0	0	...	0	0	12
murofet	0	0	12	...	1	0	0
necurs	4	0	3	...	11	0	7
nymaim	0	0	0	...	0	0	47
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	6
pykspa	0	0	0	...	0	0	4
qadars	0	0	0	...	0	0	0
ramdo	0	0	0	...	0	0	0
ramnit	0	0	4	...	4	0	14
ranbyus	0	0	1	...	1	0	1
rovnix	0	1	0	...	0	0	0
simda	0	0	0	...	0	0	4
suppobox	0	0	0	...	0	0	5
symmi	0	0	0	...	0	0	0
tinba	0	0	9	...	106	0	4
vawtrak	1	0	0	...	0	127	6
alexa	6	0	1	...	1	3	3313

Figura 4.13. Matrice di confusione del terzo modello (figura: 3.14) nel quinto fold.

Da esse (4.9, 4.10, 4.11, 4.12, 4.13) si è notato come, in media, il 6.9% di dircrypt e il 9.2% di ramnit vengano classificati come alexa.

L'ultimo esperimento dimostra come lo Stacking sia quello che offre le migliori prestazioni in assoluto primeggiando su tutte le metriche con un miglioramento di circa il 2% rispetto ai risultati ottenuti nei lavori precedenti. Ha, di contro, il problema di essere computazionalmente molto pesante e di avere anch'esso problemi nella classificazione delle famiglie "dicrypt" e "ramnit" (figura 4.4). Anche in questo caso si può far riferimento alle matrici di confusione (4.14, 4.15, 4.16, 4.17, 4.18) come nel modello precedente.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	96	0	0	...	0	0	25
corebot	0	135	0	...	0	0	0
cryptolocker	0	0	98	...	15	0	3
dircrypt	0	0	2	...	2	0	4
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	5
gozi	0	0	0	...	0	0	16
kraken	0	0	0	...	1	0	11
matsnu	0	0	0	...	0	0	5
murofet	0	0	4	...	0	0	2
necurs	5	0	3	...	2	0	7
nymaim	0	0	0	...	0	0	33
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	4
pykspa	0	0	0	...	1	0	6
qadars	0	0	0	...	1	0	4
ramdo	0	0	0	...	0	0	0
ramnit	0	0	11	...	7	0	7
ranbyus	0	0	0	...	1	0	0
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	2
suppobox	0	0	0	...	0	0	5
symmi	0	0	0	...	0	0	0
tinba	0	0	7	...	120	0	2
vawtrak	1	0	0	...	0	131	2
alexa	16	0	1	...	0	1	3323

Figura 4.14. Matrice di confusione del modello basato su Stacked Genralization nel primo fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	102	0	0	...	0	0	25
corebot	0	134	0	...	0	0	0
cryptolocker	0	0	96	...	7	0	2
dircrypt	0	0	0	...	2	0	6
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	5
gozi	0	0	0	...	0	0	22
kraken	0	0	1	...	1	0	7
matsnu	0	0	0	...	0	0	7
murofet	0	0	12	...	0	0	0
necurs	9	0	4	...	0	0	4
nymaim	0	0	0	...	1	0	32
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	3
pykspa	0	0	0	...	0	0	4
qadars	0	0	0	...	0	0	3
ramdo	0	0	0	...	0	0	0
ramnit	0	0	3	...	9	0	5
ranbyus	0	0	0	...	0	0	1
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	4
suppobox	0	0	0	...	0	0	6
symmi	0	0	0	...	0	0	0
tinba	0	0	15	...	100	0	6
vawtrak	0	0	0	...	0	127	7
alexa	7	0	0	...	2	1	3312

Figura 4.15. Matrice di confusione del modello basato su Stacked Generalization nel secondo fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	98	0	0	...	0	0	28
corebot	0	135	0	...	0	0	0
cryptolocker	0	0	91	...	8	0	0
dircrypt	0	0	3	...	5	0	7
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	7
gozi	0	0	0	...	0	0	20
kraken	0	0	1	...	3	0	12
matsnu	0	0	0	...	0	0	5
murofet	0	0	7	...	1	0	0
necurs	6	0	3	...	4	0	3
nymaim	1	0	0	...	0	0	29
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	10
pykspa	0	0	0	...	0	0	10
qadars	0	0	0	...	0	0	0
ramdo	0	0	0	...	0	0	1
ramnit	0	0	5	...	14	0	9
ranbyus	0	0	0	...	0	0	1
rovnix	0	0	0	...	0	0	0
simda	0	0	0	...	0	0	8
suppobox	0	0	0	...	0	0	3
symmi	0	0	0	...	0	0	0
tinba	0	0	17	...	109	0	2
vawtrak	0	0	0	...	0	125	10
alexa	11	0	0	...	0	1	3310

Figura 4.16. Matrice di confusione del modello basato su Stacked Genralization nel terzo fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	107	0	0	...	0	0	22
corebot	0	134	0	...	0	0	0
cryptolocker	0	0	95	...	14	0	4
dircrypt	0	0	2	...	8	0	5
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	3
gozi	0	0	0	...	0	0	16
kraken	0	0	1	...	1	0	11
matsnu	0	0	0	...	0	0	11
murofet	0	0	10	...	0	0	1
necurs	6	0	4	...	1	0	6
nymaim	1	0	0	...	1	0	30
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	5
pykspa	0	0	0	...	0	0	2
qadars	0	0	1	...	0	0	1
ramdo	0	0	0	...	0	0	0
ramnit	0	0	5	...	13	0	7
ranbyus	0	0	1	...	1	0	1
rovnix	0	0	0	...	0	0	1
simda	0	0	0	...	0	0	5
suppobox	0	0	0	...	0	0	5
symmi	0	0	0	...	0	0	0
tinba	0	0	19	...	109	0	0
vawtrak	0	0	0	...	0	129	6
alexa	4	0	1	...	1	1	3315

Figura 4.17. Matrice di confusione del modello basato su Stacked Genralization nel quarto fold.

Confusion matrix:

	conficker	corebot	cryptolocker	...	tinba	vawtrak	alexa
conficker	106	0	0	...	0	0	24
corebot	0	133	0	...	0	0	0
cryptolocker	0	0	101	...	11	0	3
dircrypt	0	0	3	...	3	0	6
emotet	0	0	0	...	0	0	0
fobber	0	0	0	...	0	0	7
gozi	0	0	0	...	0	0	20
kraken	0	0	0	...	1	0	13
matsnu	0	0	0	...	0	0	7
murofet	0	0	10	...	1	0	0
necurs	3	0	2	...	8	0	5
nymaim	0	0	0	...	0	0	35
padcrypt	0	0	0	...	0	0	0
pushdo	0	0	0	...	0	0	5
pykspa	0	0	0	...	0	0	4
qadars	0	0	0	...	0	0	0
ramdo	0	0	0	...	0	0	0
ramnit	0	0	6	...	6	0	9
ranbyus	0	0	1	...	0	0	0
rovnix	0	1	0	...	0	0	0
simda	0	0	0	...	0	0	3
suppobox	0	0	0	...	0	0	8
symmi	0	0	0	...	0	0	0
tinba	0	0	11	...	107	0	4
vawtrak	1	0	0	...	0	128	5
alexa	10	0	1	...	0	3	3309

Figura 4.18. Matrice di confusione del modello basato su Stacked Genralization nel quinto fold.

Le percentuali ottenute sono: 4.2% per i dircrypt e 5.5% per i ramnit.

Con questo stesso metodo è possibile anche valutare altre famiglie di DGA. Si è notato che alcune di esse, sebbene abbiano valori migliori nelle metriche rispetto a dircrypt e ramnit, vengono più spesso confuse con alexa. Ad esempio per lo stacking le percentuali delle peggiori famiglie sono:

- Conficker: 18.4%
- Gozi: 14%
- Kraken: 8%
- Nymaim: 23.6%

Per il terzo modello invece i risultati sono:

- Conficker: 23%
- Gozi: 17.7%
- Kraken: 10.5%
- Nymaim: 32.4%

In conclusione si possono ritenere validi i modelli ELMo"Multi" + FastText (Mono/Bi/Tri)"Multi" + Random"Multi" e Stacking, mentre sono da scartare i modelli ELMo"Bin"-FastText"Bin"-Random"Multi" e (ELMo"Bin"+FastText"Bin")-Random"Multi" dati i risultati non sufficienti.

Da questi risultati possiamo inoltre individuare come, quando si utilizza sempre il medesimo dataset per l'addestramento dei modelli di base, la topologia in parallelo sia quella migliore (figure: 3.14, 3.15): l'utilizzo di una struttura in serie pone troppo peso sull'ultimo classificatore, questo si traduce in performance molto vicine a quelle che quest'ultimo avrebbe preso singolarmente rendendo di fatto inutile l'utilizzo degli MCS. Quella in parallelo, al contrario, permette di mitigare i punti di debolezza

e i bias di ogni classificatore portando a decisione migliori. Il fusore basato su average consente di selezionare la classe che ha la maggiore probabilità di essere quella giusta basandosi sui risultati di tutti i classificatori rendendo la decisione più solida e sicura. Il fusore basato su stacking permette di trarre vantaggio da tutti i bias dei modelli di base.

Come già scritto tuttavia rimangono ancora problemi nella classificazione di alcune famiglie per tutti gli MCS, dunque, probabilmente, il problema non dipende da come gli MCS vengono costruiti ma dai modelli di base che non riescono a fornire buone prestazioni per queste famiglie.

Per capire meglio perché si hanno queste difficoltà nella classificazione bisogna individuare quali sono le caratteristiche di queste famiglie. Alcuni esempi:

- **nymaim**[6]: utilizza un DGA wordlist-based con un seed tempo-dipendente. In particolare, si basa su quattro wordlist: una con parole inglesi con l'iniziale che va dalla R alla Z, una seconda che contiene due separatori (la stringa vuota o il simbolo "-"), la terza è una lista, sempre di parole inglesi, che iniziano per lettere che vanno da C a R e l'ultima lista contiene una serie di domini di primo livello. Dall'unione di elementi di queste liste vengono creati i nomi di dominio.
- **gozi**: questo malware è peculiare perchè il suo DGA genera nomi di dominio utilizzando parole latine.
- **ramnit**[10]: il DGA genera i nomi di dominio utilizzando un algoritmo LCG con un seed "hardcoded".
- **dircrypt**[1]: anche dircrypt, come ramnit, si basa su un DGA che fa uso di un hardcoded seed con un algoritmo LCG per generare i nomi di dominio.

La famiglia "nymaim" genera nomi di dominio con forme simili a quelle dei nomi benevoli, l'unione di questa caratteristica a un seed tempo-dipendente li rende difficili da distinguere. Per questo motivo, probabilmente, risulta essere la famiglia più frequentemente confusa con "alexa".

Per quanto riguarda "gozi", a renderne difficile la classificazione, è l'utilizzo di wordlist con parole latine, infatti ne ELMo ne FastText sono addestrati con parole di questa lingua.

Con "dircrypt" e "ramnit" il problema è probabilmente diverso dai primi due. Infatti, nonostante i pessimi risultati indicati dalle metriche, i nomi di dominio che generano non vengono scambiati per benevoli così frequentemente. Si può presumere che, data la somiglianza del DGA utilizzato, molti nomi di dominio generati dai "dircrypt" vengano scambiati per "ramnit" e viceversa. Per confermare questa ipotesi, tuttavia, si necessitano di ulteriori esperimenti e l'utilizzo di matrici di confusione complete che permettano di capire meglio la distribuzione nella classificazione di queste due famiglie.

5

Conclusioni

In questa tesi abbiamo discusso le Botnet, i DGA, il Machine Learning, i sistemi multi-classificatori partendo da un problema, proponendo delle soluzioni per poi sperimentare queste soluzioni confrontandone i risultati.

I sistemi multi-classificatori sono la strada percorsa per poter creare dei modelli sempre più potenti per combattere le Botnet basate sui DGA. Confrontare questi sistemi ha permesso di individuare i migliori, quelli più adatti a questo problema. Si è notato che a fare la differenza, nel nostro contesto specifico, è la struttura del sistema ovvero quella in parallelo. Basandosi sui risultati ottenuti è il confronto tra le decisioni la soluzione migliore, come accade spesso in molti altri contesti. Questo lavoro va anche oltre, indica anche come fare questo confronto identificando nello Stacking la modalità migliore fra quelle studiate.

L'analisi sperimentale non porta alla luce solo i punti di forza ma, fortunatamente, anche quelli di debolezza. Tutti gli MCS costruiti hanno difficoltà nella classificazione di particolari famiglie per le quali questi modelli sembrano, stando ai dati, essere poco efficaci.

A partire da queste basi è possibile immaginare diversi sviluppi e miglioramenti. Si possono sperimentare nuovi fusori, si possono costruire nuovi modelli di base in grado di filtrare con maggior precisione le famiglie di DGA sulle quali non si sono ancora ottenuti risultati soddisfacenti. Questo può essere ottenuto sia creando modelli totalmente nuovi, sia utilizzando quelli già presenti con un dataset di addestramento specifico per queste famiglie in modo tale da renderli molto "bravi" a classificarle. Ad esempio, si potrebbe addestrare lo strato di embedding di un modello su parole latine per imparare a riconoscere meglio la famiglia "gozi". Oppure, per quanto riguarda "nymaim", una delle altre famiglie problematiche insieme a "gozi", essendo un malware con un DGA che utilizza liste di parole inglesi, si potrebbero individuare nuovi modelli con strato di embedding basato e addestrato su parole inglesi o migliorare quelli già presenti, ovvero ELMo. Potenziare modelli in questo verso sarebbe dunque la strada da seguire per affrontare meglio questa famiglia, tuttavia, utilizzando un seed tempo-dipendente, sarà sempre difficile da classificare e, ancor peggio, da distinguere dai nomi benevoli.

Un'altra direzione da seguire è testare i modelli su un dataset reale preso, ad esempio, dalla rete GARR per poter capire il loro comportamento "sul campo".

Riferimenti bibliografici

1. Johannes Bader. The dga of dircrypt. Accessed: 2021-10-08.
2. Cerioni C. Un sistema a classificatori multipli basato su pre-trained embeddings per il riconoscimento di dga. Master's thesis, Università Politecnica delle Marche, 2020.
3. Wikimedia Commons. File:lstm cell.svg — wikimedia commons, the free media repository, 2021. [Online; accessed 11-October-2021].
4. Wikimedia Commons. File:multilayer neural network.png — wikimedia commons, the free media repository, 2021. [Online; accessed 11-October-2021].
5. Christopher C Elisan. *Malware, rootkits & botnets: A beginner's guide*. McGraw-Hill Education, 2012.
6. johannesbader. The new domain generation algorithm of nymaim. Accessed: 2021-10-05.
7. Zhenzhu Meng, Yating Hu, and Christophe Ancey. Using a data driven approach to predict waves generated by gravity driven mass flows. *Water*, 12(2), 2020.
8. D. Nanda, P. Wadhwa, S. Singh, and D. Kumar. Botnet: Lifecycle, architecture and detection model. In *IJLTEMAS*, pages 28–32, 2014.
9. Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. A comprehensive measurement study of domain generating malware. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 263–278, 2016.
10. Michał Praszmo. Ramnit – in-depth analysis. Accessed: 2021-10-08.
11. Yanchen Qiao, Bin Zhang, Weizhe Zhang, Arun Kumar Sangaiah, and Hualong Wu. Dga domain name classification method based on long short-term memory with attention mechanism. *Applied Sciences*, 9(20):4205, 2019.
12. R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro. Analysis of botnets through life-cycle. In *Proceedings of the International Conference on Security and Cryptography*, pages 257–262, 2011.
13. M Paz Sesmero, Agapito I Ledezma, and Araceli Sanchis. Generating ensembles of heterogeneous classifiers using stacked generalization. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):21–34, 2015.
14. Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran, and Linh Giang Nguyen. A lstm based framework for handling multiclass imbalance in dga botnet detection. *Neurocomputing*, 275:2401–2413, 2018.
15. Wikipedia. Botnet — wikipedia, l'enciclopedia libera, 2021. [Online; in data 4-ottobre-2021].
16. Wikipedia. Rete neurale ricorrente — wikipedia, l'enciclopedia libera, 2021. [Online; in data 11-ottobre-2021].

17. David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.
18. Jonathan Woodbridge, Hyrum S Anderson, Anjum Ahuja, and Daniel Grant. Predicting domain generation algorithms with long short-term memory networks. *arXiv preprint arXiv:1611.00791*, 2016.
19. Michał Woźniak, Manuel Grana, and Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3–17, 2014.
20. Mattia Zago, Manuel Gil Pérez, and Gregorio Martínez Pérez. Scalable detection of botnets based on dga. *Soft Computing*, 24(8):5517–5537, 2020.
21. Mohammed Zahid, Abdelhamid Belmekki, and Abdellatif Mezrioui. A new architecture for detecting ddos/brute forcing attack and destroying the botnet behind. pages 899–903, 05 2012.