



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Rilevazione di malware basati su DGA attraverso un classificatore LSTM con embedding basato su FastText

**DGA-based malware detection through a LSTM classifier with
FastText-based embedding**

Candidato:

Lorenzo Tiseni

Relatore:

Prof. Luca Spalazzi

Correlatori:

Prof. Alessandro Cucchiarelli

Prof. Christian Morbidoni

Anno Accademico 2020-2021



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Rilevazione di malware basati su DGA attraverso un classificatore LSTM con embedding basato su FastText

**DGA-based malware detection through a LSTM classifier with
FastText-based embedding**

Candidato:

Lorenzo Tiseni

Relatore:

Prof. Luca Spalazzi

Correlatori:

Prof. Alessandro Cucchiarelli

Prof. Christian Morbidoni

Anno Accademico 2020-2021

Indice

Introduzione	1
1 Apprendimento automatico per il rilevamento di malware basati su DGA	3
1.1 DNS servers e Domain name	3
1.2 Botnets & DGA	7
1.2.1 Botnets: una minaccia alla sicurezza in rete	7
1.2.2 Meccanismi di protezione delle botnet: I DGA	10
1.3 Machine Learning	14
1.3.1 Reti Neurali	14
1.3.2 LSTM	17
1.3.3 Machine Learning per i DGA	20
1.4 FastText	25
2 Materiali e Metodi	27
2.1 Risorse	27
2.1.1 Log DNS rete GARR	27
2.1.2 Bambenek Feed	29
2.1.3 Majestic Million	30
2.2 Datasets	32
2.2.1 Dataset per il training di FastText & Risultati	32
2.2.2 Dataset per il test della rete LSTM.MI	37
2.3 Configurazioni Sperimentali	40
2.3.1 Embedding Randomico	41
2.3.2 Embedding basato su FastText	42

Indice

2.3.3	Addestramento del classificatore	43
3	Risultati sperimentali	45
3.1	Esperimenti con Dataset da 4200 nomi di dominio	45
3.2	Esperimenti con Dataset da 8400 nomi di dominio	48
3.3	Esperimenti con Dataset da 16800 nomi di dominio	51
3.4	Discussione	55
4	Conclusioni e Sviluppi Futuri	65
5	Appendice	69

Elenco delle figure

1.1	Struttura matematica di un neurone artificiale[1]	15
1.2	Struttura rete neurale artificiale[2]	16
1.3	Struttura rete neurale ricorrente[3]	17
2.1	Work-flow per la creazione del dataset per FastText	34
2.2	Prime righe del dataset usato per l'addestramento dello strato di embedding FastText basato su caratteri	35
2.3	Prime righe del dataset usato per l'addestramento dello strato di embedding FastText basato su bigrammi	35
2.4	Prime righe del dataset usabile per l'addestramento dello strato di embedding FastText basato su trigrammi	36
2.5	File di embedding per i caratteri ottenuto dal training di FastText .	36
2.6	File di embedding per i bigrammi ottenuto dal training di FastText .	37
2.7	Work-flow per la creazione del dataset per il classificatore LSTM . .	39
3.1	Report finale esperimento con Random Embedding	45
3.2	Report finale esperimento con FastText Embedding basato su caratteri	46
3.3	Report finale esperimento con FastText Embedding basato su bigrammi	47
3.4	Report finale esperimento con Random Embedding	49
3.5	Report finale esperimento con FastText Embedding basato su caratteri	50
3.6	Report finale esperimento con FastText Embedding basato su bigrammi	51
3.7	Report finale esperimento con Random Embedding	52
3.8	Report finale esperimento con FastText Embedding basato su caratteri	53
3.9	Report finale esperimento con FastText Embedding basato su bigrammi	54

Elenco delle figure

3.10	Andamento dell'accuracy complessiva al variare della dimensione del dataset e dello strato di embedding	56
3.11	Analisi del miglioramento dell'accuracy complessiva al variare delle dimensioni del dataset	58
3.12	Analisi del miglioramento dell'accuracy complessiva al variare dello strato di embedding	59
3.13	Valori di F1-score ottenuti per la famiglia Vawtrak	60
3.14	Valori di F1-score ottenuti per la famiglia Fobber	61
3.15	Valori di F1-score ottenuti per la famiglia Ramnit	61
3.16	Valori di F1-score ottenuti per la famiglia Nymaim	62
3.17	Analisi percentuale di nomi Nymaim reputati come benevoli	63
5.1	Matrice di Confusione ottenuta con Random Embedding e Dataset da 4200 nomi di dominio	70
5.2	Matrice di Confusione ottenuta con embedding FastText basato su caratteri e Dataset da 4200 nomi di dominio	71
5.3	Matrice di Confusione ottenuta con embedding FastText basato su bigrammi e Dataset da 4200 nomi di dominio	72
5.4	Matrice di Confusione ottenuta con Random Embedding e Dataset da 8400 nomi di dominio	73
5.5	Matrice di Confusione ottenuta con embedding FastText basato su caratteri e Dataset da 8400 nomi di dominio	74
5.6	Matrice di Confusione ottenuta con embedding FastText basato su bigrammi e Dataset da 8400 nomi di dominio	75
5.7	Matrice di Confusione ottenuta con Random Embedding e Dataset da 16800 nomi di dominio	76
5.8	Matrice di Confusione ottenuta con embedding FastText basato su caratteri e Dataset da 16800 nomi di dominio	77
5.9	Matrice di Confusione ottenuta con embedding FastText basato su bigrammi e Dataset da 16800 nomi di dominio	78

Introduzione

Le grandi possibilità offerte dalla rete Internet al giorno d'oggi, sono sotto gli occhi di tutti. Rispetto a 30 anni fa, infatti, è possibile comunicare in tempi istantanei con persone e luoghi, distanti migliaia di chilometri geograficamente. Inoltre, il web si è sempre più arricchito nel tempo di contenuti informativi, che permettono a chiunque dotato di conoscenze di base sufficienti, di approfondire qualsiasi argomento e, migliorare le proprie abilità. Questa cosa è diventata ancora più importante negli ultimi due anni, quando a seguito della pandemia da Covid-19, è aumentato ancora di più il numero di persone connesse alla rete e il numero di corsi o tutorial online disponibili per gli utenti.

Purtroppo, tutte queste grandi potenzialità che possiamo ammirare e sfruttare ogni giorno, non sono esenti da controindicazioni e pericoli. Il fatto che ci siano tante persone connesse alla rete, aumenta il numero di bersagli che un attaccante può andare a minacciare. Infatti il numero di attacchi, non solo a singole macchine private, ma anche a sistemi appartenenti enti pubblici e governativi, nell'ultimo periodo è significativamente cresciuto. Questo fenomeno, ha dato la possibilità agli hacker di creare facilmente delle reti di computer zombie, che potessero essere sfruttate per compiere azioni criminose, e interrompere alcuni servizi cruciali presenti sul web. Oltretutto, la presenza di queste reti è molto difficile da individuare, in quanto, spesso i possessori delle macchine, si accorgono molto in ritardo di essere stati vittima di un attacco.

Proprio a causa di questi fenomeni sono state sviluppate tutta una serie di tecniche nel corso degli anni che puntassero, non solo a prevenire le infezioni sui singoli PC, ma anche a disattivare queste reti, impedendo all'attaccante di comunicare con i

Introduzione

computer infetti. Questo lavoro quindi, si propone l'obiettivo di raffinare una tecnica di deep learning, già esistente, per la classificazione dei domini, generati da algoritmi DGA, in maniera tale da oscurarli e impedire all'attaccante di comunicare con i PC infetti. Lo scopo, spinti anche dai risultati ottenuti in lavori precedenti[4][5], è quello di dimostrare che, con uno strato di embedding basato su FastText e addestrato con dati reali, una rete neurale LSTM, usata come classificatore, riesce a riconoscere meglio i nomi di dominio DGA, in maniera tale da oscurarli tempestivamente.

Il lavoro di tesi presentato, si dividerà in varie sezioni o capitoli. Nel primo capitolo forniremo una panoramica sui concetti di base, necessari alla comprensione degli esperimenti: si analizzerà come funziona un DNS server, come agiscono le botnet e quali sono i loro meccanismi di protezione (tra cui i DGA), che cosa si intende per rete neurale, machine learning e come questi concetti si possono applicare al nostro problema, e che cos'è FastText. Il secondo capitolo illustrerà quali risorse sono state utilizzate nel corso del lavoro, e come esse sono state sfruttate per costruire i dataset per gli esperimenti. Poi si focalizzerà sulle varie parti che compongono le configurazioni sperimentali, quali lo strato di embedding e il processo di addestramento del classificatore. Nel terzo capitolo verranno analizzati tutti i risultati sperimentali, confrontando le prestazioni ottenute con l'embedding basato su FastText, con quelle ottenute senza questo strato di embedding. Infine nel quarto capitolo verranno discusse brevemente le conclusioni, e proposti alcuni sviluppi futuri.

Capitolo 1

Apprendimento automatico per il rilevamento di malware basati su DGA

1.1 DNS servers e Domain name

Iniziamo la descrizione di questo lavoro con la presentazione di alcuni concetti chiave, necessari alla comprensione sia dello scopo del lavoro stesso, che della metodologia seguita per realizzarlo.

Oggigiorno ogni server, o servizio, della rete Internet è identificato da un indirizzo univoco che permette all'utente di localizzarlo sul web, fargli delle richieste, e ottenere in risposta delle informazioni. Questo indirizzo è un indirizzo numerico che viene detto indirizzo IP. Un indirizzo IP è un numero del pacchetto IP che identifica univocamente un dispositivo detto host collegato a una rete informatica che utilizza l'Internet Protocol come protocollo di rete per l'indirizzamento tramite appunto il protocollo IP[6]. L'host tipicamente sarà un "nodo" connesso alla rete, una macchina in cui risiederà il web server associato all'indirizzo IP cercato. Un indirizzo IP può essere o in formato IPv4 o in formato IPv6:

- Nel formato IPv4 in ogni indirizzo IP abbiamo 4 byte separati da un punto(notazione dotted) che rappresentano dei numeri, che tipicamente sono convertiti in formato decimale per essere più comprensibili dalle persone.

- Nel formato IPv6, nato proprio perchè gli indirizzi IP a 32 bit sembrano non bastare più per tutte le macchine connesse alla rete, invece di 4 byte abbiamo 16 byte che rappresentano dei numeri.

L'indirizzo IP, comunque anche se costituisce un ottimo identificativo, univoco e ben formato, per una macchina connessa alla rete internet non è facile da ricordare per una persona, specialmente se non esperta. Per questo motivo il 23 giugno 1983 Paul Mockapetris, Jon Postel e Craig Partridge idearono il concetto di Domain Name System(DNS). Il Domain Name System è un sistema utilizzato per assegnare nomi ai nodi della rete (in inglese: host). Questi nomi sono utilizzabili, mediante una traduzione, di solito chiamata "risoluzione", al posto degli indirizzi IP originali. Il servizio è realizzato tramite un database distribuito, costituito dai server DNS.[7] Il DNS quindi diventa uno strumento utilissimo per gli utenti stessi e anche per gli amministratori della rete internet. Dal lato utente esso consente facilmente di ricercare sul web un server a cui richiedere del contenuto informativo, digitando il nome assegnato al nodo corrispondente a quel server che è detto nome di dominio. Dal lato amministratore invece il domain name system risulta essere un ottimo sistema per distribuire i server che offrono un servizio per un unico sito web, o bypassare nodi della rete che sono temporaneamente guasti evitando così di interrompere il servizio

Quando un utente digita il nome di dominio assegnato a un nodo della rete che vuole raggiungere per effettuare una richiesta, avviene un processo detto Risoluzione DNS. Il DNS infatti è un database distribuito in cui ad ogni indirizzo IP viene associato un nome di dominio, nel caso più semplice. Quando avviene una chiamata a un certo nome di dominio, viene fatta una query nel DNS server che ricerca l'indirizzo IP associato e risale così alla macchina cercata. Questa query è un processo complesso che si articola in varie fasi a seconda anche della forma del nome di dominio cercato

I nomi di dominio generalmente sono delle stringhe separate da punti e organizzate a livelli. Ad esempio il nome di dominio `it.overleaf.com` è organizzato su tre livelli. Tra questi il livello più importante è rappresentato dalla stringa che si incontra partendo da destra che viene detta per questo Top Level Domain o TLD. Segue

poi il dominio di secondo livello, che deve rispettare delle regole formali imposte come avere una determinata lunghezza e contenere solo dei caratteri appartenenti ad un certo set. Il terzo livello a sua volta è detto dominio di terzo livello e nel caso generale può essere seguito anche da livelli superiori.

Tale organizzazione in livelli è collegata direttamente al meccanismo di query implementato nel domain name system. Le query che avvengono nei DNS sono dette ricorsive o iterative:

- Nel caso di query iterativa quando un utente fa una richiesta per raggiungere un determinato nome di dominio prima viene contattato, il DNS server detto root server, riferito al punto con cui termina il nome di dominio, e lo si interroga per avere il riferimento al server che gestisce il TLD. Con la risposta ottenuta si interroga il DNS server che gestisce il dominio di primo livello, per ottenere il riferimento al server che gestisce il dominio di secondo livello. Il processo si ripete iterativamente finché non sono stati risolti tutti i livelli e si ottenuto il riferimento al server "autoritativo" per l'intero nome di dominio
- Nel caso di query ricorsiva il meccanismo è simile, cioè si deve risolvere prima il nome di dominio di primo livello, poi quello di secondo livello e così via fino all'ultimo livello. A differenza del caso iterativo però nel caso ricorsivo è il server che gestisce il dominio di primo livello che interrogherà il server che gestisce il dominio di secondo livello che a sua volta interrogherà quello del livello successivo. La risposta sarà poi riportata al client nell'ordine inverso in cui sono avvenute le interrogazioni.

I nomi mnemonici o nomi di dominio rappresentano perciò un ottimo modo per gli utenti di collegarsi facilmente ai servizi nel web. Oltre a questo però il DNS server ha delle caratteristiche di scalabilità e distribuzione utile anche agli amministratori degli stessi server. Infatti una delle peculiarità del DNS è che allo stesso nome di dominio posso associare più indirizzi IP e viceversa. Nel primo caso questo è un ottimo sistema per distribuire il carico di lavoro relativo a uno stesso servizio su più server, in maniera tale da alleggerire il peso delle richieste, e evitare che a causa

del disservizio di un solo server si blocchi tutti il servizio, bypassando il guasto e reindirizzando la richiesta verso un altro IP associato al domain name. Nel secondo caso associare più nomi ad un unico indirizzo IP è utile per permettere allo stesso server di ospitare contemporaneamente più siti web.

Chiaramente tali caratteristiche del DNS fanno in modo che, tutto questo processo di risoluzione e associazione di IP a nomi di dominio avvenga in maniera totalmente trasparente ai client, semplificando la vita agli utenti. Per questo motivo ogni rete o sotto-rete connessa internet o possiede un proprio DNS server oppure punta a un DNS esterno messo a disposizione gratuitamente dall'infrastruttura internet. Nonostante la trasparenza all'utente, è possibile per i più esperti modificare il server DNS preimpostato e usarne un altro. Questo di solito viene fatto per avere una risposta DNS più veloce oppure per raggiungere dei siti oscurati dalle compagnie telefoniche sui loro DNS

Alla luce della struttura del DNS, è chiaro che nel momento in cui si volesse rendere impossibile all'utente accedere a un sito contenente contenuti illegali, basta solo impedire la risoluzione DNS di quel nome oscurandolo. In questo caso il server che ospita il sito non viene chiuso, semplicemente viene tagliato il ponte che lo rendeva accessibile all'utente. Questa attività di oscuramento dei nomi di dominio in un DNS server è detta "*blacklisting*", cioè il dominio viene inserito in una lista nera. Tale trattamento viene riservato di solito ai siti di streaming video, o siti che permettono di scaricare gratis programmi con licenza a pagamento, poiché violano le norme sul copyright, oppure ai siti che dispensano dei programmi malevoli che possono infettare un PC

Chiaramente per una navigazione in internet più sicura per tutti gli utenti sarebbe necessario cercare di oscurare il più possibile tutti i nomi di dominio che fanno riferimento a siti non sicuri, o a siti che danneggiano prodotti con licenza. Tale attività oggi è diventata sempre più difficile a causa delle problematiche che descriveremo in seguito, e per questo è necessario trovare una soluzione.

1.2 Botnets & DGA

1.2.1 Botnets: una minaccia alla sicurezza in rete

Con il progredire dell'informatica e soprattutto con la diffusione massiccia della rete internet, negli anni è proliferata la diffusione mediante quest'ultima di codici malevoli capaci di danneggiare PC usati per lo più da utenti non esperti. Questi codici malevoli, detti malware, hanno la capacità di infettare un singolo PC domestico e comprometterne l'uso in vari modi. Ad esempio un singolo malware potrebbe criptare tutti i dati sensibili in un PC, o acquisirli per trasmetterli all'attaccante mediante registrazione dello schermo, oppure cancellarli tutti e rendere il PC inutilizzabile. Tutte queste azioni però rimangono confinate al singolo PC e, possono essere prevenute attraverso un monitoraggio continuo dei processi attivi nel sistema operativo, oppure risolte usando un software antivirus.

Per quanto possa essere pericoloso l'attacco su un singolo computer, comunque esso rimane circoscritto e al massimo nuoce solo al malcapitato che ne è stato colpito. Oltre a questo genere di attacchi però ultimamente i pirati informatici hanno allargato il loro target, e invece di infettare un solo computer hanno pensato infettare contemporaneamente più computer. Questo tipo di attacco porta il cracker o pirata informatico a costituire delle vere e proprie reti di "*computer zombie*" che possono essere sfruttati per compiere attività illecite anche contro terzi, e non direttamente contro i proprietari dei computer infettati. Queste reti di computer zombie sono dette **botnet**.

Il termine botnet non nasce dichiaratamente con un connotazione negativa. Infatti originariamente era definita botnet una rete di computer che connessi insieme svolgevano dei task automaticamente senza l'intervento dell'utente[8]. Poichè però questa struttura viene usata in larga parte dagli attaccanti in senso malevolo, è chiaro che essa ha assunto una connotazione negativa. Per questo adesso si tende a definire una botnet come una rete di macchine compromesse, o infette, che viene coordinata a distanza da un attaccante per perseguire scopi malevoli[8]. Colui che si occupa di gestire la botnet e gli invia i comandi viene detto **botmaster**. Tali comandi sono

inviati attraverso una componente fondamentale della botnet che è il C&C della botnet.

La botnet principalmente è composta da due componenti chiave che di seguito sono descritte:

- **Componente Host:** tale componente della botnet, è rappresentata dalla macchine compromesse che sono passate sotto il controllo dell'attaccante, attraverso l'uso di un agente malevolo, generalmente un malware in formato eseguibile o DLL, che le ha infettate. Tali macchine ricevono comandi dal botmaster e eseguono attacchi, oppure inviano dati sensibili al botmaster
- **Componente di Rete:** tale componente della botnet, è una componente online utilizzata dal botmaster per comunicare con le macchine. Essa è composta da:
 - **Command and Control Channel:** Questa parte è la parte fondamentale di una botnet in quanto è quella che permette all'attaccante di comunicare con le macchine infette, tenerle aggiornate e ricevere dati da esse. Senza tale componente la botnet diventa inutile, in quanto l'attaccante perde il contatto con le macchine infette che diventano un "esercito" senza più comandante, e quindi non più in grado di eseguire attacchi. Chiudere il canale di controllo della botnet, significa infatti uccidere la botnet
 - **Server di Distribuzione:** è la componente della botnet in cui risiede il malware da usare per infettare i bot, e tutti gli aggiornamenti ad esso correlati per tenere la sua presenza all'interno del PC infetto nascosta. Essa viene usata nella fase di infezione e poi in seguito per aggiornare i malware nei bot.
 - **Drop Zone:** è una risorsa online in cui il botmaster salva tutti i dati sensibili che i bot gli inviano

Oltre a queste componenti di base, un'altra caratteristica fondamentale delle botnet è la **struttura**, da cui dipende anche il modo in cui il botmaster comunica

con i bot. La struttura più semplice di una botnet è la struttura **centralizzata**, in cui il server di comando e controllo è unico. Quindi tutti i bot appartenenti alla botnet sono connessi ad un unico nodo o server della rete internet, sotto il controllo dell'attaccante, e da lì ricevono i comandi. In questo tipo di struttura della botnet ci sono due modi in cui i comandi sono inviati ai bot dal C&C server. Il primo modo è detto *Push Style* e in esso è il botmaster che direttamente invia il comando attraverso il server di comando e controllo ai bot della rete. Il secondo modo è detto *Pull Style* e in questo caso il botmaster non invia direttamente il comando ai bot. Infatti sono i bot che periodicamente si connettono al server di comando e controllo, per controllare se ci sono nuovi comandi, inseriti dal botmaster. Mentre nel primo caso il comando arriva quasi in real time nel secondo caso invece il comando arriva ai bot con un certo ritardo.

Un altro tipo di struttura molto utilizzato per le botnet è la struttura di tipo **decentralizzato**. Con il tempo gli attaccanti si sono accorti che la struttura centralizzata, seppur molto semplice da utilizzare, rendeva la botnet molto più vulnerabile, in quanto una volta oscurato il C&C server la botnet diventava inutilizzabile. Nelle strutture decentralizzate ogni bot che fa parte della botnet può essere utilizzato sia come client che come server di comando e controllo. In queste botnet, che sono dette P2P botnet, se viene oscurato un server di comando e controllo, la botnet rimane ancora utilizzabile, perché l'attaccante può ancora dare ordini agli altri bot sfruttandone un altro. Anche nel caso di questa struttura i modi di inviare i comandi alla botnet sono due. Il primo modo è quello di tipo **push**, nel quale l'attaccante invia il comando a un nodo della botnet, che poi viene propagato da lì ai bot "vicini". I riceventi poi propagano il comando ai loro vicini e il processo va avanti così finché tutti non hanno ricevuto il comando. Il secondo modo è quello **pull**, in cui i nuovi comandi inviati dal botmaster sono ottenuti dai singoli nodi della rete andando ad interrogare, secondo certi criteri, altri nodi della rete in cui il botmaster ha pubblicato il comando

Oltre a queste due strutture, ognuna delle quali ha i suoi vantaggi e i suoi svantaggi, poiché i pirati informatici nel tempo hanno compreso che sotto certe condizioni

un'architettura può essere meglio dell'altra, esistono delle **strutture ibride**. Un esempio di struttura ibrida è quella utilizzata dalla combinazione di malware formata da ZeusP2P e Murofet, la quale usa una struttura decentralizzata come struttura primaria. Nel momento in cui l'attaccante non riesce a connettersi a uno dei nodi della rete, per impartire ordini si avvale di una struttura secondaria centralizzata con un unico C&C server[8].

Per concludere, la potenza e la pericolosità delle botnet, oltre che nei tipi di struttura che garantiscono vantaggi all'attaccante, risiedono anche nell'uso che viene fatto di esse. Uno degli usi più frequenti delle botnet sono gli attacchi DDoS: in questo tipo di attacco il botmaster ordina a tutti i bot della rete di intasare un server web con delle richieste continue, in maniera tale da o rendere il servizio inutilizzabile, oppure creare una vulnerabilità per rubare delle informazioni sugli utenti di quel servizio. Oltre a questo uso le botnet vengono utilizzate anche per l'invio massiccio di email spam; in tal caso tale pratica è molto sicura per l'attaccante, la cui identità rimane nascosta, in quanto saranno i bot a inviare le email. Oltre a questi per citarne altri abbiamo il guadagno illecito attraverso i click sugli ad pubblicitari e la raccolta di informazioni su larga scala.

1.2.2 Meccanismi di protezione delle botnet: I DGA

La parte nevralgica della botnet, come già accennato prima risulta essere il C&C server, e per questo è anche la parte della struttura che necessita di più protezione. Infatti se il C&C server della botnet viene individuato e oscurato l'intera botnet non è più utilizzabile dall'attaccante. Per questo gli hacker o botmaster hanno cominciato a sfruttare alcuni meccanismi di protezione che sono i seguenti:

- **Bulletproof Hosting:** Tale meccanismo di protezione è fornito da compagnie di hosting dei servizi web che non si curano di cosa viene caricato nei loro servizi. Infatti normalmente i servizi di web hosting nel momento in cui qualcuno carica al loro interno del materiale che viola il copyright, o del materiale pornografico, o del materiale contenente codici maligni, essi provvedono subito a sospendere l'account dell'utente che ha effettuato i caricamenti e rimuovono il materiale.

A volte però alcuni proprietari di servizi di web hosting anche se esistono questi termini di uso del servizio, li ignorano e in cambio di soldi permettono agli utenti di caricare qualsiasi cosa, anche del codice maligno. Tali servizi di bulletproof hosting per questo sono molto usati dagli attaccanti

- **Dynamic DNS:** Questo tipo di servizio è particolarmente appetibile per i cybercriminali, perché associa un nome di dominio a un indirizzo IP che cambia dinamicamente. Questo permette ai cybercriminali di avere un nome di dominio che punta sempre allo stesso host, però con un indirizzo IP che cambiando dinamicamente li rende irrintracciabili. Inoltre esso è anche molto facile da ottenere in quanto gli attaccanti usando identità false e email fasulle riescono comunque a ottenere il servizio.
- **Fast Fluxing:** In questo caso il nome di dominio associato al C&C server viene costantemente associato a un indirizzo IP, che però ad ogni associazione cambia. Da cui un singolo nome di dominio si ritrova associati indirizzi IP multipli e quindi diventa difficile per le autorità competenti andare a individuare e chiudere il C&C server, poiché esso potrebbe trovarsi in uno qualsiasi di questi IP.

Tutti i meccanismi di protezione sopra citati, nonostante offrano un buon grado di protezione, hanno comunque delle debolezze intrinseche che possono permettere a chi di dovere di andare a bloccare il C&C server e rendere la botnet inutilizzabile. Infatti spesso il dominio per contattare il C&C server della botnet risulta essere o inserito in maniera hard-coded all'interno del codice del malware, oppure risulta essere inserito in dei file di configurazione che scaricati nei bot insieme al malware. Questo in realtà era un grosso problema per gli attaccanti, poiché gli sviluppatori di software antivirus, attraverso tecniche di reverse engineering, avrebbero potuto comunque rintracciare il nome di dominio all'interno del codice del malware, per poi oscurare il C&C server e rendere la botnet inutilizzabile.

Per prevenire questo inconveniente, l'idea vincente è stata quella di fare in modo che fossero i bot stessi a generare i nomi di dominio con cui contattare il C&C

server. Cioè i bot periodicamente generano una lista di nomi di dominio, che si riferiscono a un C&C server, e cercano di contattarlo collegandosi a uno di questi. Tra questi chiaramente solo uno sarà il nome che, una volta risolto, permetterà di collegarsi al C&C server. Questa capacità dei bot di generare da soli delle liste di nomi di dominio periodicamente nel tempo è detta **Domain Fluxing**, e gli algoritmi con cui i bot riescono a generare questi domini sono detti **Domain Generation Algorithm(DGA)**.

Scendendo nel dettaglio, abbiamo che nella fase di infezione all'interno delle macchine compromesse, verranno inseriti degli algoritmi di generazione dei domini, che genereranno una lista di domini in maniera pseudo-casuale. Tali domini saranno validi per un certo intervallo di tempo, e chiaramente nel medesimo intervallo il botmaster, che sa in anticipo come verranno generati i nomi, avrà cura di usarne uno di quelli generati in quel lasso di tempo per il suo C&C server. Scaduto questo intervallo di tempo avviene la rigenerazione dei domini sia da parte dei bot che del botmaster e tutti i domini precedenti risultano non più utilizzati. Chiaramente tutto questo crea dei vantaggi enormi all'attaccante perché:

- La pratica del blacklisting diventa inutile contro gli algoritmi di generazione dei domini in quanto, anche se si riuscissero ad oscurare tutti i nomi di dominio generati in un certo intervallo di tempo, comunque ne verrebbero generati di nuovi, che non essendo in una blacklist, risultano risolvibili.
- L'attaccante può registrare in anticipo il dominio riferito al suo C&C server perché sa in anticipo quando verrà generato.
- I domini sono usati per un breve lasso di tempo in maniera tale che sia difficile registrarli e cercare di classificarli secondo sistemi di reputazione dei domini[8]

Chiaramente i DGA hanno anche degli svantaggi, e uno dei più importanti è sicuramente il fatto che la generazione dei domini non è assolutamente casuale. Come ogni processo che in informatica genera un output casuale, anche la generazione dei domini dipenderà da un "seme" e cioè un qualcosa da cui parte la generazione casuale. Chiaramente facendo reverse engineering dell'algoritmo di generazione dei domini

all'interno di un bot, è possibile individuare il seme da cui parte la generazione casuale. In quel modo chiunque abbia scoperto tale seme sarà in grado di prevenire in anticipo tutti i nomi di dominio che verranno generati dall'algoritmo, compiendo così delle azioni preventive per oscurare in anticipo i domini. Inoltre le caratteristiche del seme ci permettono di individuare quattro classi di algoritmi DGA:

- **TID-DGA:** In questa classe di algoritmi il seme è statico e indipendente dal tempo. Ogni volta viene generata la stessa lista di domini
- **TDD-DGA:** In questo caso l'algoritmo di generazione dei domini è deterministico, cioè so in anticipo quali nomi saranno generati, però il seme di generazione varia col tempo
- **TIN-DGA:** In questo caso l'algoritmo di generazione dei domini predice una lista di domini non prevedibile a priori con un seed statico
- **TDN-DGA:** L'algoritmo genera una lista di domini non prevedibile a priori e inoltre il seme di generazione cambia nel tempo. Non è assolutamente fattibile cercare di predire la lista dei domini

Infine un altro svantaggio dei malware DGA risiede nel fatto che molte delle richieste che faranno i bot per connettersi con il C&C server, saranno delle query che daranno come risultato NXDOMAIN, e cioè delle query a domini non esistenti. Chiaramente mediante un'analisi del traffico DNS è possibile monitorare i tassi di NXDOMAIN generati dalle richieste fatte da una macchina. Se il tasso è alto allora probabilmente quella macchina potrebbe far parte di una botnet, che utilizza malware basati su DGA e quindi andrebbe monitorata. Nonostante tutti gli svantaggi però, con le odierne tecniche di blacklisting risulta molto difficile andare a bloccare la proliferazione di malware basati su DGA. Per questo si è dovuti ricorrere a nuovi strumenti compresa l'intelligenza artificiale.

1.3 Machine Learning

1.3.1 Reti Neurali

Una rete neurale artificiale (abbreviata in ANN o NN dall'inglese Artificial Neural Network) è una schematizzazione formale del funzionamento del cervello umano, o del cervello biologico più in generale [9][10]

Il cervello è formato da un grandissimo numero di neuroni, collegati tra loro, e da altrettante sinapsi. Nel momento in cui il nostro cervello deve elaborare un'informazione proveniente dal mondo esterno, i neuroni si attivano ricevendo lo stimolo esterno in forma di segnale elettrochimico, che poi fa attivare anche i neuroni vicini ai neuroni attivati per primi. In breve tempo il segnale elettrochimico si diffonde al resto del cervello, e nel processo di elaborazione vengono interessate intere aree del cervello che in "output" producono o azioni complesse o processano le informazioni ricevute.

Pensando al funzionamento del cervello, i primi studiosi che hanno lavorato nel settore dell'intelligenza artificiale avevano capito che la differenza principale tra un elaboratore e il cervello umano era la seguente: mentre un elaboratore esegue istruzioni in maniera sequenziale usando la CPU, andando a manipolare dei dati, o input, caricati in memoria centrale, invece nel cervello umano l'informazione viene processata in maniera parallela da tutti i neuroni, che allo stesso tempo svolgono la funzione di contenitori di informazioni. A differenza dei neuroni però, in cui l'elaborazione dell'informazione avviene con frequenza pari a 100 MHz, invece in un elaboratore tale frequenza arriva all'ordine del GHz. Questa cosa suggerì l'idea che si potevano ottenere ottimi risultati replicando la stessa struttura del cervello umano negli elaboratori.

Nasce quindi l'idea di rete neurale come schematizzazione matematica e formale del cervello umano. Una rete neurale, come un cervello umano, è composta da neuroni artificiali, cioè piccole unità di elaborazione, che come il neurone umano si attivano quando ricevono una quantità di segnale elettrico superiore alla sua soglia di attivazione. Tali neuroni sono collegati poi, da delle sinapsi che permettono di scambiarsi i

segnali. Al livello matematico perciò, il neurone artificiale è principalmente composto da due entità:

- **Pesi:** i pesi sono dei vettori di coefficienti che moltiplicano il segnale elettrico, o meglio la rappresentazione matematica di quest'ultimo che arriva in input. Tali grandezze hanno un ruolo chiave come vedremo in seguito nell'apprendimento perché essi vengono modificati, al fine di migliorare le prestazioni della rete
- **Funzione di attivazione:** la funzione di attivazione del neurone è la funzione che in input prende la sommatoria dei dati pesati con i pesi descritti in precedenza e produce un output che verrà poi inviato ad altri neuroni come input

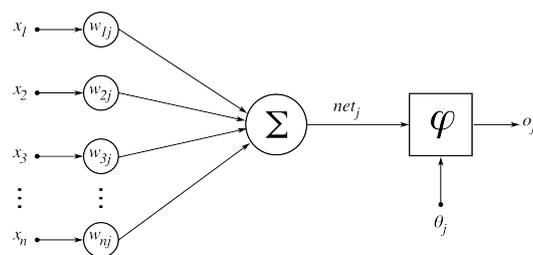


Figura 1.1: Struttura matematica di un neurone artificiale[1]

L'insieme di tutti i neuroni forma la rete neurale che è suddivisa in più strati:

- **Strato di Embedding:** essendo dei modelli matematici, le reti neurali in input si aspetteranno delle grandezze numeriche o matematiche che sono dei vettori. Per questo motivo per fare in modo che le reti neurali possano essere applicate a più ambiti(classificazione di nomi, interpretazione di immagini e video) è necessario, qualunque sia l'input, trasformare in vettori, secondo un certo criterio, i dati in ingresso, in maniera tale che siano processabili dai neuroni
- **Strato nascosto:** è lo strato interno della rete neurale cioè quello in cui avviene il processo di apprendimento. In questa parte i dati passano attraverso i neuroni e le varie sinapsi, venendo modificati alterati e contribuendo alla modifica dei pesi della rete

- **Strato di output:** è lo strato più esterno della rete ed è quello che fornisce i risultati finali del processo di apprendimento fatto dalla rete

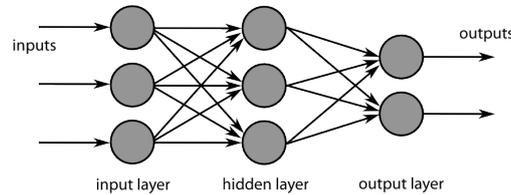


Figura 1.2: Struttura rete neurale artificiale[2]

Tale struttura è generalmente comune a quasi tutti i tipi di rete neurale, che possiamo sfruttare al giorno d'oggi. La particolarità principale di queste reti, che venne introdotta col perceptrone di Rosenblatt cioè il primo sistema cibernetico in grado di riconoscere le immagini, è la **retropropagazione dell'errore**. Nella retropropagazione dell'errore la rete neurale riceve degli input conoscendo però quali sono gli output desiderati per questi input. A questo punto la rete processa gli input e misura gli output che ottiene confrontandoli con gli output attesi per i vari input, calcolando l'errore. Quest'ultimo viene poi propagato all'indietro e usato per aggiustare i pesi della rete, in maniera tale che poi vengano ottenuti risultati migliori. Tale processo avviene per ogni input che diamo alla rete e viene definito *epoca di apprendimento*. Chiaramente più epoche di apprendimento vengono fatte e più la rete "impara" meglio a processare in futuro gli input. Con questo tipo di apprendimento la rete riesce poi a riconoscere e processare input che non aveva mai visto durante la fase di addestramento.

Esistono vari tipi di rete neurale:

- Reti neurali feed-forward: è il tipo di rete neurale più semplice. In tale rete le connessioni tra le unità non formano cicli. Questo tipo di rete neurale fu la prima e più semplice tra quelle messe a punto. In questa rete neurale le informazioni si muovono solo in una direzione, avanti, rispetto a nodi d'ingresso, attraverso nodi nascosti (se esistenti) fino ai nodi d'uscita. Nella rete non ci sono cicli. Le reti feed-forward non hanno memoria di input avvenuti a tempi precedenti, per cui l'output è determinato solamente dall'attuale input.[11]

- Reti neurali Ricorrenti(RNN): In questo tipo di reti neurali le connessioni tra le unità(i neuroni della rete) possono andare anche a formare dei cicli. All'interno dello strato nascosto della rete può accadere, che l'uscita di un sotto-strato più vicino allo strato di output può essere usato anche come ingresso di un altro sotto-strato, che si trova più indietro e vicino allo strato di input. Da cui in questa maniera l'output di un neurone non dipenderà solo dall'ingresso che gli arriva ma anche dall'output del neurone stesso, nel più semplice dei casi, all'istante precedente. Tale output a sua volta dipende dall'input all'istante precedente e dall'output ottenuto due istanti prima

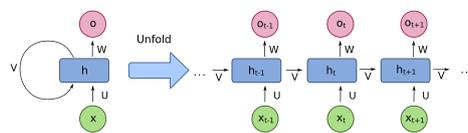


Figura 1.3: Struttura rete neurale ricorrente[3]

- Reti neurali Convolutionali: sono reti di tipo feed-forward, più avanzate usate soprattutto nel campo del riconoscimento delle immagini. A differenza delle altre reti nelle reti convoluzionali non ci sono cicli ma i sotto-strati contenuti all'interno degli strati nascosti sono ripetuti più volte. Il cuore della rete è lo strato di convoluzione e cioè quello che si occupa di identificare le forme, le curve e gli angoli presenti nell'immagine analizzata.

1.3.2 LSTM

Le reti che meglio si prestano al nostro lavoro sono un particolare tipo di reti neurali ricorrenti, che superano in prestazioni le normali reti ricorrenti. Infatti nelle reti neurali ricorrenti, i neuroni possono avere alcuni tipi di funzioni di attivazione come le sigmoidi, la tangente iperbolica oppure delle funzioni lineari rettificanti. Durante il processo di retropropagazione dell'errore e di aggiustamento dei pesi, si va ad usare il gradiente di queste funzioni di attivazione. Si possono originare due tipi di situazioni:

- Se vengono utilizzate funzioni del tipo sigmoideale o tangente iperbolica, i valori dei loro gradienti restano nell'intervallo $[0,1]$. Visto che l'algoritmo di retropropagazione prevede che i gradienti vengano moltiplicati in catena lungo gli strati, è chiaro che il prodotto di un elevato numero di valori compresi tra 0 ed 1 porta il valore complessivo a diminuire velocemente lungo la catena di neuroni e i pesi vengono aggiustati sempre peggio. Tale tipo di problema è detto *vanishing gradient* e porta ad avere dei pesi che sono oscillanti[10].
- Se al contrario vengono utilizzate funzioni lineari, come la ReLU o la lineare pura, i valori dei gradienti possono essere anche superiori ad 1 e quindi il valore complessivo può crescere enormemente lungo la catena di neuroni. Tale tipo di problema è detto *exploding gradient* e porta la rete a fare lunghi calcoli senza produrre un output[10].

Per risolvere questo problema Sepp Hochreiter e Jürgen Schmidhuber nel 1997 proposero il primo modello di rete neurale LSTM[10]. LSTM sta per Long Short-Term Memory e rispetto alle reti neurali ricorrenti, la struttura interna del neurone della rete è diversa in quanto possiede numerose porte, che gli consentono di decidere da solo, quali informazioni devono essere memorizzate all'interno della cella di memoria del neurone, e quali invece sono da dimenticare. Tale scelta viene fatta grazie a una porta detta *forget gate*, che decide se l'informazione in ingresso deve essere mantenuta, o buttata, sulla base degli ingressi, della retropropagazione dell'errore e dello stato precedente della cella

Le reti neurali LSTM con questa capacità di immagazzinare le informazioni o scartarle in maniera automatica prevengono i problemi di *vanishing gradient* e *exploding gradient*. Nel caso in cui a una certa informazione corrisponda a un gradiente, nella retropropagazione dell'errore, che tende a 0 allora questa viene dimenticata grazie alla *forget gate*. Al contrario se tale informazione corrisponde a un gradiente della funzione di attivazione che tende a diventare maggiore di 1, essa viene memorizzata e mantenuta uguale. In questa maniera tali reti riescono a effettuare degli addestramenti con dataset molto più grandi, e producono risultati

più consistenti.

Infine in generale nell'ambito delle reti neurali e dell'intelligenza artificiale, per valutare dei modelli predittivi si usano diversi parametri che useremo anche in seguito negli esperimenti. In generale per le predizioni di una rete neurale, nel nostro caso un classificatore, ci possono essere 4 possibili risultati:

- TP(true positive), la rete ha predetto positivo per un valore che è effettivamente positivo
- FP(false positive), la rete ha predetto positivo per un valore che è negativo
- TN(true negative), la rete ha predetto negativo per un valore che è effettivamente negativo
- FN(false negative), la rete ha predetto negativo per un valore che è positivo

Le varie metriche che si usano per valutare le performance di una rete sono quindi introdotti questi parametri:

$$\mathbf{Precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}} \quad (1.1)$$

Tale valore rappresenta la precisione della rete, cioè quanti positivi sono stati predetti correttamente su tutti i positivi individuati come tali dalla rete, in riferimento a una determinata classe di oggetti.

$$\mathbf{Recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} \quad (1.2)$$

La recall indica il rapporto di istanze positive correttamente individuate dal sistema

$$\mathbf{F1} = 2 * \frac{\mathbf{Precision} * \mathbf{Recall}}{\mathbf{Precision} + \mathbf{Recall}} \quad (1.3)$$

L'F1-score è la media armonica di precision e recall. Si usa tale tipo di media perchè a differenza di una media normale, quella armonica attribuisce un peso maggiore ai valori piccoli. Questo fa sì che un classificatore ottenga un alto punteggio F1, solo quando precisione e recupero sono entrambi alti.

Ora tutte queste metriche si usano per descrivere le prestazioni della rete relativamente a delle classi, che la rete deve individuare e saper riconoscere. Per valutare invece la performance della rete andando a considerare tutte le classi si usano delle medie, delle metriche presentate precedentemente, che si chiamano *Micro Average* e *Macro Average*. Per calcolare *Micro Average* si vanno a prendere tutti i contributi delle varie classi, li si somma insieme e poi si fa la media. Ad esempio per la precision quando vado a fare la micro average al numeratore avrò la somma di tutti i true positive delle varie classi e al denominatore la somma di tutti i true positive delle varie classi, più la somma di tutti i false positive delle varie classi. La *Macro Average* invece per una determinata metrica riferita alle classi, consiste nel fare la media tra i valori ottenuti per le metriche delle classi. Ad esempio per la precision se ho quattro classi, la macro average sarà la media aritmetica tra i quattro valori di precision ottenuti

Altra metrica importante è l'*Accuracy* di un modello, che sarebbe l'accuratezza del modello e viene calcolata dividendo la somma di tutti i veri positivi per le classi, per tutti i valori calcolati.

Infine, anche se non è una metrica, un altro strumento importante per l'analisi delle prestazioni di una rete è la *matrice di confusione*. Questa matrice riporta sulle righe gli elementi delle classi reali e sulle colonne le predizioni di questi elementi. In questa maniera guardando la matrice di confusione riusciamo a vedere se la rete impara meglio a riconoscere certi tipi di classi rispetto ad altre. Grazie ad essa inoltre possiamo capire se esistono classi che tendono ad inglobare elementi appartenenti ad altre o esistono classi che vengono confuse con altre classi.

1.3.3 Machine Learning per i DGA

Come accennato già prima, disattivare l'attività delle botnet, che usano per proteggersi i DGA, risulta molto difficile usando solo pratiche di blacklisting. Infatti l'ideale per spegnere una botnet sarebbe quello di analizzare una grande quantità di traffico DNS, e tale cosa non è possibile in maniera algoritmica e sequenziale. A questo proposito risulta utile sfruttare il machine learning.

Il machine learning è una variante alla programmazione tradizionale nella quale in una macchina si predispongono l'abilità di apprendere qualcosa dai dati in maniera autonoma, senza istruzioni esplicite[12]. Da cui nel machine learning la macchina basandosi sull'esperienza, e sulla forma dei dati riesce da sola, senza che noi la istruiamo con un algoritmo, a capire cosa fare, e a correggersi nel caso in cui si sbaglia. Applicheremo il machine learning al processo di classificazione dei nomi di dominio in benevoli o malevoli.

Le tecniche di machine learning, ormai sono da tempo usate nell'ambito della classificazione dei nomi di dominio, poiché ci si è accorti che tali algoritmi funzionano molto bene e permettono di ottenere prestazioni robuste anche su larghe quantità di dati. Specialmente c'è un interesse crescente nei modelli di machine learning che riescono a rilevare nomi di dominio, generati da un algoritmo DGA, in tempo reale, in maniera tale da prevenire qualsiasi tipo di comunicazione con il C&C server[13]. Oltre a queste tecniche, sono da annoverare anche le tecniche che rilevano i nomi di dominio DGA, non in real time ma basandosi su vecchi dati, in maniera tale da costruire una specie di storico delle attività sospette che avvengono in una rete[14]

Esistono due macro approcci che possono essere utilizzati per l'addestramento di modelli di machine learning:

- **Approccio Supervisionato:** in questo caso l'algoritmo di machine learning elabora delle previsioni basandosi su una serie di esempi già etichettati. Cioè basandosi su degli esempi che gli forniamo in input, il modello, nel processo di addestramento, cerca di rilevare partendo da questi esempi una funzione che li collega. Sulla base di questa funzione poi farà in futuro delle previsioni su dati nuovi e non presenti nel set di training
- **Approccio Non Supervisionato:** il set di training, e cioè l'insieme dei dati di esempio che si fornisce alla rete in fase di apprendimento non è etichettato, quindi la rete basandosi direttamente sulle caratteristiche degli input cerca di trovare tra questi delle relazioni. In seguito userà le relazioni rilevate per fare delle previsioni su dati mai visti prima.

Nel lavoro qui presentato avremo modo di testare sia approcci ad apprendimento supervisionato, cioè le reti LSTM, sia approcci ad apprendimento non supervisionato come FastText.

Un'ulteriore distinzione tra i vari modelli di machine learning può essere fatta anche sulla base delle caratteristiche o features, che questi modelli prendono in considerazione durante l'apprendimento. Nel caso della rilevazione dei nomi di dominio DGA esistono tre macro approcci per la classificazione:

- **Approccio Context-Aware:** in questo caso le caratteristiche prese in considerazione durante l'apprendimento dai dati, dipendono dall'esecuzione dello specifico DGA che si realizza in un preciso ambiente con una specifica configurazione e in un particolare lasso di tempo. Un esempio di queste feature sono quelle estratte al momento dell'ispezione delle risposte DNS[15].
- **Approccio Context-Free:** le features prese in considerazione durante l'apprendimento dai dati, dipendono solamente dalle caratteristiche e dall'analisi lessicale del nome di dominio preso in esame. Non incidono le configurazioni di ambiente, la tempistica e altri fattori menzionati prima.
- **Approccio Featureless:** il modello di machine learning non prende in considerazione nessuna caratteristica durante l'apprendimento dai dati

Tra queste tre tipologie di approccio nel caso del rilevamento di malware basati su DGA è molto utile prendere in considerazione gli ultimi due approcci descritti, per i quali esistono modelli che riescono con una buona accuratezza a prevedere se un nome di dominio è benevolo o malevolo.

Come riportato nello studio di Sivaguru et al.[13], nell'approccio basato su features context-free, si estraggono tipicamente delle caratteristiche lessicali e linguistiche dai nomi di dominio che si vanno ad analizzare. Principalmente le features scelte sono le seguenti:

- Lunghezza del Top-Level Domain
- Lunghezza del Second-Level Domain

- Numero Consonanti del SLD diviso la lunghezza del SLD
- Numero Cifre del SLD diviso la lunghezza del SLD
- Media Circolare 2-Gram
- Media Circolare 3-Gram
- Numero caratteri nel Second-Level Domain
- Numero di Token nel Second-Level Domain
- Numero di cifre nel Second-Level Domain
- Lunghezza della sequenza di consonanti nel Second-Level Domain
- Numero di caratteri unici nel dominio

Queste features estratte dai nomi di dominio vengono poi utilizzate nel processo di apprendimento supervisionato delle **Random Forests(RF)**. La scelta delle RF per questo tipo di task, risiede nel fatto che sono applicabili in molte casistiche, e le loro performance sono ottime. In più oltre ad essere modelli di grande accuratezza, le RF sono algoritmi di addestramento che scalano bene su dataset di grandi dimensioni, e questo spiega sicuramente la loro popolarità per il rilevamento dei DGA[13].

Gli approcci basati su feature sono vantaggiosi in quanto permettono di avere un processo di analisi controllato e trasparente. Nel campo dei DGA però non è facile individuare a priori quali caratteristiche l'algoritmo di generazione dei domini prende in considerazione. Per questo nel campo del rilevamento automatico di DGA, si sono presi in considerazione anche approcci featurless che sfruttano il deep learning, e in particolare le reti neurali. Nel caso del rilevamento di DGA si è scelto di utilizzare le reti neurali LSTM, per le caratteristiche che esse hanno e che abbiamo descritto in precedenza.

Principalmente nella rilevazione di malware basati su DGA, sono interessanti due tipi di classificazione, ovvero la classificazione binaria e la classificazione multiclasse. Nella classificazione binaria lo scopo è quello di dividere i nomi di dominio benevoli

dai nomi di dominio malevoli. Nella classificazione multiclasse lo scopo invece è quello di capire la famiglia di DGA alla quale appartiene il nome di dominio analizzato. Chiaramente la classificazione multiclasse risulta essere un problema molto più complesso, perchè esistono molte famiglie di DGA, e molte di queste generano più domini malevoli rispetto ad altre nello stesso arco temporale. Ciò rende la classificazione multiclasse un processo in cui per ogni nome si possono assegnare tante etichette diverse, una per classe, e con un grande squilibrio in termini di quantità di nomi per ogni classe. Quest'ultimo problema viene detto *multiclass imbalance*[13].

Il problema della multiclass imbalance nella classificazione dei domini DGA, è una problematica fondamentale. Infatti anche se alcuni algoritmi DGA generano in proporzione un numero di nomi minore, rispetto a quello generato da altri, non è detto comunque che questi siano associati a un malware meno pericoloso. Da cui a causa del problema della multiclass imbalance, può accadere che la normale rete LSTM che utilizziamo per la classificazione, durante l'addestramento conosca meno domini di una famiglia, piuttosto che di un'altra. Questo in futuro potrebbe generare previsioni sbagliate da parte della rete, che potrebbe non riuscire in alcun modo a riconoscere le famiglie di DGA meno numerose.

Per questo motivo Duc Tran et al.[14] hanno introdotto un meccanismo di compensazione della multiclass imbalance, creando un algoritmo che durante il processo di retropropagazione dell'errore permette alla rete di tenere conto delle classi meno numerose. Infatti la rete, durante il processo di apprendimento supervisionato, commette errori maggiori sugli elementi del dataset che non aveva mai visto prima. L'errore viene poi utilizzato nella retropropagazione dell'errore per correggere i pesi dei neuroni della rete in maniera tale da commettere un errore minore quando rivedrà un elemento simile. Se però ci sono elementi poco numerosi, anche se la rete durante l'addestramento li vede tutti, in fase di test continuerà comunque a commettere degli errori non indifferenti, che la porteranno a fare previsioni errate. L'algoritmo della multiclass imbalance ideato da Duc Tran et al. risolve questo problema: ad ogni famiglia presente nel dataset viene assegnato un certo costo, un peso che poi andrà ad influire nella retropropagazione dell'errore. Meno la famiglia è numerosa, cioè

il dataset contiene pochi nomi di dominio appartenenti a quella famiglia, e più il peso assegnato sarà grande. In questa maniera quando la rete vedrà un elemento appartenente a una famiglia poco numerosa, se l'errore commesso nel riconoscerlo supererà una certa soglia, allora questo verrà moltiplicato per il peso assegnato alla famiglia. Ciò porterà, durante la retropropagazione dell'errore ad avere un adeguamento più alto per le famiglie meno numerose. In questa maniera la rete riuscirà a riconoscere meglio anche elementi di famiglie meno numerose nel dataset.

Proprio per questo motivo invece di una normale rete LSTM, useremo per i test una rete LSTM.MI pre addestrata, che è capace di riconoscere in maniera migliore nomi di dominio di famiglie non molto numerose nel dataset di training.

1.4 FastText

Come già accennato prima le reti neurali per funzionare correttamente necessitano di uno strato di embedding, che converta gli input in vettori che la rete può processare. L'embedding infatti consiste in una mappatura dell'input, cioè consiste nel rappresentare variabili discrete, nel nostro caso i nomi di dominio, come vettori continui che possono essere collocati in uno spazio vettoriale.

Anche la rete LSTM.MI che utilizzeremo per questo lavoro necessita di uno strato di embedding. Nel lavoro originale di Duc Tran et al. lo strato di embedding della rete non fa altro che, ricercare tutti i caratteri nel dataset di training, associarli in maniera casuale ad un numero, e poi trasformare tutti i nomi di dominio presenti nel dataset in vettori numerici in cui ogni componente rappresenta un carattere del nome, che però viene sostituito con il numero associatogli prima. Tale tecnica di embedding è detta *embedding randomico*.

In realtà esistono però anche altri modi per realizzare questo strato di embedding, che sono più complicati e potrebbero avere prestazioni migliori. Tra questi scegliamo di utilizzare *FastText*. *FastText* è un'estensione di un'architettura (Word2Vec) rilasciata da Facebook AI Research nel 2016. È una libreria open-source utilizzata per compiere varie operazioni quali la rappresentazione delle parole come vettori,

e la classificazione del testo. FastText si propone come obiettivo, dato un insieme di parole all'interno di frasi, di apprendere da solo delle rappresentazioni vettoriali continue per queste parole.

Rispetto ad altre tecniche di machine learning per l'apprendimento della rappresentazione continua delle parole, FastText, invece di prendere in considerazione la parola intera, e le parole ad essa vicine nel dataset, per creare il vettore ad essa associata, prende in considerazione la rappresentazione interna della parola dividendola in n-grammi[16]. Infatti presa una parola, dopo averla divisa in n-grammi, trova prima una rappresentazione vettoriale degli n-grammi che compongono la parola stessa. Poi, ricava il vettore associato alla parola come somma di tutti i vettori associati agli n-grammi che la compongono. In questa maniera non solo si ottengono rappresentazioni migliori delle parole, viste nel training, ma si riescono anche a costruire rappresentazioni di parole mai viste nel dataset, come somma di n-grammi.

Inoltre FastText a differenza di altre tecniche di rappresentazione delle parole come vettori continui, garantisce, proprio perché le parole sono viste come somma di n-grammi, prestazioni migliori anche con dataset ridotti, rispetto ad altre tecniche. Tutti questi vantaggi che ci da FastText sono il motivo per cui in questo lavoro abbiamo scelto di usare questa tecnologia come strato di embedding della rete. Il risultato dell'addestramento di FastText è infatti un file contenente tutte le rappresentazioni dei caratteri, bigrammi o trigrammi, trovati nel dataset di training come vettori a 128 componenti. Tali file verranno poi usati in fase di embedding per tradurre i nomi di dominio in rappresentazioni vettoriali continue.

Capitolo 2

Materiali e Metodi

2.1 Risorse

2.1.1 Log DNS rete GARR

Una delle risorse principali di cui abbiamo usufruito durante questo lavoro sia per la creazione dei dataset per l'addestramento di FastText, sia per la creazione di un dataset per la rete LSTM.MI, è il log del DNS server della rete GARR.

La rete GARR venne istituita nel 1991 dal Consorzio per l'Armonizzazione della Rete della Ricerca. L'obiettivo di questa rete è quello di connettere tutte le principali università, scuole e istituti di ricerca, con una rete a banda ultra larga, che faciliti le attività di ricerca e di studio. Chiaramente anche la nostra università, la politecnica delle Marche è connessa alla rete GARR.

Data la sua grandezza, la rete GARR dispone di un suo DNS server interno, che ovviamente si occuperà di risolvere solo le chiamate a nomi di dominio effettuate da macchine connesse alla rete stessa. Il DNS server, ogni volta che avviene una chiamata a un certo nome di dominio, registra tale chiamata in un file di log, cioè un file che permette all'essere umano di vedere gli eventi registrati dalla macchina, in un certo lasso di tempo. Accedendo ai file di log ci è possibile estrarre tutte le informazioni relative ad ogni singola richiesta di risoluzione di un nome di dominio, compreso il nome di dominio stesso. Tali file sono lo strumento che abbiamo utilizzato per costruire parte dei nostri dataset di training.

Capitolo 2 Materiali e Metodi

Il Dipartimento di ingegneria dell'informazione, infatti, per questo lavoro, ha concesso l'accesso a una macchina, collegata alla rete GARR, che salva tutti i log generati dal DNS server della rete stessa. Ogni ora il DNS server registra in un file di testo *.log*, che viene compresso per risparmiare spazio, tutte le chiamate a nomi di dominio, effettuate in quell'ora, da macchine connesse alla rete. Tale operazione viene ripetuta per ogni ora del giorno per tutti i giorni. Questo insieme di file, che possono essere scaricati in una macchina locale, contengono, oltre ai nomi di dominio stessi, anche altre informazioni che possono essere utili nel caso in cui si effettui un'analisi passiva sul traffico DNS.

Quello che è interessante per il nostro lavoro sono i nomi di dominio, che useremo sia per addestrare FastText, sia per testare la rete LSTM.MI. L'uso di questi file è vantaggioso, perché ci consente di avere accesso a tantissimi nomi di dominio, anche selezionando pochi file di log. Infatti, prendendo come esempio il file scaricato nel periodo che va da mezzanotte all'una di notte, del 01 agosto 2021, notiamo che sono state salvate circa 200000 chiamate a nomi di dominio, un numero che poi cresce nelle ore centrali del giorno. Già scaricando solo un giorno di log GARR si riescono a visionare circa 7 milioni di chiamate a nomi di dominio, compresi i duplicati, che sono certamente un numero sufficiente per addestrare lo strato di embedding basato su FastText.

Un'ultima osservazione da fare sulla rete GARR è che essa a differenza di una rete generica, come potrebbe essere il DNS server di Google, è soggetta a fluttuazioni sul numero di chiamate a nomi di dominio nelle diverse ore del giorno. Infatti poiché essa connette, istituti di istruzione e ricerca italiani, è chiaro che durante la notte ci sarà un minor numero di chiamate a nomi di dominio rispetto al giorno, come è chiaro che durante i giorni feriali saranno presenti più chiamate rispetto ai giorni festivi. Inoltre tale rete, viene usata specialmente da studenti e professori universitari, quindi da persone con un grado di conoscenza della rete, e delle minacce che ci sono online, superiore a quelle di un generico utente domestico. Ciò comporta, come vedremo anche dopo nella costruzione del dataset relativo alla rete, che sarà più difficile trovare tante chiamate a nomi di dominio appartenenti a famiglie di DGA,

perché statisticamente sono minori i nodi della rete infettati.

2.1.2 Bambenek Feed

Negli ultimi anni, data la loro efficienza, si è assistito a una crescita dell'uso, da parte degli attaccanti, degli algoritmi di generazione dei domini. A tale aumento però è corrisposta anche una crescita degli enti di ricerca che hanno incentrato il loro lavoro sui DGA. Tra queste particolarmente degna di nota è il *Bambenek Consulting*, che è un'organizzazione che si occupa di contrastare le minacce generate dai criminali online.

Il punto fondamentale, che rende il Bambenek Consulting un'organizzazione molto importante, è che essa produce ogni giorno un *feed* in cui sono contenuti tutti i nomi di dominio che possono essere usati, dai cyber-criminali che utilizzano i domain generation algorithms. Questi feed, principalmente, sono utilizzati, anche nei lavori di ricerca citati, come blacklist, che contengono nomi di dominio generati da diverse famiglie di DGA.

Il feed è un file *.csv* compresso contenente un'intestazione che riporta quando il feed è stato generato e i giorni presi in considerazione per la validità dei domini. A seguito dell'intestazione è presente una lista di domini generati da diverse famiglie di DGA che però fanno riferimento, non solo al giorno in cui il feed è stato emesso, ma anche a giorni precedenti e successivi. Infatti, come riportato nel capitolo 1, i domain generation algorithm generano una lista di domini, che possono essere usati per contattare il C&C server, che è valida solo in un certo intervallo di tempo. Ad esempio una famiglia di DGA può generare una lista di domini che sarà valida per un intervallo di tempo pari a 5 giorni. Da cui se un nome di dominio verrà generato il giorno 26 agosto, esso verrà incluso nel feed, generato da bambenek, già dal 24 agosto e rimarrà incluso fino al giorno 28. Dato questo comportamento si dice che il feed giornaliero è "centrato" nel giorno in cui è emesso.

Il feed viene curato dai ricercatori del Bambenek Consulting, che si occupano anche di quantificare le finestre temporali, per cui i nomi di dominio appartenenti a una famiglia di DGA rimangono validi. Il discorso diventa più facile, nel caso in cui

la generazione dei domini non dipende dal tempo: in quel caso i nomi hanno validità più estesa e sono rimossi o aggiunti al feed a seconda delle circostanze.

In questa maniera, grazie al feed del Bambenek Consulting, si riesce ad ottenere una lista di domini certamente malevoli, generati in un certo intervallo di tempo, che possiamo confrontare con quelli contenuti nei log del DNS server della rete GARR. Utilizzando l'informazione che i nomi di dominio nel bambenek feed, relativi a un certo giorno, sono certamente malevoli, sapremo per certo che, se ritroviamo gli stessi nomi all'interno del log del DNS server della rete GARR, allora questi potranno essere etichettati come malevoli all'interno del dataset.

In ogni feed sono presenti varie informazioni sui nomi di dominio, oltre ai nomi di dominio stessi, come la famiglia a cui questi appartengono, la loro finestra di validità, e il link al manuale redatto dal Bambenek Consulting, per la risoluzione dei problemi che può creare quella famiglia. Chiaramente sono di nostro interesse solo i primi due dati contenuti nel feed, essenziali per il processo di classificazione sia binaria che multiclasse.

Per i motivi descritti sopra i feed del Bambenek Consulting risultano essere molto utili e affidabili, anche perché oltre al normale feed giornaliero riguardante i DGA, l'organizzazione fornisce anche un altro feed denominato *High-Confidence-DGA-feed*. Tale feed contiene solo nomi di dominio relativi a famiglie di DGA, che durante il processo di classificazione generano un coefficiente di falsi positivi molto basso.

2.1.3 Majestic Million

Grazie alla risorsa descritta nel paragrafo precedente, abbiamo un modo per capire quali nomi di dominio, contenuti nel log del server DNS della rete GARR, sono certamente malevoli. Però, è chiaro che ci serve uno strumento analogo, anche per capire quali nomi di dominio sono certamente benevoli.

Nelle ricerche citate, per questo scopo si usava un servizio della compagnia Amazon, che si chiama *AWS Alexa*. AWS Alexa è un sistema di reputazione e analisi del traffico web, che offre un servizio in particolare, denominato *top-sites*, che permette, a pagamento, di avere accesso alla lista dei nomi di dominio certamente benevoli,

che hanno avuto più chiamate nel periodo di tempo considerato. Data la natura a pagamento di questo servizio, nonostante sia estremamente affidabile, si è deciso di optare per un'altra opzione offerta dalla compagnia inglese Majestic.

Il servizio di reputazione, che abbiamo utilizzato per ottenere una whitelist di nomi di dominio benevoli, è il Majestic Million. Tale servizio fornisce l'elenco del primo milione di nomi di dominio più importanti nel web, e con più sotto-reti di provenienza. In questo caso la classifica, che è disponibile sul sito di Majestic Million, è stilata secondo delle statistiche e criteri che vengono stabiliti dalla compagnia Majestic. In realtà, come è evidente dalla descrizione che ne abbiamo dato, e che viene data anche dalla stessa compagnia, il Majestic Million non è un vero e proprio sistema di reputazione dei nomi di dominio.

Quello che però ci spinge a scegliere la lista prodotta da questo servizio, come lista di nomi di dominio certamente benevoli, è che i nomi di dominio presenti nella classifica fanno riferimento a siti molto visitati nel web in tutto il mondo. Oltretutto i tool di analisi utilizzati dalla compagnia Majestic, per stilare questa classifica, sono molto avanzati. Queste sono garanzie sufficienti affinché la lista redatta dalla Majestic possa essere usata come whitelist, in sostituzione del servizio top-sites di Amazon.

Come il feed del Bambenek Consulting, anche il Majestic Million è costantemente aggiornato dalla compagnia omonima. L'unica differenza tra il Bambenek Feed e il Majestic Million, è che, mentre il feed dei domini malevoli è prodotto ogni 24 ore, invece la classifica del milione di siti più visitati viene aggiornata solo il giovedì e la domenica di ogni settimana. Per questo motivo, mentre per il feed Bambenek è necessario un download giornaliero della risorsa, invece nel caso di Majestic Million, il download può essere effettuato solo nei giorni in cui c'è un aggiornamento della lista. Inoltre, la classifica rimane abbastanza stabile: analizzando due file successivi nel tempo, scaricati dal Maejstic Million, si può verificare che la variazione dei nomi di dominio presenti nella classifica è irrilevante ai fini del nostro esperimento.

Da sottolineare, è anche il fatto che il Majestic Million, come top-sites di Alexa, è rivolto a un pubblico internazionale e contiene nomi di dominio, riferiti a siti di

tutto il globo. Dunque se la variazione dei nomi di dominio nella classifica al livello globale nel corso del tempo è piccola, possiamo supporre che, anche la variazione dei soli nomi di dominio nazionali italiani, sia piccola. Da cui nella costruzione del dataset non sono necessari tanti file di Majestic Million.

In conclusione, con Majestic Million abbiamo uno strumento certamente affidabile, con cui possiamo capire quali nomi di dominio, contenuti nei log del DNS server della rete GARR, sono certamente benevoli. Il file, che contiene questi nomi di dominio benevoli, scaricabile dal sito di Majestic Million, è in formato *.csv* e quindi possiamo facilmente estrapolare da esso, tutte le informazioni che ci servono riguardo i nomi di dominio con il linguaggio Python.

2.2 Datasets

Descritti le risorse di cui ci possiamo avvalere, per costruire i dataset di cui necessitiamo, analizziamo come sono stati costruiti al livello tecnico.

2.2.1 Dataset per il training di FastText & Risultati

Prima di poter testare il classificatore LSTM.MI, è necessario ottenere i files di embedding, che sono il risultato dell'addestramento di FastText, per la rappresentazione delle parole come vettori. Questo ci ha spinto, per prima cosa, a costruire un dataset di addestramento per FastText, utilizzando i dati reali provenienti dal DNS server della rete GARR.

Per iniziare, sono stati selezionati un insieme di file *.log*, scaricati dalla macchina del dipartimento connessa alla rete, da cui estrarre i nomi di dominio. Il periodo di osservazione scelto è stato quello che andava dal 01 agosto 2021 al 26 agosto 2021. A questo punto in maniera automatica, con un script scritto nel linguaggio Python3, si sono processati, giorno per giorno, tutti i file di log raccolti nel periodo indicato.

Il processo di raccolta dei nomi si sviluppa in questa maniera:

- Viene decompresso il file di log relativo alla prima ora del giorno processato, e si inizia la lettura.

- Ogni riga del file di log contiene tutte le informazioni relative a una chiamata al server DNS della rete GARR, avvenuta in quel giorno, e in quell'ora. Per ogni riga dunque lo script mette tutte le informazioni relative a quella chiamata in un oggetto, creato ad hoc per questa situazione
- Una volta inseriti i dati in questo oggetto, il programma seleziona la proprietà dell'oggetto che memorizza il nome di dominio, e lo scrive in una riga di un altro file di testo.
- Conclusa questa operazione, si passa alla riga successiva del file di log e si ripetono i passaggi sopra, fino a che non si arriva all'ultima riga del file.

Questa serie di passaggi viene ripetuta per ogni ora di log in un giorno, e per tutti i giorni nel periodo di osservazione scelto.

Al termine del processo di raccolta, il risultato ottenuto è stato un file di testo in cui erano memorizzati circa 130 milioni di nomi di dominio, relativi a chiamate al server DNS della rete GARR. Questo ha rappresentato un primo corpus grezzo di dati, però non ancora adatto per addestrare lo strato di embedding. Per questo è stato necessario raffinare questo file svolgendo una fase di *pre-processing*. Per prima cosa, si è notato che era possibile che, tra i 130 milioni di nomi di dominio collezionati ci fossero nomi duplicati. Quindi, per diminuire il rumore nel dataset di addestramento dell'embedding, si è deciso di eliminare tutti i nomi di dominio duplicati. Per farlo è bastato inserire i nomi contenuti nel file in una lista e poi creare da quella lista un set, che per definizione è una struttura dati che non ammette duplicati. Questa eliminazione, ha portato il numero di nomi di dominio a calare sensibilmente: il nuovo file, ottenuto scrivendo in ogni riga un elemento del set creato, conteneva 18 milioni di nomi di dominio circa.

Ottenuto il nuovo corpus con i nomi di dominio unici, si è proceduto con un'altra fase di pre-processing, in cui si andavano ad eliminare dei domain name, che contenevano caratteri non ammessi dallo standard internazionale. Infatti era possibile che, tra i nomi di dominio ottenuti, ci fossero dei nomi non conformi allo standard, che erano

presenti a causa di qualche chiamata errata, da parte degli utenti connessi alla rete. Per questo utilizzando la seguente *regular expression*

$$\wedge[A-Za-z0-9._-]*\$$$

si sono eliminati dal corpus tutti i nomi di dominio non ammessi. Tale operazione ha alterato di poco la dimensione del dataset, che comunque ha mantenuto un numero di nomi di dominio sempre di poco superiore a 18 milioni. Il processo spiegato precedentemente a parole è riportato sinteticamente in figura 2.1

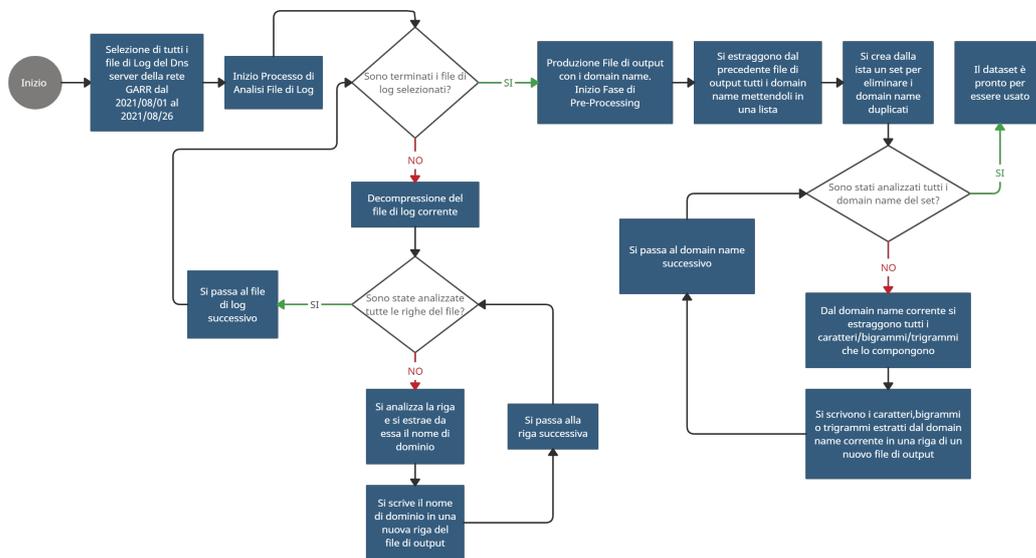


Figura 2.1: Work-flow per la creazione del dataset per FastText

Terminata la fase di pre-processing si è ottenuto perciò un corpus di nomi di dominio sufficiente per iniziare l'addestramento di FastText. Il nostro scopo però, più che ottenere delle rappresentazioni vettoriali, in uno spazio continuo, dei nomi di dominio interi, è stato quello di ottenere delle rappresentazioni vettoriali di caratteri, bigrammi e trigrammi, contenuti nei nomi di dominio del corpus. Per questo, è stato sviluppato uno script in Python3 che separasse con uno spazio i caratteri contenuti nel nome di dominio. Il risultato ottenuto da questa operazione, è il dataset usato per l'addestramento dello strato di embedding FastText della rete, basato su caratteri.

La stessa operazione, fatta per ottenere il dataset per l'embedding basato su caratteri, è stata ripetuta, sempre utilizzando lo script citato prima, anche per

```

108107138157IN - AddrArPa
wwrv4rmmil
SSSUP2ssSUPit
b-019-a30004088240081189c27522f4a4100een5u8ubjp4w7l4tkrcrgjcmeq5avtismcafeecom
wwdisegnidaolorarecom
wwWIFogovIT
41018sr00ns328r8sp34o45o5r8op84p361213590rs2sn9r5394nnpn9n2268s0288npor741q13
6393648s230679r723324n018n6r056n9or607547q8rnr63nr0p01p43961rerp1631850nr08r
qnn4q2080s162859q14rn320s0nsq7qq2s66o70n9931np766rn5645s1i03ssophosxlnet
s-019-20000000001838123c2f92900iu1uh1ube35v7zhq1idrpb7d6iavtismcafeecom
31019sr00nq926s12p374qq05r87pnlq21n213590rs2sn9r539o73qop786q59253qp5r741q13
6393648s247p01n84noqq0oq6s5r26o4r40022qs29603rr00n7r9074nqr24s1qo654o3pp76q8
2922p07np18n8p802qr19534n8032pnqs694p7026s0o3p882i00ssophosxlnet
CORSiUnIB0iT
wwdhakeaocuntnit
linksamboosscm
f1b000000af37746c75646c64656c69766572796d706d6963726f736f6674636f6d80h0d6b04
32webcfs07com
31019sr00ns0qs2p5p35s1rn5r8r29n2907213590rs2sn9r539211r3sp2o7662r69p5r741q13
6393648s247p01n84noqq0oq6p9q26s123p0232s18506rp8qp486752rs31sqrs0q14033s6nq9
p3n4622066296n100s0992nrnn78i03ssophosxlnet
dnospbinterbusinessit
studenti-21-227unib0cC0niIT
58810740blspamcopnet
21529138157in-addrarpa
wwwicupe20univaqit
nmfdujuuufwjdnunimibit
31019sr00ns60ps91p375p3o5r877502r0n213590rs2sn9r53932npp3qqn684212np5r741q13
6393648s247p01n84noqq0oq6s5s26n52040p35038013rsr800s50543862nsp12r4o3pp76q8
2dnr3105o3567qoqr0n41ss4rqop102o96p4530pp88rp5p9ri03ssophosxlnet
ringerlinkcomezproxyuniboit
q-1318mil
59137185155IN - AddrArPa
98a197acd24909711b86c40a8db2e0d0safeframegooglesyndicationcom
22916785209in-addrarpa
0tm4i3vg4qq9d3awafnr1nuqfgiavtismcafeecom

```

Figura 2.2: Prime righe del dataset usato per l'addestramento dello strato di embedding FastText basato su caratteri

ottenere i dataset adatti per addestrare l'embedding basato sui bigrammi, e quello basato sui trigrammi. Lo script stavolta prima ha ricavato tutti i bigrammi o trigrammi presenti in un nome di dominio, e poi li ha separati con uno spazio.

```

18Ba a3 3k m6 65 5s 7v 7e e3 3w wn nw w7 7e e5 5d dt tq q6 6d dp p1 15 5a av vt ts sm mc ca af fe ee ec co om
vp pn nu us se er r0 - -2 29 9v vp pn nu us se er r5 Su un ni im mo oi it
31 lo o1 lo os sr r0 00 09 93 3s s2 2n np p9 9s sp p3 37 78 80 8o op pq q8 8q 6p pp p4 4s s0 01 13 38 8r rp p3 39 91 1s ss so oq qq qr ro oq q7 77
75 57 78 0q q9 95 51 11 iq q1 13 3o o2 2s qs s5 5p pl lr r7 74 41 1q q3 33 36 63 39 93 36 64 48 85 s2 23 3o or ro o2 24 48 89 99 9s s6 6r rr rd pl 16
6n n0 05 5p pn n8 82 27 78 0p p2 26 6s s5 54 4o or rn n7 71 14 40 8q q3 qq q2 2s ss 50 05 54 40 09 9p p2 21 1r rp q9 92 2r r3 3r r4 44 4n n2 27 77
7q qo oq qr r6 63 34 44 4n n7 79 9q q7 7r rp pn ns so o7 70 0o o9 95 56 67 77 78 88 83 q3 q8 83 35 5q q4 43 33 37 78 87 77 73 31 i0 00 0s ss so op ph
ho os sx xl ln ne et
LL LD Da ap p t tC CP PD Dc c MMS SD Dc cS Su un Ni iT Tn Ni IT
Fw wM MG GR Re eG Go o- -g gw WI It
8m m6 6- -0 01 13 3- -0 08 80 00 01 18 89 9c c2 27 75 58 82 2f f4 4a ao 00 0u u3 35 5m mc cg qg qb bp pv vs s9 91 14 4q qm mv v3 3p p7 7p pq q2 2l lg
gq qa av vq qs sm mc ca af fe ee ec co om
71 16 60 01 18 85 51 15 55 51 IN N- -a ad Dd dr rA AR RP Pa
62 22 28 82 27 71 15 57 7I IN N- -A AD DD DR RA AR RP PA
17 7- -1 12 21 1- -1 11 14 4- -1 16 09 9a ap pp pl le eb bo ot ta ap pp pl le ec co om
g1 lk kr rh hu uc cr rg 0g gu un ni ir ro om ma a3 3i it
Pa ar rT TE EC Ci iP PA Az zI IO On ne ep pp PR Rf FV Vg gU UN NI Iu ud di iT
2 21 13 35 54 41 15 51 1i in n- -a ad dd dr ra ar rp pa
iM Mg gg ge eo op p0 0r rt tA AL le ei is sP Pr ra am mb bI IE EN Nt tE EI It
ng gy ye ew ww uu uu ug gt ti ir rc cc cs sm me ei it
SV VW W2 2K K8 8v vs sC CC Cm mU UN Ni ic ca am pP Pu us s- -I IN Nt ti iT
zm mm md ds sb bf fa ad du un ni ir ro om ma a3 3i it
97 78 89 90 05 5i in n- -a ad dd dr ra ar rp pa
63 21 12 29 92 20 06 61 19 93 3i in n- -a ad dd dr ra ar rp pa
pR r1 lX xD Dn ns sN Nt TS S1 Is st ta aT Tl lT
s 57 79 9j js sn n0 0m mp p5 56 6b bi lg gd dg ga av wh hr r3 3s s7 77 73 3q qa av vq qs sm mc ca af fe ee ec co om
11 la a5 51 1m m9 91 11 73 37 7s sa af fl lv v0 0s sw w5 5m ma as sa aq qj ja av vt ts sm mc ca af fe ee ec co om
EL le ea aR RN Ni In nG Gu uM Ni if fg GI It
gI IO Ov ve EC Ci it tT Ta ad dE EL lL LA AS Sc ci ie eN NZ Za aI IT
aa ab ba az zd dw wc cv vl lu un ni ir ro om ma a1 1i it
xq qf fj ju ub ba ac cc cu uh he eu uy yi ir rf fw wf fa a- -p p3 3d d0 0h h9 9- -5 5f f3 32 2d df ff f4 43 3- -c cl li ie en nt tn ns sv v4 4- -s sa
ak ka am ma ai ih hd dn ne et
dh hc cp p- -0 01 18 8e el la ab bs si is ss sa ai it
25 52 20 01 11 10 00 01 15 51 1i in n- -a ad dd dr ra ar rp pa
78 81 11 11 14 47 79 90 0i in n- -a ad dd dr ra ar rp pa
t td dh hb bg gq qm mm ms sz zv v4 4i 1i iq qc ca a9 9p pw w5 5n nd dl lc cb ba av vt ts sm mc ca af fe ee ec co om
s sx xo ol l2 2q qm ml l1

```

Figura 2.3: Prime righe del dataset usato per l'addestramento dello strato di embedding FastText basato su bigrammi

Usando i primi due dataset si sono addestrati i diversi strati di embedding basati su FastText, uno basato sui caratteri e uno basato sui bigrammi. Questi strati di embedding sono rappresentati da dei file *.vec*, cioè dei file di testo in cui ad ogni

```
dol oll lly lyl yiN iNg NgR gRE REU EUn Uni nim imo moR oRE REi Eit
del ele let et0 t08 089 89p 9p0 p0l Oli lIT IOI Oit
eur uro rot oto too ool ols lsd sdt dta tat atu tun uni nib ibo boi oit
ytd tdt dtj tiv jva vaa aat ate ten ene neo eou tun uni niv iwr vri rit
Dns Nsl stV iNo Vot cFl Cfs ISc Scu uol ole tel eLa laz Azl zio IOI OIT
ass sSI STS ISt STE TEn EnZ nZA ZAl AiE iEO IOI OIT
187 871 718 186 864 641 411 113 138 381 8in in- n-a -ad add ddr dra rar arp rpa
hos ost st- t- -15 153 53- 3-1 -12 121 215 151 Sis iss sSa SaI aIT
291 916 167 671 713 136 361 613 130 301 0in in- n-a -ad add ddr dra rar arp rpa
ld lda dPc aP P t tC tCP CP9 P91 911 110 100 002 029 297 97- 7-A -A8 A8E 8EA EA- A-4 -4d 4df d7f f7- 7-B -B4 B41 412 12- 2-E -E1 E11 114 142 429 293
934 34E 4Ec EcF cFc FcD cD0 DOM OMa MaI aIN INs s_M MSd Sdc dCs cSc SCI IRr IRa RAi aIT
472 729 292 920 206 061 619 193 931 3in in- n-a -ad add ddr dra rar arp rpa
HPA PAM AMM MiN iNo N03 03U 3Un Uni nIm Imo mor Ore reI eIT
310 101 019 19s 9sr srR r00 009 093 93r 3rS r54 544 442 423 23p 3p3 p37 374 740 405 05p 5pq pqr qr8 r8o 8or 0r2 r24 242 42q 2q8 q8r 8r8 r8r 8rp rp3 p39
391 91s 1ss sso soq oqq qqr qro roo oq7 q77 776 76n 6n9 n95 05s 5so s0l 013 135 35s 5s1 s1o 1o2 o2n 2n0 n0r 0rp rp1 p1r 1r7 r74 741 41q 1q3 q33 336 363
639 393 936 364 640 48s 8s2 s2o 20q 00p 09s 090 098 093 03s 03s 3sq 041 q1l 10o 00p 00p 0p9 n98 90s 85s s5f 5fr rrr r72 724 244 441 41s 150 500 002 02q 2q0
q00 008 087 871 71r 1rn nrR rR0 r8o 804 04r 4r2 r21 214 14s 45o 50p 086 963 63r 3rS rsn snq nqQ qq7 q76 767 673 73r 3r3 3r6 3p4 6p7 707 702 02n 2nq
nqr qr8 r63 634 344 44r 4r7 r7q 7q5 q50 506 065 065 5pp pp9 p97 970 70o 00r 095 957 576 76p 6p7 p7r 7r9 r98 980 80s 0sn sn0 n03 032 32q 2q2 7q5 7sr sr5
r54 54r 4rp rpR pn9 n9s 9s1 s1o 1o2 028 287 872 726 264 641 411 11s 157 s7n 7nq nq0 q0n 0n7 n74 746 466 665 65s 5s2 s25 252 528 28q 8q1 q10 103 03s 3ss
sso sop oph pho hos osx sxl xln lne net
lam amy mys ysu sur uru run uni nit itn tni nit
170 701 013 132 322 220 206 061 619 193 931 3in in- n-a -ad add ddr dra rar arp rpa
098 986 865 566 565 653 53s 3sy syf yfq fqB qbh bho hoy oyr rpy pyp yvV vpx pxa xar arg rgn gm0 m00 00s 0ss sso sop oph pho hos osx sxl xln lne net
FRo Ron ont tEn tEnD Nd- d-S -ST STR TRr rEA EAa Am- m-0 -02 02U 2Un Uni nib ibo boI oIT
dlm lms mo so- o-t -te tes est std tde des esc scb cbe bel el- l-2 -24 243 43- 3-1 -14 144 44d 4de des esc scd cdI dLa lam ami ml
Www wca cal aio i0d 0Du DuI uil lIl lIO IOI OIT
67r 776 04c 4c2 c29 29d 9dd ddr dfI f17 17c 7ca cab a0d 0dc dce ce5 e5 545 545 52a 2a1 a1a 1a0 a01 015 15e 5ea eab abs bsa saf afe fef efr fra ram
ame mep 600 000 00l 01e 1es esy sYn vYn ndi dlc lca cat ael tli tion onc onco
TiR R00 0cI cIN INI NiT ItE tEs s1u IUu UNi NiG Ige eai eIT
811 116 161 612 127 271 715 157 517 7in in- n-a -ad add ddr dra rar arp rpa
ssm smg mgu gun uni nIT ITn Tni nIT
310 101 019 19s 9ss s00 00n 0nr nr1 r1r 1r8 r8r 8r1 r19 19p 9p3 p39 398 987 87s 7sp sqq pqr qr8 r83 830 30n 0n2 n2n 2n4 n4n 4n9 n9p 9p1 p10 105 053
532 32n 2n2 2n9 29e 960 600 0ns n54 s4p 4p4 4p2 42p 2p3 p31 31p 1p4 pq3 q3q 3q2 q27 27n 7n9 n92 922 22o 2oq oqp qpo por 0r7 r74 741 41q 1q3 q13 136 363
639 393 936 364 648 48s 8s2 s2q 2q1 q12 125 25s 5s1 s12 12q 2qr qrn rn9 n97 970 70o 003 03p 3ps s5s 5s3 5s6 369 694 948 480 808 085 85q 5q2 q2r 2rn
rn9 n9o 9or 0r1 1rn nr1 r5r rsn sn5 n5r 5ro r01 010 105 05o 508 081 812 124 24s 4sp 5pp p9p 9p5 p5s 5s4 s47 477 773 73n 3n3 n30 30p 09q 9q0 q0o
00o 000 000 002 025 259 597 97r 7rS rS1 s11 11r 1r4 r49 49q 9q1 q11 11p 1p4 p4s 4s0 s09 09q 9q0 030 030 308 082 82p 2p9 p9o 9o7 077 779 797 972 729 291
```

Figura 2.4: Prime righe del dataset usabile per l'addestramento dello strato di embedding FastText basato sui trigrammi

carattere o bigramma, è associato un vettore di 128 componenti. I risultati sono mostrati in 2.5 e 2.6

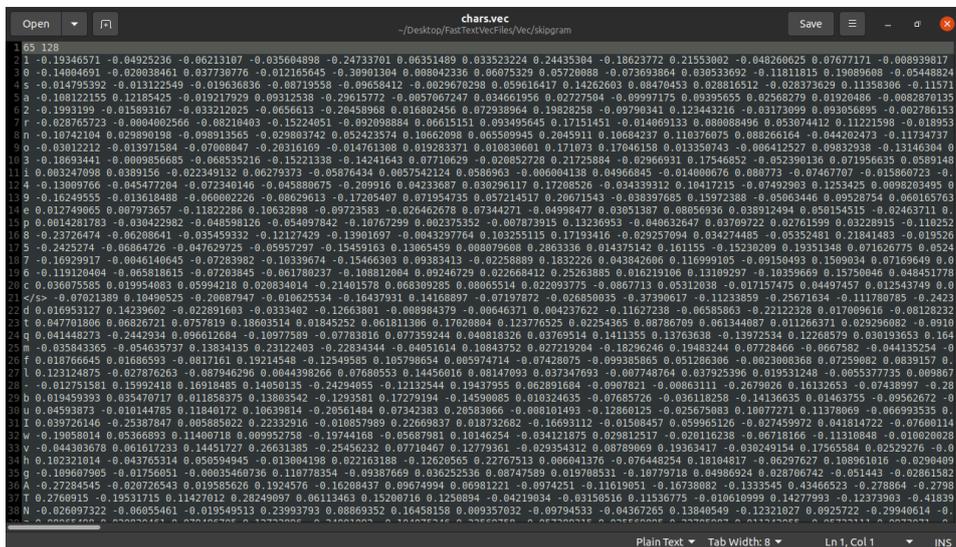


Figura 2.5: File di embedding per i caratteri ottenuto dal training di FastText

Purtroppo non è stato possibile portare a compimento il training dello strato di embedding basato sui trigrammi a causa della grande quantità di risorse richieste.

```

Open  bigrams.vec  Save
~/Desktop/FastTextVecFiles/Vec/skipgram

1 om 0.27191266 0.2205701 0.10044126 -0.29902307 0.025041047 -0.041049695 -0.35302274 -0.20340051 0.040801703 0.19106592 -0.12504523 -0.2019526 -0.6110046
2 co 0.266636746 0.09734428 0.12972365 -0.07943395 -0.015239271 -0.3004948 -0.44295257 0.0011578673 -0.22076145 0.024547752 -0.16116086 0.29117515 0.20713514
3 om 0.47106405 0.23269786 -0.16589648 -0.10979809 -0.34230077 0.39474386 0.14781429 -0.6841476 -0.14474091 0.0152968895 -0.45191652 -0.22175969 0.150454988 -
5 co 0.15125735 0.02637397 -0.10263981 -0.16998449 0.24570176 0.21689081 -0.45538354 -0.28592327 0.14475283 0.19952054 -0.4616303 -0.05539575 -0.017202444 -
6 it -0.07925974 -0.18061888 -0.31591013 0.045194365 -0.13454788 0.21651697 0.22743631 0.07186289 0.2624647 0.11731566 0.03556395 -0.05284636 0.33191413 0.15
7 ca 0.12919297 -0.33767074 -0.55449026 -0.2140558 -0.22347994 -0.1467932 0.106313694 -0.11662237 -0.40027592 -0.13227479 -0.00704277 -0.20975065 0.11853129
8 ai 0.21462825 0.026008 -0.06615497 0.05912696 -0.35142583 -0.3666428 -0.48903483 -0.0887834 0.07339945 -0.18044937 0.3240493 0.013279193 0.310
9 ec 0.55342525 -0.055170784 -0.09261009 0.08279728 -0.14522758 0.56468505 -0.26720977 0.07755986 -0.16350667 0.3795375 -0.29455745 -0.15422718 -0.047352463
10 in 0.46272153 -0.23920432 -0.20480183 0.10936754 -0.049804153 -0.10662677 -0.45389586 0.21928456 -0.05349386 0.4100334 -0.37069175 0.3088154 0.34883076 0.1
11 ad 0.54222543 -0.54847753 0.18732928 0.63065314 0.20803949 -0.12537718 -0.27712834 0.34957069 -0.28597862 0.47906647 -0.7886806 0.113292626 0.3308417 -0.00
12 rc 0.08165174 -0.09509957 -0.19339975 0.04593462 -0.00338898 0.059049308 0.117929665 0.57052886 -0.0519986 0.022424179 0.3901531 -0.17307983 0.06649898 -
13 bl 0.11878948 -0.26673636 0.3762152 -0.06826989 -0.047454037 0.06850822 -0.24546714 -0.13387594 -0.1729089 0.28108502 -0.52268296 0.3240493 0.013279193 0.310
14 av 0.08152329 -0.15507872 0.42166518 0.08625687 0.06773039 0.20175417 0.20042199 -0.20276216 -0.5578888 0.5110611 -0.24226886 0.09381672 0.4893093 0.38669
15 ne 0.030247971 -0.22083414 -0.31448138 -0.12869106 -0.1074953 -0.099246616 0.030613597 0.30439064 -0.0686415 0.38065627 0.14849273 0.15097078 -0.079486616
16 rc 0.0032327575 -0.1103754 -0.11318946 0.4486461 0.02813767 -0.53728 0.2497291 -0.1533988 -0.5807605 0.009302465 -0.71799804 0.5238669 0.37813344 -0.4666
17 fp 0.33901964 -0.3423445 0.024919052 0.38461378 -0.47678626 -0.455639 0.022372998 0.23571467 -0.07702944 0.6018591 -0.405998 0.47412384 0.39242265 -0.2966
18 in 0.116325257 -0.18715426 -0.2677775 -0.23028102 -0.31041223 -0.1432603 0.21076891 0.15513995 0.25236586 -0.3315413 -0.08823488 -0.2848497 -0.088519245 -0.
19 ad 0.040194802 0.06640737 -0.28984758 0.3847259 -0.20493674 -0.28232682 -0.31841043 -0.34331256 -0.068771616 0.04301785 -0.0341677 0.33774668 0.8124225 -0.
20 nc 0.7259266 -0.2335577 -0.55030656 0.46137684 -0.7302145 0.007829612 -0.059021 -0.40677437 -0.27420884 -0.18630524 -0.4342115 0.0819963 0.7225967 0.182532
21 lg 0.32223945 -0.14934418 0.08445871 0.18283550 -0.030824102 -0.37442115 -0.1502405 0.1590907 -0.19731177 0.21659597 -0.20737689 0.060846232 0.36097917 0.0
22 ss 0.035712034 -0.02080169 0.33227718 0.6607919 -0.07966034 -0.44341314 -0.21199313 -0.15680388 0.25117698 -0.076645966 0.015585327 -0.2608889 -0.17973953
23 ke 0.2188791 -0.20716666 -0.066849684 0.37071785 0.07307328 0.09115695 0.01069344 0.6423134 0.031051885 0.4242239 -0.58261657 -0.002369584 0.2494575 -0.094
24 io 0.23312838 0.12752656 0.3310888 -0.061263368 0.07486099 -0.149072 -0.07731663 0.004898514 -0.41744083 -0.014474537 -0.030742635 0.0530752 0.5803371 0.5
25 ne 0.06059180 -0.101033279 -0.21211515 -0.028012056 -0.20015846 0.00970864 -0.00709061 0.3078102 -0.0090131 0.30049507 0.1344979 -0.11440331 -0.0734984 -0.
26 rc 0.49102818 -0.19125677 0.074013485 0.20657702 0.3107297 -0.15611254 -0.21185365 -0.099819094 -0.15534383 -0.055781725 0.21395774 -0.09670597 0.24290599
27 ar 0.07401295 -0.5836685 0.015334247 0.20715742 -0.20570712 0.13072194 -0.76375645 -0.15937142 0.14210606 0.3917268 -0.4589804 0.7596265 0.38738126 -0.739
28 ad 0.2405222 -0.21044746 -0.40022165 0.678138 -0.24579851 0.14185238 -0.6976423 0.42562056 -0.30651224 -0.22386569 -0.4076203 0.71088625 0.30569556 -0.4230
29 sm 0.14957115 -0.504791 -0.22399605 -0.34183073 -0.15731323 0.42565536 -0.09108126 0.25100643 -0.35066298 -0.0941374 -0.002821133 -0.3495313 -0.27486094 -0.
30 nc 0.1201612 -0.270216 -0.20820194 -0.20143797 -0.11240707 0.22458078 -0.03214069 -0.10220415 -0.1460147 0.21668996 -0.17773215 -0.47174054 -0.2577857 -0.
31 bl 0.49086175 -0.31227964 0.02775993 0.10022034 -0.3957925 0.23279426 -0.03635544 -0.002108431 -0.07252817 0.40189783 -0.25481707 0.022567518 0.29039213 0
32 av 0.27328956 -0.25719163 -0.28473392 -0.050823958 -0.5784375 0.4568682 -0.10253971 0.16751823 -0.38737404 0.03020176 0.011310479 -0.3225858 -0.09188952 -0.
33 ne 0.2980842 -0.071016455 0.14609219 0.555375 -0.37091783 -0.65391356 -0.026837822 -0.21248142 -0.027233716 0.12326563 0.010574458 -0.06130006 -0.49169397
34 rc 0.05612053 -0.25601726 -0.110561308 0.17301001 -0.34020293 -0.21029210 -0.26078016 -0.1764997 0.20929293 0.45686638 -0.55085077 0.5042039 0.5543103 -0.
35 lg 0.25780186 -0.07329228 -0.0087404 0.33917135 0.11565028 0.25623867 0.106267974 0.40429482 -0.08303458 0.38043504 0.663777 -0.1634633 0.4061293 -0.01145
36 io 0.17159832 -0.44440773 -0.055359326 0.34367412 -0.25021198 -0.13400914 -0.048565544 0.64765656 0.007305034 0.17218755 -0.45629707 0.1609921 -0.24068852 0.
37 un 0.20149633 -0.11347588 -0.15757996 -0.2998461 -0.3798961 -0.080800405 0.1712479 -0.071699806 0.19741416 -0.06498238 -0.10510674 -0.21768892 0.16323557 0.0
38 li 0.4196548 -0.35240924 0.13151766 0.08688757 -0.06725469 -0.21608178 -0.22078772 -0.13872644 -0.27906567 0.18277612 -0.27747843 0.26170102 0.25773093 0.5

```

Figura 2.6: File di embedding per i bigrammi ottenuto dal training di FastText

2.2.2 Dataset per il test della rete LSTM.MI

Una volta ricavati i file per gli strati di embedding del classificatore, il passaggio successivo è stato quello di costruire un dataset con cui riaddestrare la rete LSTM.MI. Tale dataset, chiaramente, deve contenere un numero di nomi etichettati come benevoli non di troppo superiore a quello di nomi malevoli. Il massimo bilanciamento che ci potevamo permettere, come indicato anche nell'articolo di Duc Tran et al.[14], era avere un nome di dominio marcato come DGA, per ogni mille nomi di dominio marcati come alexa.

Quindi, per cercare di costruire un dataset valido per il classificatore ci siamo avvalsi di tutti gli strumenti e le risorse presentate sopra. Il periodo di osservazione scelto per la creazione del dataset è stato l'intervallo che va dal 26 agosto 2021 al 13 settembre 2021, in quanto si è pensato che le due settimane indicate potessero essere sufficienti, per avere un dataset non troppo sbilanciato. Per questa operazione, si sono scaricati giornalmente, nel periodo scelto, tutti i log del DNS server della rete GARR e i feed del Bambenek Consulting. Invece, per le motivazioni già accennate nella sezione dedicata, il feed di Majestic Million è stato scaricato solo nei giorni 26 e 30 agosto e nei giorni 2,6,9 e 13 settembre.

A questo punto, si è redatto uno script in Python3, per rendere automatica

l'etichettatura dei nomi di dominio nel dataset. Il processo svolto dal codice scritto, si avvale dei file scaricati in precedenza, e comincia l'etichettatura dal giorno 26 agosto, seguendo questa serie di passaggi:

- Vengono caricati, in una struttura di tipo dizionario del linguaggio Python, tutti i nomi di dominio malevoli contenuti nel Bambenek Feed relativo al giorno di partenza. La scelta del dizionario come struttura dati viene fatta, per diminuire la complessità computazionale dell'algoritmo di etichettatura.
- Viene svolta la stessa operazione per il feed Majestic del giorno corrente, cioè tutti i domini benevoli sono caricati all'interno di un altro dizionario Python.
- Viene decompresso il file *.log* relativo alla prima ora del giorno di partenza. Inizia quindi il processo di analisi: per ogni chiamata a nome di dominio presente nel file di log, si controlla se il domain name è contenuto nel dizionario dei nomi benevoli, e se lo è viene scritto nel dataset marcato come **legit**. Altrimenti, si controlla se il domain name è contenuto nel dizionario contenente i nomi malevoli, e se lo è, viene inserito nel dataset marcato come **dga**.
- Se il domain name, processato attualmente, non era in nessuno dei dizionari, viene semplicemente saltato. Invece, se si trova un corrispondenza in uno dei dizionari precedenti, oltre a scriverlo sul dataset lo si inserisce in una lista di dizionari, denominata *analyzed_traffic*, di cui il primo elemento è un dizionario contenente tutti i nomi benevoli finora processati, e il secondo è un dizionario analogo al precedente, ma per i nomi DGA.
- Terminato il processo di analisi del primo nome di dominio, si passa ad analizzare il successivo. Però, oltre a controllare se è contenuto nel dizionario ottenuto dal feed Majestic o in quello ottenuto dal Bambenek Feed, stavolta, si controlla anche se esso è contenuto o meno in uno dei due dizionari presenti in *analyzed_traffic*. Se lo è il domain name viene saltato, altrimenti è processato nella maniera descritta nei punti precedenti.

A partire dal terzo nome di dominio processato, si ripete il processo descritto sopra per ogni nome, fino a che non si esaurisce il file relativo alla prima ora del giorno. A quel punto si passa ai file successivi e si ripetono gli stessi passaggi, fino a che non si esauriscono i file di log relativi a un'intera giornata. Quindi si passa alla giornata successiva: si ricarica il feed del Bambenek Consulting relativo al nuovo giorno e, solo se necessario, il feed del Majestic Million. Viene decompresso il file *.log* relativo al nuovo giorno analizzato, e il processo si ripete automaticamente fino all'ultimo giorno indicato.

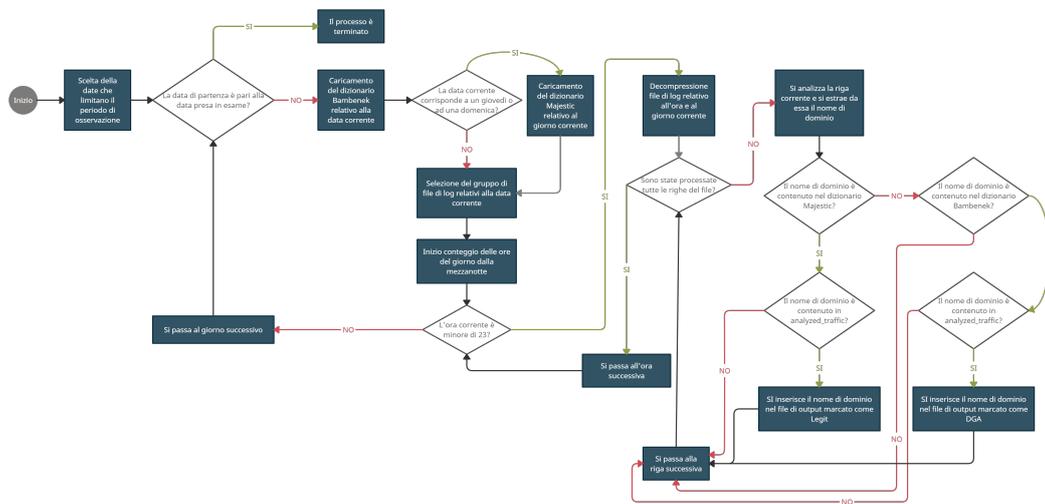


Figura 2.7: Work-flow per la creazione del dataset per il classificatore LSTM

Il dataset che si ottiene dal processo di elaborazione descritto, e sintetizzato in figura 2.7, è un file *.csv*, che contiene per ogni riga, oltre al nome di dominio, anche altre informazioni:

- Nella prima colonna c'è l'etichetta che ci dice se il nome è benevolo, o malevolo (*legit* o *dga*)
- Nella seconda colonna si trova la famiglia a cui appartiene il nome di dominio, quindi o la famiglia di *dga* nel caso sia un nome malevolo, o *alexa* nel caso sia benevolo
- Nella terza colonna si trova il nome di dominio, in cui però viene eliminato il punto tra il TLD e il second level domain

- Nella quarta colonna si ha il nome di dominio
- Nella quinta colonna, ci sono i caratteri che compongono il nome di dominio separati da spazio
- Nella sesta colonna ci sono i bigrammi che compongono il nome di dominio separati da spazio
- Nella settima colonna ci sono i trigrammi che compongono il nome di dominio separati da spazio

Eseguendo tutto questo processo di creazione del dataset, nel periodo indicato, purtroppo il risultato ottenuto è stato diverso da quello sperato. Il dataset prodotto, è risultato essere un dataset fortemente sbilanciato, in quanto su 45000 nomi di dominio trovati, solo due nomi sono risultati essere marcati come DGA. Approfondendo il fenomeno e verificando che non ci fossero errori sistematici nello script di creazione del dataset, purtroppo si è arrivati alla conclusione che nel periodo di osservazione scelto, c'erano pochissime macchine connesse alla rete GARR e allo stesso tempo controllate da un botmaster. Questo ha reso il dataset ottenuto inutilizzabile per qualsiasi tipo di esperimento. Per questo motivo il DII ha concesso la possibilità di utilizzare dei dataset, costruiti in maniera sintetica, di diverse taglie, per ottenere dei risultati sperimentali concreti. Le taglie di dataset fornite sono state tre, e con quelle sono stati svolti gli esperimenti, discussi nel capitolo 3:

- Un dataset contenente 4200 nomi di dominio benevoli e malevoli
- Un dataset contenente 8400 nomi di dominio benevoli e malevoli
- Un dataset contenente 16800 nomi di dominio benevoli e malevoli

2.3 Configurazioni Sperimentali

Concludiamo il capitolo illustrando le configurazioni sperimentali, adottate per ottenere i risultati e raggiungere l'obiettivo finale di questo lavoro. Nonostante il numero di esperimenti in totale, avendo tre taglie di dataset, sia nove, descriveremo

solo tre configurazioni sperimentali. Infatti il set-up sperimentale nel passaggio da una taglia di dataset più piccola a una più grande, o viceversa, rimane identico. Le tre configurazioni sperimentali che descriveremo si differenzieranno, infatti, sono nel modo in cui è realizzato lo strato di embedding del classificatore. Una volta fissato l'embedding, il processo di training è identico in tutti i casi e perciò, verrà descritto un'unica volta

2.3.1 Embedding Randomico

Il primo tipo di embedding adottato, ricalca quello dell'esperimento svolto dal ricercatore Duc Tran et al.[14]. Come avevamo già accennato nel capitolo 1, una rete neurale non può processare un input in cui sono presenti caratteri, ma può processare solo numeri interi. Per questo, è inserito a monte del classificatore LSTM.MI, prima dello strato nascosto della rete, uno strato di embedding, che in questo caso è detto **randomico**.

Usando questa tipologia di embedding, per prima cosa si estrae la rappresentazione in caratteri (la quinta colonna del file *.csv* descritto nella sezione precedente) di tutti i nomi di dominio presenti nel dataset, ottenendo una lista di liste, in cui la componente *i*-esima è una lista, contenente tutti i caratteri del nome di dominio *i*-esimo. Quindi si scorre la lista e, ogni volta che all'interno delle liste componenti si rileva un carattere mai rilevato in precedenza, lo si aggiunge a un dizionario in cui la chiave è il carattere e il valore è il numero intero ad esso assegnato casualmente.

Creato questo dizionario, che definiremo *dizionario dei caratteri validi*, si ripeterà il processo di scansione della lista di liste, solo che in questa fase, ogni carattere presente in ogni lista componente, verrà sostituito con il numero intero assegnatogli secondo il dizionario. In questa maniera la lista di liste di caratteri, si trasformerà in una lista di liste di interi. Questa lista di liste infine, viene trasformata in una matrice con un numero di righe pari al numero di liste componenti, e un numero di colonne pari alla dimensione della lista più lunga.

Svolto questo processo il dataset, formato da liste di caratteri, viene convertito in una matrice di interi, che è l'input che la rete si aspetta di ricevere nel suo strato

nascosto. Questa operazione è vitale, affinché si possa costruire il modello della rete LSTM.MI e iniziare il training del classificatore.

2.3.2 Embedding basato su FastText

Oltre al tipo di embedding descritto nella sezione precedente, si è deciso di sperimentare altri due tipi di embedding per il classificatore. Questi strati di embedding sono realizzati mediante i file *.vec* ottenuti dall'addestramento precedente di FastText, e li definiremo rispettivamente **Embedding FastText basato su caratteri** e **Embedding FastText basato su bigrammi**.

Sfruttando questo nuovo tipo di embedding i passaggi da seguire per convertire i caratteri, presenti all'interno dei nomi di dominio del dataset, sono più complicati. Per prima cosa si estrae dal dataset la rappresentazione in caratteri o in bigrammi dei nomi di dominio, ottenendo una lista di liste simile al caso dell'embedding randomico. Dopodichè sfruttando un oggetto della classe `Tokenizer` di Python si va a creare un dizionario, denominato *word_index*, con tutti i caratteri o bigrammi presenti all'interno dei nomi di dominio, in cui ad ogni carattere o bigramma si associa un numero intero. Quindi secondo l'associazione creata con questo dizionario, si sostituiscono i caratteri, o bigrammi, presenti nelle liste componenti con il numero intero associato. Si ottiene così una lista di liste di interi, che viene trasformata in una matrice, allo stesso modo di come avveniva nel caso dell'embedding randomico.

Ora, mentre nel caso dell'embedding randomico il processo di creazione dello strato di embedding terminava così, nel caso di embedding FastText si svolge un ulteriore passaggio: sfruttando il file *.vec*, ottenuto in precedenza, o per i caratteri, o per i bigrammi, creo un dizionario in cui la chiave è un carattere o un bigramma contenuto nel file, e il valore è l'array di coefficienti assegnatogli durante l'addestramento di FastText. Ottenuto questo nuovo dizionario, si va quindi a creare la **matrice di embedding**, che è una matrice in cui la riga *i*-esima contiene la rappresentazione vettoriale a 128 caratteri, estratta dal file *.vec*, del carattere o bigramma, a cui nel *word_index* è stato associato il numero *i*.

Tale matrice di embedding ottenuta, viene poi usata per la costruzione del modello del classificatore LSTM.MI. Infatti nella costruzione dello strato di embedding del classificatore, possibile sia nel caso randomico che nel caso FastText con un metodo Python, si inserisce come parametro la matrice di embedding. In questa maniera, quando il modello riceverà in input la matrice costruita prima dalla lista di liste, sulla base del numero intero che è stato assegnato ad ogni carattere o bigramma, ricercherà la riga corrispondente nella matrice di embedding, così da sostituire ogni intero della matrice con un vettore a 128 componenti, ottenendo un tensore a tre dimensioni.

2.3.3 Addestramento del classificatore

Fissato lo strato di embedding, FastText o randomico, il processo di training del classificatore si svolge allo stesso modo in entrambi i casi. La metodologia utilizzata per l'addestramento del classificatore, è la **k-fold cross-validation**, che consiste nel reiterare l'addestramento per un numero di passi, definiti fold, pari a k. Il motivo per cui viene scelta questa tecnica, è che essa ci permette di valutare le prestazioni del classificatore, anche se usiamo dataset relativamente piccoli per allenarlo.

Usando questa metodologia, il primo passo da compiere è quello di dividere il dataset, in dataset di training e dataset di test. Questa divisione però non rimane costante durante tutti i passi, o fold del processo: prima di iniziare il processo di addestramento infatti il dataset viene diviso in un numero di gruppi pari al numero k specificato. Dopodiché viene scelto casualmente uno dei gruppi come dataset di Test, mentre gli altri gruppi vengono uniti per formare il training dataset. A quel punto viene creato il modello del classificatore LSTM.MI, con il tipo di embedding scelto, e inizia l'addestramento. Alla fine del processo, si elaborano i risultati ottenuti dal modello, sulla base delle predizioni che esso ha fatto sui campioni contenuti nel test dataset, e si salvano i risultati. Terminato quindi l'addestramento si riparte dall'inizio: viene selezionato un altro dei quattro gruppi non ancora usato per il test, casualmente, e si ripete tutto il processo di addestramento.

Alla fine quello che si ottiene, sono k report, generati a seguito dei k fold svolti, che possono essere utilizzati per stabilire quali sono le prestazioni della rete. Nel nostro caso specifico si è considerato k pari a 5, cioè si è iterato per cinque fold, in maniera tale che ad ogni fold il dataset di test fosse il 20% del totale. Ad ogni fold si sono salvati i risultati delle metriche che ci interessavano, sia per ogni famiglia, che al livello globale, e poi si è fatta la media semplice delle metriche contenute nei cinque report, per ottenere un unico report finale, mediante cui svolgere il processo di analisi.

Per ogni taglia di dataset, perciò si utilizza la metodologia espressa sopra, e si ripete il training del classificatore provando tre diversi strati di embedding:

- Il primo esperimento consisterà nell'addestrare il classificatore con embedding randomico.
- Il secondo esperimento consisterà nell'addestrare il classificatore con embedding FastText basato su caratteri
- Il terzo esperimento consisterà nell'addestrare il classificatore con embedding FastText basato su bigrammi

Capitolo 3

Risultati sperimentali

Terminati gli esperimenti condotti, con le configurazioni descritte nel capitolo 2, si procede alla presentazione, e analisi dei risultati.

3.1 Esperimenti con Dataset da 4200 nomi di dominio

```
Class report:
precision  recall  f1-score  support
conficker  0.544444  0.247059  0.306333  17.000000
corebot    0.957310  0.976471  0.966003  17.000000
cryptolocker  0.311126  0.288235  0.292053  16.000000
dircrypt   0.301236  0.176471  0.216364  17.000000
emotet     0.746709  0.800000  0.758195  17.000000
fobber     0.441168  0.709559  0.534029  16.000000
gozi       0.400000  0.107353  0.161111  17.000000
kraken     0.500000  0.155882  0.226560  17.000000
matsnu     0.679444  0.541176  0.575947  17.000000
murofet    0.822367  0.658824  0.724963  17.000000
necurs     0.565714  0.119853  0.186950  17.000000
nymaim     0.000000  0.000000  0.000000  16.000000
padcrypt   0.796049  0.905882  0.841415  17.000000
pushdo     0.764923  0.450735  0.541547  17.000000
pykspa     0.596032  0.204412  0.289992  17.000000
qadars     0.686923  0.870588  0.759994  17.000000
ramdo      0.803893  0.903676  0.848690  16.000000
ramnit     0.470000  0.261029  0.333854  17.000000
ranbyus    0.414438  0.330882  0.353892  17.000000
rovnix     0.517782  0.786765  0.594062  16.000000
simda      0.629924  0.497059  0.536034  17.000000
suppobox   0.599286  0.420588  0.464029  17.000000
symmi      0.923947  0.962500  0.941221  17.000000
tinba      0.370810  0.358824  0.358367  17.000000
vawtrak    0.200000  0.011765  0.022222  17.000000
alexa      0.771083  0.936190  0.845030  420.000000
accuracy   Precision Recall  F1_score  support
macro avg  0.569793  0.487761  0.487648  840.000000
weighted avg  0.666714  0.703333  0.659573  840.000000

          macro    micro
f1_score  0.487648  0.703333
precision  0.569793  0.703333
recall    0.487761  0.703333

Overall accuracy = 0.7033333333333334
```

Figura 3.1: Report finale esperimento con Random Embedding

Capitolo 3 Risultati sperimentali

La figura 3.1, riporta i risultati ottenuti dall'addestramento del classificatore con l'embedding Randomico. Generalmente, le prestazioni del classificatore risultano essere discrete, data l'accuracy del 70.3%. É possibile notare che, mentre ci sono delle famiglie di malware, che ottengono ottime prestazioni in termini di F1-score, come Corebot, Emotet, Qadars, Ramdo, e Symmi, invece per altre famiglie, le prestazioni della rete sono pessime. Tra queste famiglie per cui si hanno pessime prestazioni vi sono Gozi, Necurs, Nymaim, Ramnit e Vawtrak. In particolare per Nymaim la rete ottiene un valore di F1-score nullo: tale avvenimento dipende dal fatto che, su 17 campioni di nomi di dominio nymaim presenti nel dataset di test, ben 15 sono confusi con nomi benevoli, appartenenti alla famiglia alexa, e altri due sono scambiati per altre due famiglie di malware. La matrice di confusione, che testimonia questo avvenimento, è riportata in Appendice in 5.1.

In figura 3.2, sono visibili i risultati ottenuti dall'addestramento del classificatore con embedding FastText basato su caratteri. Dai risultati presentati nel report, si

```
Class report:
precision  recall  f1-score  support
conficker  0.526061  0.423529  0.460655  17.000000
corebot    1.000000  0.988235  0.993939  17.000000
cryptolocker 0.474791  0.346324  0.396580  16.000000
dircrypt   0.293997  0.317647  0.298284  17.000000
emotet     0.810212  0.988235  0.885698  17.000000
fobber     0.491348  0.781618  0.601113  16.000000
gozi       0.503571  0.281618  0.348000  17.000000
kraken     0.433983  0.265441  0.326891  17.000000
matsnu     0.766366  0.701471  0.698896  17.000000
murofet    0.856909  0.752941  0.797200  17.000000
necurs     0.306061  0.200735  0.239485  17.000000
nymaim     0.100000  0.011765  0.021053  16.000000
padcrypt   0.961667  0.870588  0.913636  17.000000
pushdo     0.825275  0.451471  0.566304  17.000000
pykspa     0.676540  0.463971  0.545747  17.000000
qadars     0.977124  0.917647  0.944606  17.000000
ramdo      0.941667  0.880882  0.909232  16.000000
ramnit     0.319643  0.202941  0.229584  17.000000
ranbyus    0.599740  0.585294  0.579741  17.000000
rovnix     0.953513  0.940441  0.946459  16.000000
simda      0.863312  0.534559  0.655589  17.000000
suppobox   0.496752  0.275000  0.344354  17.000000
symmi      0.977124  1.000000  0.988225  17.000000
tinba      0.507729  0.439706  0.463143  17.000000
vawtrak    0.783333  0.118382  0.202222  17.000000
alex       0.786852  0.947143  0.858866  420.000000
-----
Precision  Recall  F1_score  support
accuracy   0.748571
macro avg  0.662830  0.564907  0.585212  840.000000
weighted avg 0.722703  0.748571  0.716824  840.000000

          macro    micro
f1 score  0.585212  0.748571
precision 0.662830 0.748571
recall    0.564907 0.748571

Overall accuracy = 0.7485714285714286
```

Figura 3.2: Report finale esperimento con FastText Embedding basato su caratteri

3.1 Esperimenti con Dataset da 4200 nomi di dominio

assiste a un miglioramento generale, delle prestazioni del classificatore, per tutte le famiglie rispetto all'embedding randomico. Si continuano infatti, ad ottenere ottime prestazioni per le famiglie per cui si ottenevano ottime prestazioni nel caso dell'embedding randomico. In più, si assiste a un miglioramento netto delle prestazioni della rete per famiglie come Gozi, Necurs, Pykspa, Rovnix e Vawtrak, che nel caso precedente avevano un F1-score certamente inferiore. Purtroppo, nonostante i miglioramenti, sono da segnalare le pessime prestazioni, che si ottengono per le famiglie Nymaim e Ramnit: per la famiglia Ramnit infatti, il cambio di embedding non produce nessun miglioramento in termini di F1-score; per la famiglia nymaim ancora una volta l'F1-score è molto basso, a causa del fatto che su 17 campioni Nymaim, ben 16 sono scambiati come benevoli, come riportato nella matrice di confusione in Appendice(5.2). Al livello globale comunque il cambio di embedding produce un miglioramento dell'accuracy che arriva quasi al 75%.

```
Class report:
precision  recall  f1-score  support
conficker  0.518182  0.400000  0.448127  17.000000
corebot    1.000000  0.988235  0.993939  17.000000
cryptolocker  0.462103  0.394853  0.401992  16.000000
dircrypt   0.362491  0.329412  0.299953  17.000000
emotet     0.977124  0.964706  0.970400  17.000000
fobber     0.434804  0.373529  0.395364  16.000000
gozi       0.736401  0.491912  0.530669  17.000000
kraken     0.621885  0.619118  0.617411  17.000000
matsnu     0.807546  0.683824  0.712597  17.000000
murofet    0.769195  0.741176  0.743075  17.000000
necurs     0.448146  0.464706  0.444894  17.000000
nymaim     0.472857  0.178676  0.253771  16.000000
padcrypt   0.962402  0.870588  0.912326  17.000000
pushdo     0.839945  0.819853  0.824298  17.000000
pykspa     0.784786  0.737500  0.744074  17.000000
qadars     0.974167  0.870588  0.918903  17.000000
ramdo      0.884676  0.964706  0.919318  16.000000
ramnit     0.389892  0.345588  0.355263  17.000000
ranbyus    0.538066  0.595588  0.555586  17.000000
rovnix     0.956656  0.854412  0.897967  16.000000
simda      0.779524  0.688235  0.720073  17.000000
suppobox   0.552329  0.508088  0.516485  17.000000
symmi      1.000000  1.000000  1.000000  17.000000
tinba      0.389744  0.332353  0.332736  17.000000
vawtrak    0.733846  0.512500  0.602829  17.000000
alexas     0.855608  0.927143  0.889292  420.000000
Precision  Recall   F1_score  support
accuracy   0.778571
macro avg  0.702014  0.640665  0.653898  840.000000
weighted avg  0.775720  0.778571  0.767182  840.000000

          macro    micro
f1 score  0.653898  0.778571
precision 0.702014 0.778571
recall    0.640665 0.778571

Overall accuracy = 0.7785714285714286
```

Figura 3.3: Report finale esperimento con FastText Embedding basato su bigrammi

Infine in figura 3.3, si possono osservare i risultati ottenuti dall'addestramento del

classificatore, con embedding FastText basato su bigrammi.

Come visibile dal report, il passaggio dall'embedding basato sui caratteri a quello basato su bigrammi migliora le prestazioni, portando l'accuracy del modello al 78%. Infatti, non solo sono mantenute ottimali le prestazioni per quelle famiglie, per cui già nei casi precedenti si ottenevano ottimi valori delle metriche, ma si riesce anche a produrre miglioramenti significativi per le famiglie, per cui le prestazioni del classificatore nei casi precedenti erano pessime. Tale miglioramento, in F1-score, è riscontrato specialmente per le famiglie Gozi, Kraken, Necurs, Nymaim, Pushdo, Pykspa, Suppobox e Vawtrak. Specialmente quest'ultima sembra beneficiare molto del passaggio da caratteri a bigrammi. Oltre a questi miglioramenti, si assiste però anche dei peggioramenti, nei valori di F1-score per alcune famiglie, nel passaggio da caratteri a bigrammi, come ad esempio Fobber. Infine ancora una volta, come mostrato nelle matrici di confusione in Appendice(5.3), più del 50% dei campioni della famiglia Nymaim sono confusi come campioni della famiglia Alexa.

3.2 Esperimenti con Dataset da 8400 nomi di dominio

Dal report in figura 3.4, si nota che, l'aumento di dimensioni del dataset produce un miglioramento diffuso delle prestazioni per quasi tutte le famiglie di malware. In particolare famiglie come Conficker, Cryptolocker, Gozi, Kraken, Pykspa, Rovnix e Emotet fanno registrare grossi aumenti in F1-score. In contro tendenza, rimane la famiglia Ramnit, per cui le prestazioni del classificatore rimangono simili, a quelle ottenute con la taglia di dataset più piccola. Con il cambio del dataset, rimane però la tendenza del classificatore, a scambiare nomi di dominio appartenenti a famiglie come Gozi, Nymaim e Vawtrak, come nomi benevoli, come è visibile nella matrice di confusione 5.4 in appendice. Infine, l'aumento del volume del dataset, globalmente, produce un aumento dell'accuracy importante, che passa dal 70% al 76% arrivando quasi ad equiparare, quella ottenuta dal modello con FastText Embedding basato su caratteri, addestrato però con un dataset più piccolo.

Il report in figura 3.5, testimonia che, anche nel caso in cui sia usato un embed-

3.2 Esperimenti con Dataset da 8400 nomi di dominio

```
Class report:
precision  recall  f1-score  support
conficker  0.616407  0.410695  0.491643  34.000
corebot    0.954902  0.988057  0.970926  34.000
cryptolocker  0.525671  0.443850  0.464572  33.000
dirccrypt  0.489953  0.264706  0.326443  34.000
emotet     0.890346  0.982353  0.933345  34.000
fobber     0.561371  0.804635  0.653930  33.000
gozi       0.825000  0.226203  0.325123  34.000
kraken     0.738723  0.390731  0.507753  34.000
matsnu     0.759782  0.659180  0.679562  34.000
murofet    0.900948  0.691087  0.781004  34.000
necurs     0.622887  0.218895  0.317118  34.000
nymaim     0.486111  0.071658  0.117978  33.000
padcrypt   0.948836  0.863280  0.902307  34.000
pushdo     0.838593  0.615686  0.696061  34.000
pykspa     0.561459  0.398396  0.453049  33.000
qadars     0.931238  0.885561  0.907580  34.000
ramdo      0.916152  0.945989  0.929618  33.000
ramnit     0.340976  0.294118  0.308932  33.000
ranbyus    0.568776  0.665062  0.611443  34.000
rovnix     0.843522  0.898039  0.861751  33.000
simda      0.717809  0.645276  0.672765  34.000
suppobox   0.546751  0.526916  0.516191  34.000
symmi      0.964118  0.957754  0.959402  33.000
tinba      0.527637  0.611765  0.544128  33.000
vawtrak    0.562232  0.165241  0.244532  33.000
alexas     0.805842  0.945476  0.869495  840.000
          Precision Recall  F1_score  support
accuracy                                     0.765
macro avg  0.709463  0.598870  0.617179  1680.000
weighted avg  0.755971  0.765000  0.738203  1680.000

          macro  micro
f1_score  0.617179  0.765
precision 0.709463  0.765
recall    0.598870  0.765
Overall accuracy = 0.765
```

Figura 3.4: Report finale esperimento con Random Embedding

ding FastText basato su caratteri, l'aumento del volume del dataset produce un miglioramento globale per tutte le famiglie di malware. L'accuracy del modello passa infatti dal 75% al 79%. Confrontando invece questo report, con il precedente in figura 3.4, si nota che il cambio di embedding ancora una volta produce un aumento di prestazioni al livello globale, per tutte le famiglie di malware. In particolare continuano a beneficiare di questo cambio le famiglie Gozi, Necurs, Pykspa, Rovnix e Vawtrak. Al contrario ancora una volta, il cambio di embedding, anche se aumentano le dimensioni del dataset, non migliora le prestazioni del classificatore, nel riconoscere la famiglia Ramnit. Inoltre, come nei casi precedenti, il classificatore continua a scambiare molti nomi di dominio, appartenenti alle famiglie Vawtrak e Nymaim, per nomi di dominio benevoli (si veda la matrice di confusione 5.5 in Appendice). Globalmente, si assiste a un aumento dell'accuracy del modello, a seguito del cambio di embedding, anche se l'aumento è minore, rispetto a quello registrato con il dataset più piccolo, cioè dal 76,5% al 79%.

Capitolo 3 Risultati sperimentali

```
Class report:
precision  recall  f1-score  support
conficker  0.681048  0.493761  0.567692  34.000000
corebot    1.000000  0.994118  0.997015  34.000000
cryptolocker  0.560364  0.616221  0.584219  33.000000
dircrypt   0.532881  0.352941  0.406306  34.000000
emotet     0.928679  0.994118  0.960241  34.000000
fobber     0.544373  0.870232  0.665646  33.000000
gozi       0.739673  0.366845  0.487618  34.000000
kraken     0.726204  0.473975  0.559881  34.000000
matsnu     0.924455  0.613369  0.706328  34.000000
murofet    0.933333  0.750980  0.824017  34.000000
necurs     0.646838  0.290018  0.398039  34.000000
nymaim     0.513434  0.101783  0.166099  33.000000
padcrypt   0.971663  0.904278  0.932829  34.000000
pushdo    0.829018  0.638681  0.709591  34.000000
pykspa     0.806897  0.614795  0.694153  33.000000
qadars     1.000000  0.957754  0.977958  34.000000
ramdo      0.993750  0.976114  0.984799  33.000000
ramnit     0.310914  0.251515  0.270997  33.000000
ranbyus    0.629705  0.712834  0.665903  34.000000
rovnix     0.981818  0.910160  0.943997  33.000000
simda      0.737139  0.751337  0.740273  34.000000
suppobox   0.580201  0.602317  0.588060  34.000000
symmi      0.982353  0.980235  0.985162  33.000000
tinba      0.543953  0.540285  0.528241  33.000000
vawtrak    0.719423  0.254367  0.373444  33.000000
alexa      0.825531  0.945714  0.881421  840.000000
Precision  Recall    F1 score  support
accuracy   0.793095
macro avg  0.755525  0.652567  0.676920  1680.000000
weighted avg  0.789241  0.793095  0.775005  1680.000000

          macro  micro
f1_score  0.676920  0.793095
precision 0.755525  0.793095
recall    0.652567  0.793095

Overall accuracy = 0.7930952380952381
```

Figura 3.5: Report finale esperimento con FastText Embedding basato su caratteri

Analizzando il report in figura 3.6, è possibile osservare che, anche nel caso di embedding FastText basato su bigrammi, l'aumento del volume del dataset produce un aumento globale delle prestazioni (l'accuracy cresce dal 78% all' 82%). L'unica famiglia per cui le prestazioni del classificatore non migliorano all'aumentare del dataset resta Ramnit. Confrontando, invece, questi risultati con quelli ottenuti con embedding FastText basato su caratteri, e stessa taglia del dataset, si osserva ancora una volta a un miglioramento significativo globalmente (l'accuracy cresce dal 79% all' 82%), e per molte famiglie. Tra queste è importante citare Gozi, Necurs, Nymaim, Pushdo, Pykspa, Simda, Suppobox e Vawtrak. Per quest'ultima in particolare, si conferma ancora il miglioramento significativo delle prestazioni, nel passaggio da caratteri a bigrammi, anche con un dataset più grande. Allo stesso modo, si registra, passando da caratteri a bigrammi, un peggioramento significativo in F1-score per la famiglia Fobber, come nel caso precedente. Le famiglie per cui la rete ottiene prestazioni peggiori sono Nymaim, Dircrypt e Ramnit. Guardando la matrice di

3.3 Esperimenti con Dataset da 16800 nomi di dominio

confusione 5.6, in appendice, notiamo che mentre i nomi di dominio Nymaim sono scambiati come benevoli, i nomi di dominio Dirccrypt sono scambiati principalmente come nomi di dominio Ramnit, e i nomi di dominio Ramnit sono principalmente confusi con quelli Fobber.

```
Class report:
precision  recall  f1-score  support
conficker  0.744744  0.565954  0.635056  34.000000
corebot    1.000000  0.970410  0.984707  34.000000
cryptolocker 0.574570  0.592335  0.578773  33.000000
dirccrypt  0.433591  0.382353  0.396602  34.000000
emotet     0.950292  0.988235  0.968523  34.000000
fobber     0.541697  0.631016  0.579698  33.000000
gozi       0.829936  0.503922  0.592376  34.000000
kraken     0.665997  0.580570  0.610300  34.000000
matsnu     0.853181  0.790374  0.818064  34.000000
murofet    0.851292  0.719964  0.775219  34.000000
necurs     0.701923  0.538503  0.608602  34.000000
nymaim     0.606007  0.251872  0.351629  33.000000
padcrypt   0.981605  0.911230  0.944381  34.000000
pushdo    0.930375  0.799287  0.859238  34.000000
pykspa     0.845560  0.783601  0.811252  33.000000
qadars     0.982511  0.945633  0.961995  34.000000
ramdo      0.936880  0.958467  0.947005  33.000000
ramnit     0.357924  0.287166  0.313880  33.000000
ranbyus    0.682465  0.748307  0.707535  34.000000
rovnix     0.942906  0.856150  0.896468  33.000000
simda      0.776047  0.798396  0.783844  34.000000
suppobox   0.680905  0.638681  0.650177  34.000000
symmi      0.971410  0.993939  0.982340  33.000000
tinba      0.530654  0.510517  0.501193  33.000000
vawtrak    0.887220  0.620321  0.725716  33.000000
alex       0.868066  0.947143  0.905596  840.000000
          Precision Recall  F1_score  support
accuracy                               0.820833
macro avg  0.774145  0.704398  0.726545  1680.000000
weighted avg 0.819241  0.820833  0.812448  1680.000000

          macro  micro
f1_score  0.726545  0.820833
precision 0.774145  0.820833
recall    0.704398  0.820833

Overall accuracy = 0.8208333333333333
```

Figura 3.6: Report finale esperimento con FastText Embedding basato su bigrammi

3.3 Esperimenti con Dataset da 16800 nomi di dominio

Il primo report in figura 3.7, evidenzia ancora una volta che un ulteriore aumento del volume del dataset, produce un aumento delle prestazioni globali per tutta la rete. Infatti ancora una volta si rilevano, aumenti del valore di F1-score per molte famiglie come Cryptolocker, Gozi, Kraken, Matsnu, Necurs, Pykspa, Rovnix, Emotet e Vawtrak, che sono le stesse che avevano beneficiato dell'aumento del volume del dataset anche nel caso precedente. Come accadeva anche con le taglie di dataset più piccole, nel caso dell'embedding randomico si registrano pessime prestazioni del classificatore per la famiglia Nymaim, di cui 52 campioni su 67, sono scambiati come

Capitolo 3 Risultati sperimentali

nomi benevoli (si veda matrice di confusione 5.7, in Appendice). Al contrario di quanto accaduto prima, si riscontra un piccolo aumento del valore di F1-score per la famiglia Ramnit. Globalmente, si registra un aumento importante dell'accuracy del modello, passando dal 76.5% all'81.5%. Infatti, si ottengono prestazioni, migliori rispetto al caso con embedding FastText basato su caratteri e dataset da 8400 nomi di dominio, e vicine a quelle ottenute dal classificatore, con embedding FastText basato su bigrammi e dataset da 8400 nomi di dominio.

```
Class report:
precision  recall  f1-score  support
conficker  0.730543  0.507243  0.592298  68.000000
corebot    0.973991  0.997015  0.985314  67.000000
cryptolocker 0.672442  0.581782  0.622470  67.000000
dircrypt   0.650342  0.348683  0.415884  68.000000
emotet     0.967647  0.970588  0.968748  68.000000
fobber     0.686593  0.762818  0.717460  67.000000
gozi       0.743459  0.493942  0.542020  67.000000
kraken     0.819813  0.584284  0.677495  68.000000
matsnu     0.896279  0.730290  0.793616  67.000000
murofet    0.917927  0.721071  0.799332  68.000000
necurs     0.698863  0.438806  0.535568  67.000000
nymaim     0.676354  0.125241  0.203178  67.000000
padcrypt   0.984826  0.914486  0.947853  68.000000
pushdo     0.893400  0.710448  0.786997  67.000000
pykspa     0.819796  0.753731  0.774786  67.000000
qadars     0.977871  0.905092  0.939809  68.000000
ramdo      0.973200  0.943415  0.957644  67.000000
ramnit     0.388573  0.469842  0.405820  67.000000
ranbyus    0.799966  0.781299  0.788848  68.000000
rovnix     0.987292  0.901888  0.942309  67.000000
simda      0.846841  0.719315  0.773545  68.000000
suppobox   0.681371  0.485601  0.545870  68.000000
symmi      0.993884  0.982090  0.987784  67.000000
tinba      0.578535  0.720983  0.631064  67.000000
vawtrak    0.840748  0.330158  0.460163  67.000000
alexa      0.833130  0.954777  0.889667  1685.000000
Precision  Recall  F1_score  support
accuracy   0.815015
macro avg  0.808988  0.685957  0.718675  3370.000000
weighted avg 0.820596  0.815015  0.800783  3370.000000

macro  micro
f1_score 0.718675 0.815015
precision 0.808988 0.815015
recall    0.685957 0.815015

Overall accuracy = 0.8150148367952522
```

Figura 3.7: Report finale esperimento con Random Embedding

Visionando il report in figura 3.8, si nota che anche questa volta l'aumento del volume del dataset, produce un aumento delle prestazioni del classificatore, sia per le metriche relative alle singole famiglie, sia globalmente, in quanto l'accuracy del modello cresce dal 79% all'82%.

Confrontando, invece, i risultati di questo report, con quelli visibili nel precedente in figura 3.7, si rileva ancora una volta un miglioramento importante, nel valore di F1-score, per le famiglie Gozi, Suppobox e Vawtrak, nel passaggio da embedding

3.3 Esperimenti con Dataset da 16800 nomi di dominio

```

Class report:
precision    recall  f1-score   support
conficker    0.761254  0.539991  0.625305   68.000000
corebot      0.997059  0.991045  0.993984   67.000000
cryptolocker 0.639743  0.533670  0.532506   67.000000
dircrypt     0.526506  0.448551  0.434113   68.000000
emotet       0.965983  0.991133  0.978267   68.000000
fobber       0.656559  0.636304  0.635092   67.000000
gozi         0.819062  0.544688  0.643917   67.000000
kraken       0.748436  0.678402  0.707794   68.000000
matsnu       0.872408  0.804917  0.834949   67.000000
murofet      0.798633  0.800790  0.782908   68.000000
necurs       0.630921  0.420896  0.492754   67.000000
nymaim       0.692211  0.184767  0.285046   67.000000
padcrypt     0.991209  0.958648  0.974383   68.000000
pushdo       0.920901  0.758209  0.829491   67.000000
pykspa       0.866088  0.747849  0.799794   67.000000
qadars       0.996923  0.916769  0.954870   68.000000
ramdo        0.987920  0.973354  0.980473   67.000000
ramnit       0.390069  0.294557  0.299171   67.000000
ranbyus      0.854601  0.710843  0.760428   68.000000
rovnix       0.982073  0.925856  0.951898   67.000000
simda        0.848379  0.709701  0.766146   68.000000
suppobox     0.763313  0.624978  0.667180   68.000000
symmi        0.991218  0.997015  0.994074   67.000000
tinba        0.515116  0.559394  0.533817   67.000000
vawtrak      0.970569  0.419579  0.578077   67.000000
alexa        0.849749  0.966528  0.904242  1685.000000
Precision    Recall    F1_score   support
accuracy                                           0.826766
macro avg    0.809112  0.697632  0.728488  3370.000000
weighted avg 0.828633  0.826766  0.812903  3370.000000

          macro    micro
f1_score  0.728488  0.826766
precision 0.809112  0.826766
recall    0.697632  0.826766

Overall accuracy = 0.8267655786350149

```

Figura 3.8: Report finale esperimento con FastText Embedding basato su caratteri

randomico a embedding FastText basato su caratteri. Per le altre famiglie le prestazioni crescono di poco oppure restano quasi inalterate. Fanno eccezione, in questo caso particolare, solo le famiglie Cryptolocker, Tinba, Ramnit e Fobber, per le quali c'è un peggioramento non trascurabile delle prestazioni. Mentre per le prime due non è mai stato riscontrato un comportamento del genere, e quindi il peggioramento potrebbe essere un caso, invece le altre due non sono nuvole a questi comportamenti: Fobber aveva già dimostrato di reagire male ai cambi di embedding, nel passaggio da FastText basato su caratteri a FastText basato su bigrammi; per Ramnit allo stesso modo si continuano a vedere delle oscillazioni nel valore di F1-score, ogni volta che viene aumentato o diminuito il volume del dataset, oppure quando c'è un cambio di embedding. Globalmente si nota ancora una volta che il cambio dell'embedding, produce un miglioramento nell'accuracy complessiva del modello. Però viene confermata anche, la tendenza da parte di questo miglioramento a ridursi, con l'aumento della dimensione del dataset, in quanto l'accuracy cresce solo dell' 1.1%, mentre con

Capitolo 3 Risultati sperimentali

il dataset da 8400 nomi di dominio era cresciuta del 3.5%.

L'ultimo report, relativo alla figura 3.9, testimonia ancora, che l'aumento del volume del dataset anche con embedding FastText basato su bigrammi, produce un aumento del valore di F1-score per molte famiglie di malware, escluse quelle che lo avevano già alto. Ancora una volta si nota che la famiglia Ramnit, risulta essere un'eccezione, in quanto il suo valore di F1-score non aumenta, con l'aumento del volume del dataset. Al livello globale, l'accuracy del modello cresce passando dall'82% all'84.5%.

```
Class report:
precision  recall  f1-score  support
conficker  0.753315  0.581782  0.649670  68.000000
corebot    0.991089  0.973134  0.981670  67.000000
cryptolocker  0.677373  0.650044  0.656069  67.000000
dircrypt   0.434184  0.298727  0.349601  68.000000
emotet     0.974364  0.979324  0.976526  68.000000
fobber     0.599390  0.580114  0.575058  67.000000
gozi       0.763379  0.554434  0.616917  67.000000
kraken     0.851288  0.634021  0.725762  68.000000
matsnu     0.803008  0.751668  0.775977  67.000000
murofet    0.789738  0.753687  0.760478  68.000000
necurs     0.652361  0.573134  0.607313  67.000000
nymaim     0.760930  0.452371  0.563128  67.000000
padcrypt   0.964892  0.961589  0.963085  68.000000
pushdo     0.938921  0.865672  0.900085  67.000000
pykspa     0.874158  0.875461  0.874154  67.000000
qadars     0.993939  0.955487  0.974224  68.000000
ramdo      0.972947  0.961326  0.967070  67.000000
ramnit     0.315087  0.470237  0.376464  67.000000
ranbyus    0.805665  0.751668  0.777569  68.000000
rovnix     0.971094  0.877963  0.921847  67.000000
simda      0.902536  0.858033  0.877091  68.000000
suppobox   0.829054  0.703863  0.759751  68.000000
symmi      0.979832  0.988104  0.983855  67.000000
tinba      0.614176  0.535953  0.568075  67.000000
vawtrak    0.886676  0.753117  0.811058  67.000000
alexas     0.890682  0.957982  0.922977  1685.000000
Precision  Recall  F1_score  support
accuracy   0.845757
macro avg  0.807311  0.742265  0.765980  3370.000000
weighted avg  0.847347  0.845757  0.841314  3370.000000

          macro  micro
f1_score  0.765980  0.845757
precision 0.807311  0.845757
recall    0.742265  0.845757

Overall accuracy = 0.8457566765578635
```

Figura 3.9: Report finale esperimento con FastText Embedding basato su bigrammi

Valutando invece, la variazione delle prestazioni del classificatore, ottenuta con il cambio di embedding, si rileva che c'è un miglioramento importante, in F1-score, per alcune famiglie di malware come Cryptolocker, Necurs, Nymaim, Pushdo, Pykspa, Simda, Suppobox e Vawtrak. Degni di nota sono i miglioramenti ottenuti per le famiglie Necurs, Nymaim e Vawtrak, nel passaggio da caratteri a bigrammi, per cui si registrano degli aumenti in F1-score tali che, per tutte e tre si ha un valore di

F1-score discreto e superiore a 0.56. Per le altre famiglie le prestazioni migliorano di poco o rimangono stabili. Le uniche eccezioni a questo comportamento, sono le famiglie Dircrypt e Fobber, per cui si registra un degrado delle prestazioni: mentre per Dircrypt è la prima volta, che si riscontra questo comportamento da parte del classificatore, per Fobber invece, si rileva un fenomeno già analizzato anche nei casi precedenti. Importante, è anche sottolineare che le famiglie per cui la rete ottiene le prestazioni peggiori sono Dircrypt e Ramnit, a causa del fatto che, molti campioni appartenenti alla famiglia Dircrypt sono scambiati come campioni della famiglia Ramnit, e molti campioni appartenenti alla famiglia Ramnit sono confusi con campioni Fobber (si veda la matrice di confusione 5.9 in appendice). Concludendo, si nota ancora una volta un aumento dell'accuracy complessiva del modello, nel passaggio da embedding basato su caratteri a embedding basato su bigrammi. Purtroppo è confermata anche la tendenza di questo aumento a ridursi, con la crescita della grandezza del dataset, in quanto, mentre prima il miglioramento è stato del 3%, qui è stato solo del 2%.

3.4 **Discussione**

Effettuata l'analisi dei nove report, presentati nelle precedenti sezioni, si è arrivati a trarre una serie di conclusioni dagli esperimenti, di cui alcune erano prevedibili, anche alla luce degli esperimenti realizzati precedentemente a questo lavoro, e invece altre sono state un elemento di novità molto particolare. Il primo risultato importante della serie di esperimenti condotti, è che:

Uno strato di embedding basato su FastText, sia basato su caratteri, sia basato su bigrammi, migliora le prestazioni del classificatore, rispetto al caso in cui venga usato uno strato di embedding randomico.

Questo primo risultato rilevato è testimoniato dall'andamento dell'accuracy complessiva del modello, al variare delle dimensioni del dataset e dello strato di embedding, presentato in figura 3.10

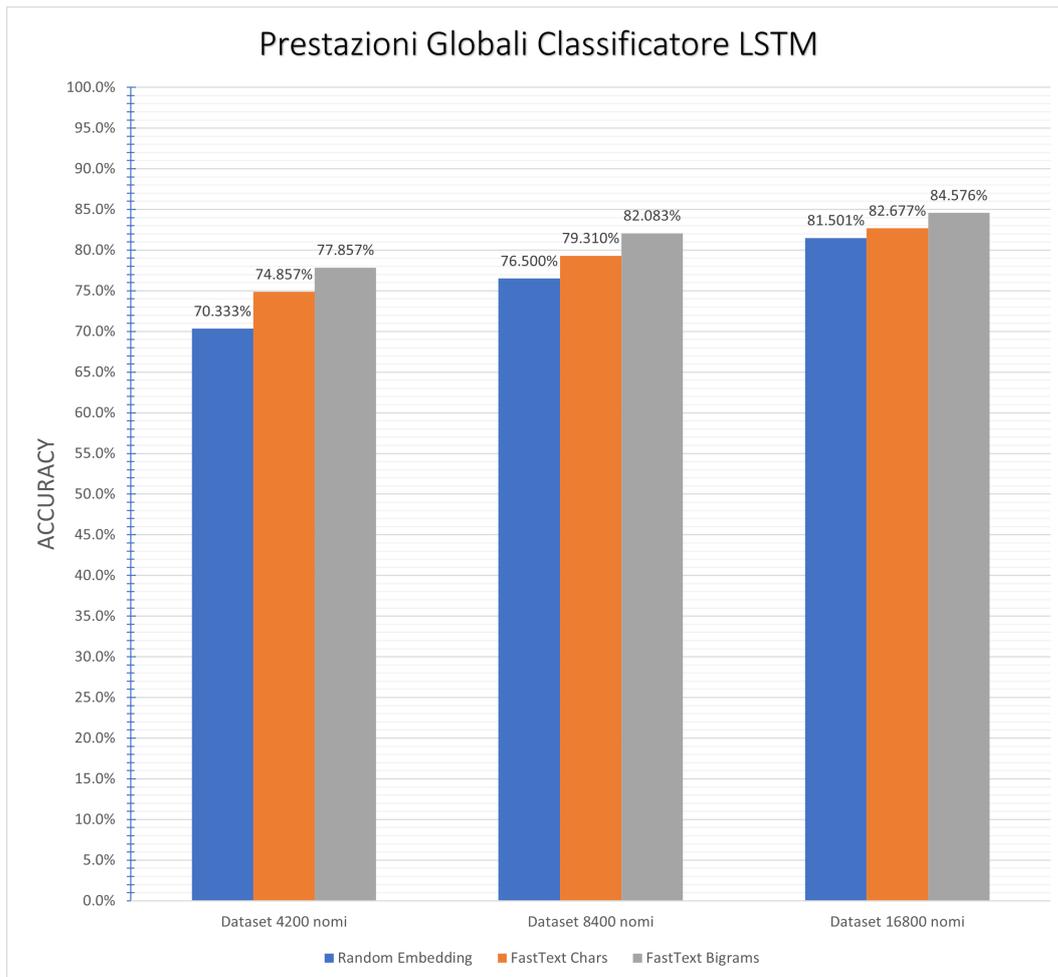


Figura 3.10: Andamento dell'accuracy complessiva al variare della dimensione del dataset e dello strato di embedding

L'istogramma dimostra che, per ogni taglia di dataset utilizzata, indicata sull'asse delle ascisse, l'accuracy del modello, ottenuta con uno strato di embedding randomico, è sempre minore dell'accuracy ottenuta nel caso di embedding FastText basato su caratteri o basato su bigrammi. Inoltre, il grafico mette in luce anche un altro aspetto fondamentale, e non scontato, cioè che tra gli embedding FastText testati, in ogni caso si ottengono sempre prestazioni migliori, con lo strato basato sui bigrammi. Oltre a questo risultato che è quello principale che si voleva ottenere con questo lavoro, possiamo proseguire enunciandone altri. Il secondo risultato, quindi, che si può ricavare dagli esperimenti è il seguente:

L'aumento delle dimensioni del dataset si traduce in un miglioramento delle prestazioni del classificatore, qualunque sia lo strato di embedding, sia relativamente alle singole famiglie, sia globalmente.

Questo è testimoniato dall'andamento dell'accuracy complessiva del modello, al variare delle dimensioni del dataset, sempre visibile nell'istogramma in figura 3.10. Infatti, confrontando le colonne dello stesso colore nel grafico, che fanno riferimento a uno stesso tipo di embedding, si riscontra un aumento dell'accuracy significativo, ad ogni raddoppio della dimensione del dataset. Tale fenomeno risulta ancora più chiaro, se lo si analizza attraverso il grafico visibile in figura 3.11, che, oltre a presentare i valori di accuracy per ogni tipo di embedding, al variare del dataset, li interpola per ricercare una tendenza.

Ognuno dei punti evidenziati nel grafico, rappresenta una coppia, formata dalla taglia di dataset utilizzata, espressa in nomi di dominio, e dall'accuracy ottenuta dal modello fissando lo strato di embedding. Il grafico non solo corrobora ulteriormente, la tesi che le prestazioni aumentano con la crescita del volume del dataset, ma mette in luce anche un'altra cosa importante: guardando le rette a tratto continuo, ottenute mediante interpolazione lineare dei punti, si può notare che esse sembrano tendere, indipendentemente dallo strato di embedding, verso lo stesso punto. Ciò significa che se fossero stati condotti altri esperimenti con dataset più grandi, probabilmente

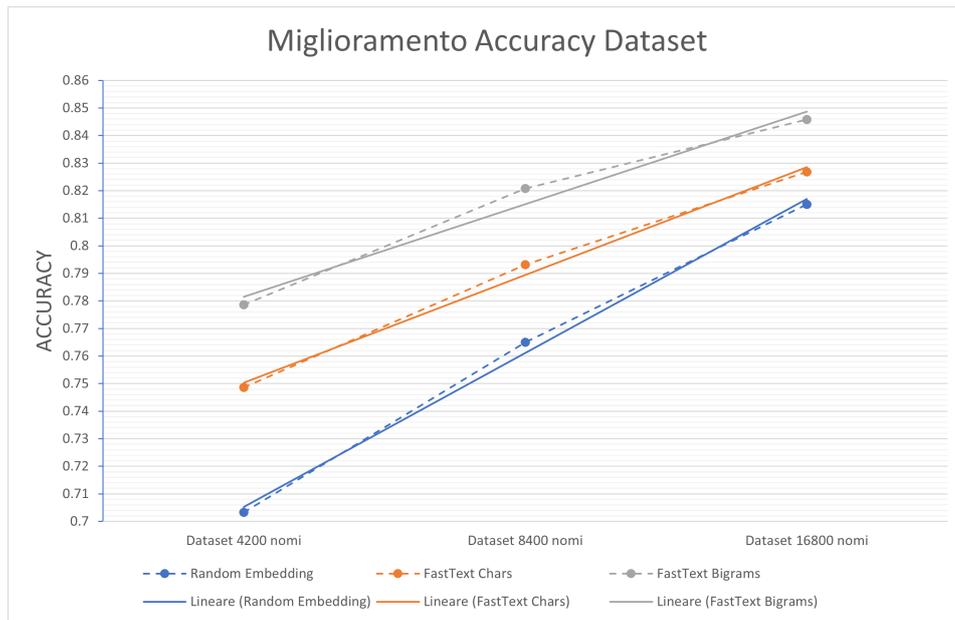


Figura 3.11: Analisi del miglioramento dell'accuracy complessiva al variare delle dimensioni del dataset

saremo riusciti ad individuare un punto in cui, anche se lo strato di embedding cambia, le prestazioni del classificatore cambiano poco o quasi per nulla.

Quest'ultima riflessione ci permette di introdurre, un'altra conclusione ricavata dagli esperimenti:

Il miglioramento delle prestazioni, prodotto dall'uso di uno strato di embedding basato su FastText, decresce con l'aumento della dimensione del dataset.

Questo è certamente visibile anche dall'istogramma in figura 3.10, in cui si nota che le colonne relative alle prestazioni ottenute con i vari tipi di embedding, si appaiano se il dataset cresce. In realtà, il modo migliore di verificare quanto appena detto è analizzare la figura 3.12. In essa, ognuno dei punti rappresenta una coppia, formata dallo strato di embedding utilizzato e dall'accuracy ottenuta dal modello, con una certa taglia di dataset. Quindi la pendenza delle rette, che si ottengono mediante interpolazione lineare dei punti del grafico, per ogni dimensione del dataset, rappresenta l'aumento di accuracy prodotto dal cambio di embedding. Il grafico mostra che la pendenza delle rette, diminuisce ogni volta che si raddoppia

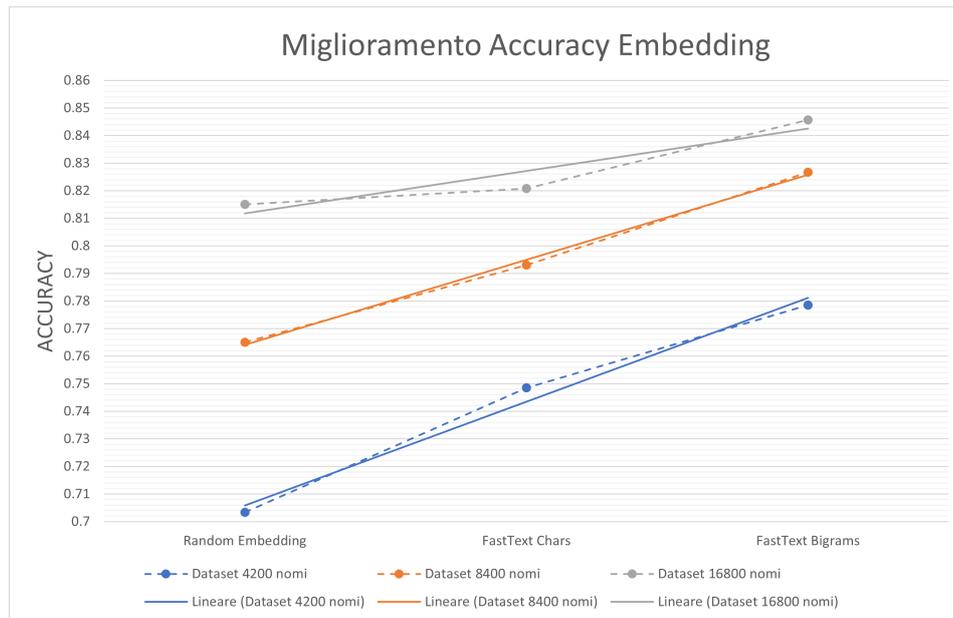


Figura 3.12: Analisi del miglioramento dell'accuracy complessiva al variare dello strato di embedding

la dimensione del dataset, corroborando la tesi enunciata. Probabilmente se fossero stati condotti altri esperimenti, con taglie di dataset più grandi, si sarebbe osservato che aumentando sempre di più il volume del dataset, la pendenza delle rette sarebbe diventata quasi nulla, ottenendo così aumenti minimi di accuracy.

Terminata una discussione più globale degli esperimenti, è interessante vedere come variano le prestazioni del classificatore per le singole famiglie di malware, cambiando strato di embedding e dimensione del dataset. Dall'analisi degli esperimenti si è dedotto che:

Non sempre l'uso di uno strato di embedding basato su FastText, o l'aumento delle dimensioni del dataset, produce un miglioramento nelle prestazioni del classificatore per una certa famiglia di malware.

Valutando i report, ci si è accorti che molte per molte famiglie le prestazioni del classificatore migliorano, a seguito di un cambio di embedding, o di un aumento delle dimensioni del dataset, o di tutti e due. Un esempio di questo è fornito dalla famiglia Vawtrak, per cui l'F1-score ottenuto dal classificatore aumenta sensibilmente, quando

viene usato uno strato di embedding FastText basato su bigrammi, come visibile in figura 3.13

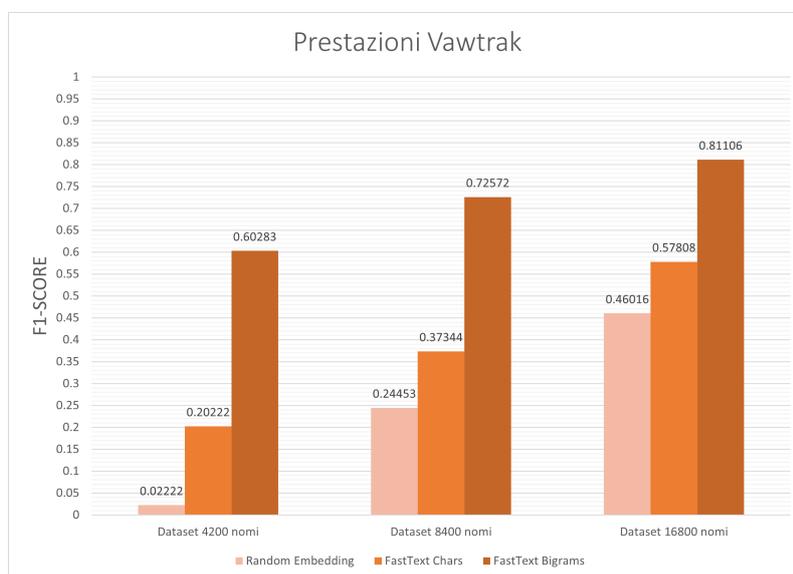


Figura 3.13: Valori di F1-score ottenuti per la famiglia Vawtrak

Compatibilmente con quanto raffigurato nel grafico, si nota che, nel passaggio da embedding randomico a embedding FastText basato su caratteri, l'aumento di F1-score rilevato, è più piccolo di quello che si registra, passando da embedding FastText basato su caratteri, a quello basato sui bigrammi. Infatti, i punteggi di F1-score, ottenuti dal classificatore, per la famiglia Vawtrak, usando un embedding basato su bigrammi, sono nettamente superiori a quelli che si ottengono con gli altri embeddings, per tutte e tre le taglie di dataset.

Caso opposto a quello presentato, è quello della famiglia Fobber, per cui invece, si registra un degrado importante delle prestazioni del classificatore, quando si usa un embedding FastText basato su bigrammi. La figura 3.14 evidenzia che, qualsiasi sia la taglia di dataset sperimentata, l'F1-score ottenuto dal classificatore per la famiglia Fobber, nel caso di embedding basato su bigrammi, è minore rispetto a quello ottenuto negli altri due casi. Certamente, si nota anche che le prestazioni del classificatore per questa famiglia, beneficiano dell'aumento di volume del dataset, specialmente se lo strato di embedding usato è quello randomico.

Singolare è il caso della famiglia Ramnit, per cui, non solo il classificatore non

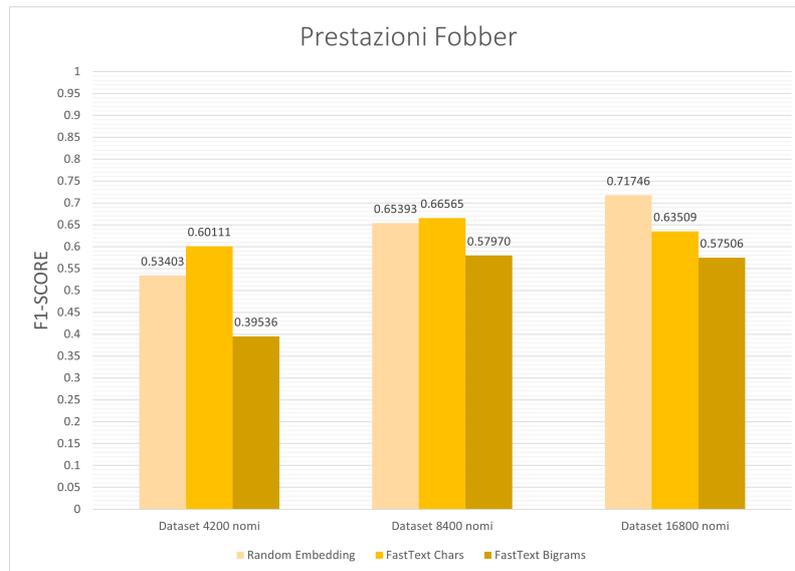


Figura 3.14: Valori di F1-score ottenuti per la famiglia Fobber

riesce a ottenere prestazioni discrete e accettabili, ma non si riesce ad individuare un comportamento univoco, a seguito dell'aumento o diminuzione del dataset, o del cambio di embedding. L'F1-score ottenuto dal classificatore, per questa famiglia, non risulta avere un'andamento ben definito come dimostrato dalla figura 3.15. Ad

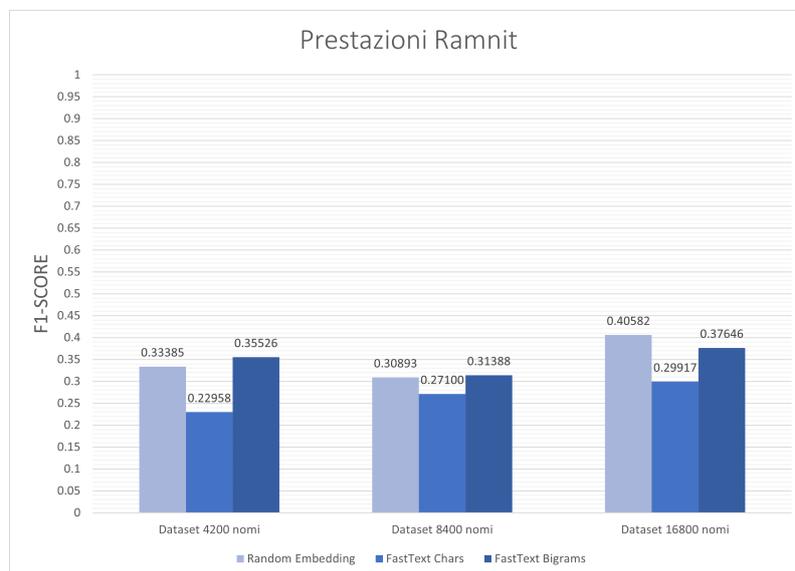


Figura 3.15: Valori di F1-score ottenuti per la famiglia Ramnit

esempio l'F1-score nel caso dell'embedding FastText basato su caratteri, diminuisce passando dal dataset più piccolo al dataset intermedio, però aumenta passando dal

dataset intermedio a quello più grande.

Ultima famiglia particolarmente degna di nota, è la famiglia Nymaim, per cui il classificatore, finché il dataset non ha un volume sufficiente, non riesce a ottenere un valore di F1-score discreto, compatibilmente con quanto riportato in figura 3.16 Singolare è sicuramente il valore 0 di F1-score ottenuto con il dataset più piccolo ed

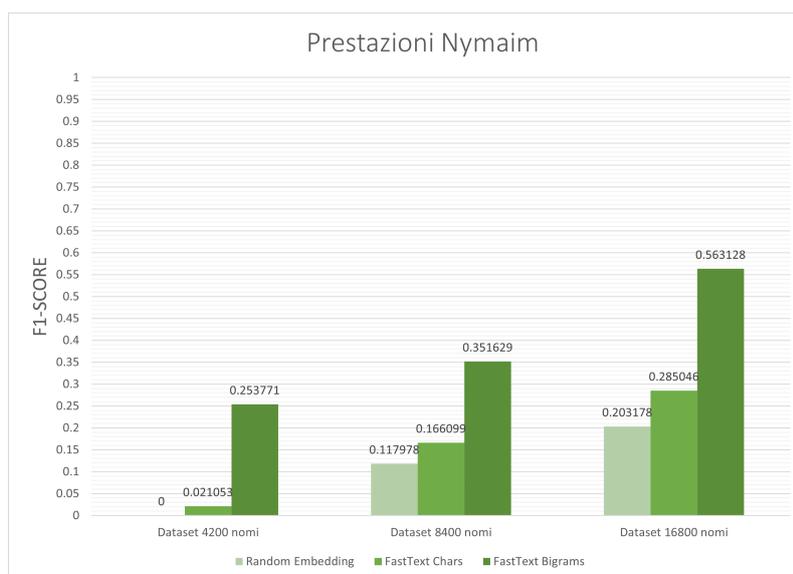


Figura 3.16: Valori di F1-score ottenuti per la famiglia Nymaim

embedding randomico.

I valori di F1-score così bassi, ottenuti dal classificatore, per la famiglia Nymaim, hanno determinato che venisse realizzata un'indagine più approfondita, riguardo questa famiglia, usando le matrici di confusione. Da questa ulteriore analisi si è derivata l'ultima conclusione ottenuta:

I nomi di dominio della famiglia Nymaim, indipendentemente dalla dimensione del dataset e dallo strato di embedding, sono quelli che più facilmente vengono scambiati come nomi di dominio benevoli.

Tale affermazione è dimostrata, oltre che dalle matrici di confusione presenti in appendice, anche dalla figura 3.17 Dall'analisi del grafico, si nota subito che la percentuale di nomi di dominio Nymaim, confusi come nomi benevoli è sempre alta e

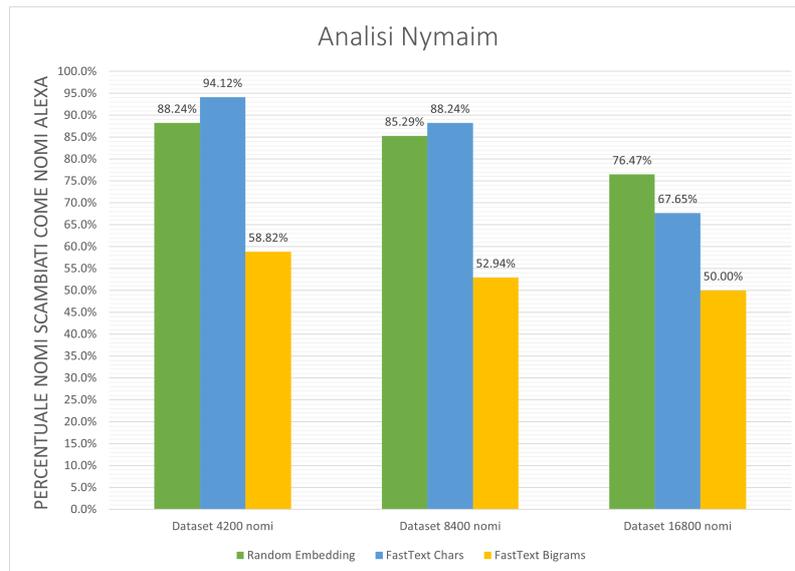


Figura 3.17: Analisi percentuale di nomi Nymaim reputati come benevoli

sopra il 50%. Comunque, si nota anche che un modo per limitare questa confusione, è quello di usare un embedding FastText basato su bigrammi, grazie al quale la percentuale di confusione cala drasticamente, se confrontata con gli altri due casi. Oltre a questo anche l'aumento della dimensione del dataset, alla luce anche della figura 3.16, certamente influisce positivamente, e diminuisce questa percentuale.

Capitolo 4

Conclusioni e Sviluppi Futuri

A seguito dello studio e degli esperimenti, svolti in questo lavoro, possiamo certamente ritenerci soddisfatti. Lo scopo presentato all'inizio, cioè dimostrare che uno strato di embedding basato su FastText addestrato con dati reali, migliora le prestazioni del classificatore LSTM.MI, è stato raggiunto. In tutti gli esperimenti svolti con lo strato di embedding FastText, si sono riscontrate prestazioni migliori del classificatore rispetto al caso dell'embedding randomico, sia globalmente, sia puntualmente per le varie famiglie di malware. In particolare, con embedding FastText, il classificatore migliora le prestazioni per quelle famiglie, per cui nel caso dell'embedding randomico si avevano seri problemi.

La peculiarità interessante però di questa conclusione, sta sicuramente nel fatto che, questo miglioramento, generato dal cambio di embedding, non è stato riscontrato solo con un dataset, ma con più dataset di grandezze diverse. In più, si è osservato che il volume del dataset influisce sul miglioramento, generato dal cambio di embedding, delle prestazioni globali del classificatore: più il volume del dataset cresce e più il miglioramento diminuisce. Tale comportamento, anche se può sembrare negativo, in realtà, dà la possibilità di addestrare il classificatore con dataset più piccoli, impiegando meno tempo e ottenendo comunque ottime prestazioni. Certamente, questa soluzione risulta molto utile, nel momento in cui si sottopone il classificatore a cicli successivi di addestramento per mantenerlo aggiornato, specialmente nel caso in cui cambino gli algoritmi di generazione dei domini, per una o più famiglie di

malware.

Altra conclusione importante relativa agli esperimenti, infine, è che esiste una stretta minoranza di famiglie di malware, per cui le prestazioni del classificatore non migliorano a seguito del cambio di embedding. Questo comportamento particolare, rappresenta uno dei pochi lati negativi che si ha quando si cambia embedding, oltre ai costi di addestramento di FastText, che sono comunque importanti. Il peggioramento riscontrato non è tale da spingerci ad abbandonare questa soluzione, ma va tuttavia segnalato. Infatti, se in una rete di computer prevalgono le infezioni, generate proprio da quelle famiglie di malware, per cui con FastText si ottengono prestazioni peggiori, allora sarebbe bene, in quel caso, studiare un'altra tecnica di embedding.

Nonostante le conclusioni raggiunte siano molto interessanti, e costituiscano un successo importante, comunque sono parziali. Innanzitutto non è stato possibile, a causa della mancanza di mezzi e di tempo, addestrare uno strato di embedding FastText basato su trigrammi. In più le taglie di dataset sperimentate sono relativamente piccole, e quindi le conclusioni raggiunte, sono frutto di previsioni, comunque abbastanza consistenti, ma basate su delle tendenze individuate nei vari esperimenti. Infine, i dataset utilizzati sono bilanciati e costruiti sinteticamente, data l'impossibilità di costruire un dataset, con dati provenienti dal traffico reale.

Tutta questa serie di impedimenti, rendono certamente il lavoro svolto migliorabile, e aprono, a una serie di sviluppi e lavori futuri molto interessanti. La prima proposta, è quella di sperimentare LSTM.MI, utilizzando uno strato di embedding FastText basato su trigrammi. Infatti, dato il miglioramento significativo, che si ottiene passando da embedding basato su caratteri, a embedding basato su bigrammi, si potrebbe pensare che l'uso dei trigrammi, possa migliorare ulteriormente le prestazioni del classificatore. Magari, per le famiglie per cui si ottenevano pessime prestazioni con i bigrammi, si potrebbero ottenere risultati migliori con i trigrammi. Allo stesso modo però è da considerare anche l'ipotesi inversa, cioè che, l'uso di uno strato di embedding basato su trigrammi, possa peggiorare le prestazioni per altre famiglie, come si era già visto nel caso dei bigrammi. Perciò testare anche questo strato di embedding potrebbe essere interessante.

Altra proposta da considerare in futuro, è quella di sperimentare LSTM.MI, con i vari strati di embedding e con dataset anche più grandi. A causa della mancanza di tempo per fare gli esperimenti in questo lavoro, il dataset più grande che è stato sperimentato aveva solo 16800 nomi di dominio. Quindi sarebbe interessante, testare la rete neurale con dataset più voluminosi, sia per verificare ulteriormente, che l'aumento del volume del dataset riduce l'aumento delle prestazioni, provocato dall'uso di embedding basato su FastText, sia per stabilire una "soglia" per le dimensioni del dataset, oltre la quale addestrare uno strato di embedding basato su FastText diventa troppo costoso. In più, una terza proposta da sperimentare in futuro, potrebbe essere quella di utilizzare un dataset, costruito con dati provenienti dal traffico reale. In questa maniera, si potrebbero determinare i vantaggi, derivanti dall'uso uno strato di embedding basato su FastText, in un caso più vicino alla realtà, con un dataset sbilanciato, in cui sono presenti solo alcune famiglie di malware.

Capitolo 5

Appendice

Nel Capitolo 3 si è dedicato molto spazio alla discussione dei risultati sperimentali, sia mediante la presentazione dei report specifici per gli esperimenti, sia mediante l'uso di grafici riassuntivi che supportassero le conclusioni raggiunte. Un altro strumento molto utile, presentato nella sezione 1.3.2 del Capitolo 1, per l'analisi degli esperimenti è la **matrice di confusione**. Essa nel caso di molti esperimenti è stata utile, per comprendere il motivo delle basse prestazioni ottenute dal classificatore per famiglie come Nymaim, Ramnit e Dirccrypt.

Si riportano quindi, in questa sezione, nove matrici di confusione, ognuna relativa a uno specifico fold di un esperimento, per motivi di spazio e fruizione dei contenuti proposti in questo lavoro di tesi.

	conficker	corebot	cryptolocker	dircrypt	emotet	fofbot	gozi	kraken	matnu	mirrofit	neurus	nyamal	padcrypt	pushido	pykspa	qadars	rand0	ramnt	ranbyus	rovntx	sinda	suppobox	symt	tlmba	vevtrak	alaxa
conficker	4	0	0	0	0	2	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	7	
corebot	0	16	0	0	0	4	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	
cryptolocker	0	0	4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
dircrypt	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
emotet	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
fofbot	0	0	0	0	3	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
gozi	0	0	0	0	0	0	4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	
kraken	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3	
matnu	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	
mirrofit	1	0	1	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
neurus	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	17	
nyamal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
padcrypt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
pushido	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
pykspa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
rand0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ramnt	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ranbyus	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
rovntx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
sinda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
suppobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
symt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
tlmba	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
vevtrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
alaxa	3	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	

Figura 5.1: Matrice di Confusione ottenuta con Random Embedding e Dataset da 4200 nomi di dominio

	conficker	corebot	cryptolocker	dircrypt	enotet	fabber	gozi	kraken	malnu	mutofet	neurus	nyman	padcrpft	pushdo	pyksna	qadars	ramdo	ramnit	ranbyus	ronix	simda	suppobox	symmi	timba	vevtrak	alexa
conficker	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10
corebot	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
cryptolocker	0	0	8	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
dircrypt	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
enotet	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
fabber	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
gozi	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
kraken	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
malnu	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
mutofet	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
neurus	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
nyman	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3
padcrpft	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	3
pushdo	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	3
pyksna	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	3
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	3
ramdo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	3
ramnit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	3
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	3
ronix	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	3
simda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
suppobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
symmi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
timba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
vevtrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
alexa	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	486

Figura 5.2: Matrice di Confusione ottenuta con embedding FastText basato su caratteri e Dataset da 4200 nomi di dominio

	conficker	corobot	cryptolocker	dircrypt	emotet	fohuf	gol	kraken	matanu	murfet	necur	nyman	padcrypt	pustido	pykspa	qadars	rando	ramit	ranbyus	ronlx	smda	suppobox	symbl	tlmba	vawtrak	alex
conficker	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
corobot	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
cryptolocker	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
dircrypt	0	0	0	3	10	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
emotet	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
fohuf	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
gol	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
kraken	2	0	0	1	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
matanu	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
murfet	1	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
necur	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
nyman	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
padcrypt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
pustido	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
pykspa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
rando	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ramit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ronlx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
smda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
suppobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
symbl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
tlmba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
vawtrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
alex	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figura 5.3: Matrice di Confusione ottenuta con embedding FastText basato su bigrammi e Dataset da 4200 nomi di dominio

	conficker	coreana	cryptolocker	dircrypt	emotet	fohber	gozi	kraken	matnu	mutoret	recurt	nymania	padcrypt	pushho	pykspa	qadars	ramlo	ramlit	ranbyus	ronnx	stinda	suppobox	symi	tinba	vettrak	alexa
conficker	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
coreana	0	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
cryptolocker	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
dircrypt	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
emotet	0	0	0	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
fohber	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
gozi	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
kraken	3	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
matnu	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
mutoret	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
recurt	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
nymania	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	
padcrypt	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	
pushho	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	
pykspa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	
ramlo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0	
ramlit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ronnx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
stinda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
suppobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
symi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
tinba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
vettrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
alexa	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figura 5.4: Matrice di Confusione ottenuta con Random Embedding e Dataset da 8400 nomi di dominio

	conflicter	corebot	cryptolocker	dircrypt	emotet	fofber	go21	kraken	matemu	mufofet	necur	nyman	padcrypt	pushho	pykspa	qadars	rando	ramit	randyus	ronux	sinda	supobox	sym1	tliba	vavtrak	alexa
conflicter	16	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	
corebot	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	
cryptolocker	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	1	2	1	0	0	0	1	
dircrypt	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	2	
emotet	0	0	0	0	54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
fofber	0	0	0	0	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	
go21	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	
kraken	2	0	0	2	0	0	0	18	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	
matemu	0	0	0	0	0	0	2	0	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
mufofet	0	0	0	4	0	0	0	0	0	25	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
necur	3	0	0	0	0	0	0	1	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
nyman	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	39	
padcrypt	0	0	0	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0	
pushho	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	
pykspa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	2	
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0	0	
rando	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34	0	0	0	0	0	0	0	0	
ramit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	4	
randyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	0	0	0	
ronux	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	
sinda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0	0	0	0	
supobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	16	
sym1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33	0	0	
tliba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	
vavtrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	
alexa	2	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	
																									783	

Figura 5.5: Matrice di Confusione ottenuta con embedding FastText basato su caratteri e Dataset da 8400 nomi di dominio

	conficker	corebot	cryptolocker	dircrpnt	emotet	flbber	gozi	kraken	matnu	murfet	recur	nymlab	padcrpt	pushdo	pkyspa	qadars	ramdo	ramnt	ranbyus	rovnix	sinda	supobox	symbl	tinba	votrak	alex7
conficker	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
corebot	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
cryptolocker	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
dircrpnt	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
emotet	0	0	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
flbber	0	0	0	2	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
gozi	0	0	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
kraken	1	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
matnu	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
murfet	2	0	0	6	1	0	0	2	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
recur	0	0	0	0	0	0	0	0	0	1	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
nymlab	0	0	0	0	0	1	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	18	
padcrpt	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	1	
pushdo	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	1	
pkyspa	1	0	0	0	0	0	0	1	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	6	
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0	0	
ramdo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	0	0	0	0	0	4	
ramnt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	4	
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	26	0	0	0	0	1	
rovnix	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	1	
sinda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	
supobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0	0	
symbl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34	0	
tinba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	
votrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	
alex7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	
																									738	

Figura 5.6: Matrice di Confusione ottenuta con embedding FastText basato su bigrammi e Dataset da 8400 nomi di dominio

	conflicter	conbot	cryptoocker	dicrypt	emiet	fabber	gozl	kraken	masnu	mutoret	recur	nyralm	patcrypt	pushno	pykska	qadars	rande	ramit	ranbyus	rovnlx	sinda	suppobox	symt	linba	vawtrak	aloxa
conflicter	29	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28
conbot	0	68	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cryptoocker	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dicrypt	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
emiet	0	0	0	0	67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fabber	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
gozl	0	0	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
kraken	1	0	0	1	0	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11
masnu	0	0	0	0	0	0	0	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mutoret	2	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
recur	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nyralm	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
patcrypt	0	0	0	0	0	0	0	0	0	0	0	0	69	0	0	0	0	0	0	0	0	0	0	0	0	0
pushno	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0
pykska	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54	0	0	0	0	0	0	0	0	0	0	0
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	63	0	0	0	0	0	0	0	0	0	0
rande	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	0	0	0	0	0	0	0	0	0
ramit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0
rovnlx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sinda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
suppobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
symt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
linba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
vawtrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aloxa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 5.7: Matrice di Confusione ottenuta con Random Embedding e Dataset da 16800 nomi di dominio

	conficker	corebot	cryptolocker	dircrypt	emotet	foober	gozi	kraken	matnu	murfet	recur	nyama	padcrypt	pushho	pykspa	qudars	ramdo	ramlit	ranbyus	ronyx	simda	suppobox	symi	linba	vantrak	alex	
conficker	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	
corebot	0	65	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
cryptolocker	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
dircrypt	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
emotet	0	0	0	0	68	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
foober	0	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
gozi	0	0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
kraken	1	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
matnu	0	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
murfet	4	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
recur	0	0	0	0	0	0	0	0	0	0	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
nyama	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
padcrypt	0	0	0	0	0	0	0	0	0	0	0	0	64	0	0	0	0	0	0	0	0	0	0	0	0	0	1
pushho	0	0	0	0	0	0	0	0	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	1
pykspa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52	0	0	0	0	0	0	0	0	0	0	0	1
qudars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	1
ramdo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	66	0	0	0	0	0	0	0	0	0	1
ramlit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	1
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	1
ronyx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	0	0	0	0	0	0	1
simda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0	0	0	0	0	1
suppobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0	0	0	0	1
symi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	0	0	0	1
linba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
vantrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
alex	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	167

Figura 5.8: Matrice di Confusione ottenuta con embedding FastText basato su caratteri e Dataset da 16800 nomi di dominio

	conficker	corobot	cryptolocker	dircrypt	emotet	fabber	gozi	kraken	matnu	murfet	neurus	nyman	padcrypt	pushdo	pykspa	qadars	ramdo	ramnit	ranbyus	romix	simda	supobox	symtl	timba	vattrak	alexz	
conficker	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	
corobot	0	66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cryptolocker	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dircrypt	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
emotet	0	0	0	0	67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fabber	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
gozi	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
kraken	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
matnu	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
murfet	0	0	0	0	2	0	0	0	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neurus	4	0	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nyman	0	0	0	0	0	0	0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nyman	0	0	0	0	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0
padcrypt	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0
pushdo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0
pykspa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	66	0	0	0	0	0	0	0	0	0	0	0
qadars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	0	0	0	0	0	0	0	0	0	0
ramdo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ramnit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ranbyus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
romix	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
simda	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
supobox	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
symtl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
timba	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
vattrak	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
alexz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 5.9: Matrice di Confusione ottenuta con embedding FastText basato su bigrammi e Dataset da 16800 nomi di dominio

Bibliografia

- [1] Wikimedia Commons. File:artificialneuronmodel.png — wikimedia commons, the free media repository, 2020. [Online; accessed 13-September-2021].
- [2] Wikimedia Commons. File:multilayerneuralnetworkbigger english.png — wikimedia commons, the free media repository, 2020. [Online; accessed 13-September-2021].
- [3] Wikimedia Commons. File:recurrent neural network unfold.svg — wikimedia commons, the free media repository, 2020. [Online; accessed 13-September-2021].
- [4] Costantino Cerioni. Un sistema a classificatori multipli basato su pre-trained embeddings per il riconoscimento di DGA. Master's thesis, Università Politecnica delle Marche, Italy, 2020.
- [5] David Caprari. Rilevazione di malware da traffico DNS passivo con una rete neurale LSTM. Master's thesis, Università Politecnica delle Marche, Italy, 2020.
- [6] Wikipedia. Indirizzo ip — wikipedia, l'enciclopedia libera, 2021. [Online; in data 10-settembre-2021].
- [7] Wikipedia. Domain name system — wikipedia, l'enciclopedia libera, 2021. [Online; in data 10-settembre-2021].
- [8] Christopher C Elisan and Mikko Hypponen. *Malware, rootkits & botnets: A beginner's guide*. McGraw-Hill New York, 2013.
- [9] Alessandro Mazzetti. *Reti neurali artificiali*. Apogeo, 1991.

Bibliografia

- [10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [11] Wikipedia. Rete neurale feed-forward — wikipedia, l’enciclopedia libera, 2020. [Online; in data 12-settembre-2021].
- [12] Wikipedia. Apprendimento automatico — wikipedia, l’enciclopedia libera, 2021. [Online; in data 13-settembre-2021].
- [13] Raaghavi Sivaguru, Chhaya Choudhary, Bin Yu, Vadym Tymchenko, Anderson Nascimento, and Martine De Cock. An evaluation of dga classifiers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5058–5067. IEEE, 2018.
- [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] Mattia Zago, Manuel Gil Pérez, and Gregorio Martínez Pérez. Scalable detection of botnets based on dga. *Soft Computing*, 24(8):5517–5537, 2020.
- [16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Ringraziamenti

Completando questo lavoro, si è chiuso un percorso di studi interessante e non privo di difficoltà e fatica.

Innanzitutto vorrei ringraziare il mio relatore, il prof. Luca Spalazzi, e i correlatori, il prof. Alessandro Cucchiarelli e il prof. Christian Morbidoni, per essere stati sempre disponibili, mettendo a disposizione tempo, risorse e conoscenze, per permettermi di completare questo lavoro di tesi.

Ringrazio i miei genitori, e tutta la mia famiglia per avermi dato la possibilità di intraprendere questo percorso di studi, supportandomi ogni giorno, anche in quelli più difficili.

Ringrazio tutti i miei amici, che anche nei momenti più difficili durante questo percorso, mi hanno sempre tirato su di morale, non facendomi mai perdere il sorriso e la voglia di andare avanti.

Infine un ringraziamento particolare va a Miriam, che mi è sempre stata vicino in ogni circostanza, sostenendomi in ogni mia scelta. Grazie di tutto, ti voglio bene.

Ancona, Ottobre 2021

Lorenzo Tiseni