UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica L-31

TESI DI LAUREA

PROFILAZIONE DI ATTIVITÀ ILLECITE NEL DEEP E DARK WEB MEDIANTE LATENT DIRICHLET ALLOCATION ED ALGORITMI GENETICI

RELATORE

Prof. **Fabio Palomba**

Università degli studi di Salerno

CANDIDATO

Emanuele Fittipaldi

Matricola: 0512105816

Anno Accademico 2020-2021

Ringrazio la mia famiglia e chiunque abbia creduto in me durante questo percorso.

Sommario

Le attività illecite hanno trovato terreno fertile nell'anonimato garantito dal Deep/Dark-Web. E' nata l'esigenza di riuscire a riconoscere e dunque a profilare i siti web dove si perpetrano attività illecite. Una possibile idea è quella di adottare delle tecniche di intelligenza artificiale per riuscire a fare ciò. L'obiettivo che si vuole raggiungere con questa tesi è quello di riuscire ad effettuare una predizione accurata dell'attività perpetrata su un determinato sito appartenente al Deep/Dark-web a partire dal solo testo che vi è ivi presente, adottando tecniche di Topic Modelling come LDA e attraverso l'ottimizzazione degli iperparametri mediante l'adozione di algoritmi genetici. Per riuscire a fornire ad LDA una base testuale di partenza qualitativamente più alta possibile, è stata creata una opportuna pipeline di lavoro dedita al preprocessing e all'estrazione del testo, cercando di abbracciare il problema nella forma più generale possibile aumentando la ripetibilità dell'esperimento.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Motivazioni e Obiettivi

É un errore pensare che la criminalità sia legata soltanto a fatti che avvengono nel "mondo reale", per strada, come lo spaccio di droga, la vendita di merce contraffatta e così via. La criminalità organizzata così come il singolo criminale, ha allargato i suoi orizzonti nel corso degli anni sempre di più, ricercando di continuo dei modi per garantire ai partecipanti del loro business transazioni sicure e sopratutto non tracciabili. Il grado di anonimato e la complessità di queste organizzazioni ha seguito uno sviluppo che va di pari passo con il progresso tecnologico.

L'ambiente che ha permesso alla criminalità organizzata di avere uno spazio sicuro dove poter esercitare attività illecite e garantirsi l'anonimato è diventato il mondo del web e nello specifico il Deep/Dark-Web, ovvero la porzione del web che non è indicizzata dai motori di ricerca.

Prima di tutto è bene fare una differenza tra Deep e Dark-Web. Per capire la differenza bisogna tenere conto che l'Internet con il quale abbiamo a che fare tutti i giorni appartiene a quella che è la porzione *superficiale* di Internet. Più in dettaglo per *superficiale* si intende tutta quelle pagine del Web che è possibile raggiungere attraverso un motore di ricerca come per esempio Google, Bing etc. Queste pagine vengono "trovate" dal motore di ricerca grazie al lavoro svolto dai web crawler, ovvero software in grado di acquisire una copia di queste pagine web che poi indicizzano.

Al contrario nel Deep-Web ci sono tutte quelle pagine web che non sono indicizzate dai motori di ricerca, e quindi che non possiamo trovare tramite una ricerca su, per esempio Google. Data la natura "invisibile" ai motori di ricerca, si potrebbe pensare che queste pagine siano losche e popolate da criminali, ma non è sempre il caso. Esempi di pagine non indicizzate ma che sono perfettamente innocue sono per esempio le nostre email e le transazioni bancarie.

Come è intuibile però, esistono delle pagine appartenenti al Deep-Web usate dalla criminalità organizzata. Infatti, parliamo di Dark-Web per riferirci a questa piccola porzione di web non indicizzata che possiamo considerare un sottoinsieme del Deep-Web. Secondo le stime dei ricercatori della NASA, sono circa decine di migliaia gli indirizzi URL appartenenti a questa piccola porzione del Web, pochi comunque se confrontati alla grandezza del Web.

Le pagine che troviamo nel Dark-Web hanno un dominio ben specifico (.onion) le quali sono ospitate su dei server che utilizzano il protocollo Tor, protocollo sviluppato in origine dal dipartimento di difesa statunitense per consentire comunicazioni anonime e sicure. Purtroppo questo protocollo è diventato disponibile al pubblico nel 2004 dando il via ad utilizzi leciti come quelli legati alla protezione della propria privacy, ma anche ad utilizzi illeciti come quelli già accennati.

Sul Dark-Web è possibile trovare delle pagine specifiche per la vendita di qualsiasi tipo di merce acquistabile in Bitcoin, un tipo di cryptovaluta che consente l'anonimato per ogni transazione che si effettua. Il lavoro di tesi qui proposto è quello di riuscire a trovare dei modi alternativi per aiutare le LEA a scovare, in modo quasi del tutto automatico, luoghi di questo genere al fine di porre un freno alla crescita esponenziale di tali fenomeni. La tecnica alternativa che è stata proposta è quella di basarsi soltanto sul contenuto testuale che è possibile estrapolare da questi siti tramite un opportuno processo di *Scraping*. L'obiettivo è stato quello di inferire delle caratteristiche attraverso il modello di Machine Learning *Latent Dirichlet Allocation* a partire dal testo estratto dalle pagine web contenute nel dataset ANITA. Questo dataset contiene diverse centinaia di dump dei maggiori Marketplace attivi nel Dark-Web. Una volta estratte queste caratteristiche, sono state usate per ricavare le features con le quali si potesse allenare un successivo modello di Machine Learning, un Decision Tree al fine di performare la classificazione per inferire il crimine perpetrato sulla pagina web in questione.

1.2 Risultati

I risultati hanno dimostrato che effettivamente un approccio text based associato al campo della Cyber Threat Intelligence è possibile. Da questa tesi è emerso che ogni sito web, o qualsivoglia fonte di natura digitale dove c'è del testo che è possibile estrarre dispone di una *impronta digitale* che nel corso della tesi si è scoperto essere la *distribuzione di probabilità dei Topic latenti* nel testo che è possibile estrarre grazie ad LDA. E' stato dimostrato inoltre che esiste una correlazione tra questa distribuzione di probabilità e le label associate ad un testo, e questo ci ha permesso di applicare tecniche di apprendimento supervisionato come Decision Tree. Anche se i risultati in termini di accuratezza non son egregi questo vuole essere una indice di quanto ancora è possibile esplorare questa direzione e quanto tutta la pipeline creata abbia ampia possibilità di configurazione. Questa tesi non vuole essere una soluzione esaustiva al problema affrontato ma bensì un contributo alla ricerca di nuove strate per aiutare le LEA a contrastare il fenomeno della criminalità organizzata online.

1.3 Struttura della tesi

La tesi è strutturata nel seguente modo:

- **Elenco delle figure**
- **Elenco delle tabelle**
- **Introduzione** - Quali sono state le motivazioni che mi hanno spinto a trattare l'argomento oggetto di tesi e quali sono gli obiettivi che voglio raggiungere tramite l'approccio qui proposto.
- **Stato dell'arte** - Analisi del problema della classificazione dei siti appartenenti al Deep/Dark-web, delle tecniche attualmente disponibili nello stato dell'arte e degli studi condotti sull'argomento. Analisi di come Mark-V è inquadrabile panorame dello stato dell'arte e cenni del progetto ANITA.
- **Mark-V: Una tecnica di intelligenza artificiale per il riconoscimento dei siti del Deep/Dark-Web** - In questo capitolo viene presentata ed analizzata l'intera pipeline di MARK-V. Vengono affrontate in quattro sezioni diverse le fasi principali di cui questa pipeline è composta, ovvero Preprocessamento, Training di LDA, Tuning degli iperparametri e conversione in problema supervisionato mediante feature engineering ed algoritmi di classificazione.

- **Valutazione Preliminare** - In questo capitolo vengono espressi quali sono stati gli obiettivi dello studio empirico, e quali sono state le domande di ricerca alle quali si è tentato di dare una risposta attraverso il lavoro svolto in questa tesi. Vengono inoltre analizzati sia il contesto di studio e sia come i dataset utilizzati sono stati preparati per questo studio. Il capitolo termina con l'esporre qual è stata la metodologia di validazione, esplorando la metrica scelta ed infine i risultati ottenuti applicando MARK-V al problema della classificazione dei siti web appartenenti al Deep/Dark-Web.
- **Conclusioni** - In questo capitolo vengono espressi quali sono stati i risultati, quali sono i possibili sviluppi futuri di MARK-V
- **Bibliografia** - Riferimenti Bibliografici e a risorse prese dal Web.

CAPITOLO 2

Background

Per capire a fondo l'approccio tentato in questa tesi riguardo lo sviluppo di una nuova tecnica per aiutare le forze dell'ordine nell'individuare siti malevoli, diventa imprescindibile familiarizzare con gli elementi che ne costituiscono i punti cardini. Di conseguenza provvederò a fornire al lettore gli elementi per poter comprendere appieno il lavoro qui proposto. Durante lo sviluppo di questo progetto mi sono imbattuto in diverse tecnologie e algoritmi nuovi, uno di questi è LDA Blei et al. [2003] il quale sta per Latent Dirichlet Allocation per via della distribuzione di Dirichlet su cui questa tecnica di Machine Learning si fonda. Prima di descrivere il suo funzionamento a grandi linee, è necessario precisare il significato di alcuni termini che userò da qui in poi. Un documento siamo abituati a pensarlo come un foglio di carta più o meno riempito di parole, chiameremo di conseguenza "documento" una qualsiasi fonte testuale. L'insieme di più documenti lo chiamiamo corpus, mentre un Topic è un insieme a cui una parola appartiene. In questo insieme sono presenti anche altre parole, le quali tutte insieme creano un contesto in cui è probabile trovare quelel parole, ovvero un Topic per l'appunto.

2.1 LDA

2.1.1 Il problema

Supponiamo di avere molti documenti, quindi un corpus di documenti. Ogni documento ha un Topic, ovvero una serie di argomenti che sono trattati in quello specifico documento. Il

problema è che non conosciamo in anticipo quali sono i Topic trattati in questi documenti, tutto quello di cui disponiamo sono le parole ivi contenute. LDA ci permette di capire quali sono questi Topic e di poter descrivere ciascun documento come una percentuale di ciascuno di essi. Noi esseri umani siamo in grado di dedurre di cosa parla un testo di senso compiuto, ma al contrario di come si potrebbe pensare, siamo in grado di dedurre i Topic di un documento anche se esso fosse soltanto una collezione di parole e dunque non di senso compiuto. Come facciamo? semplicemente conoscendo il significato delle parole e il contesto in cui tipicamente esse vengono usate siamo in grado di capire se un determinato documento stia parlando di sport invece che di scienza e cos' via.

il problema è che il computer non può contare sulla semantica delle parole per trarre informazione, e di conseguenza si deve affidare ad altri stratagemmi. E' qui che entra in gioco LDA il quale è in grado di creare una figura geometrica ad n dimensioni avente per vertici una serie di Topic e di piazzare ogni documento all'interno di questo spazio in modo corretto. La vicinanza di questo documento ad uno specifico Topic indica che quel determinato Topic è presente in quantità maggiori rispetto ad un altro Topic associato ad un vertice più lontano. Il modo in cui LDA riesce a piazzare correttamente ogni documento all'interno di questo spazio è tramite una moltitudine di "macchine genera documenti", tra le quali ce ne sarà sicuramente una che è capace di generare un testo molto simile al testo di partenza. Diciamo di conseguenza che tale macchina dispone del settaggio migliore. Una volta ottenuto il settaggio migliore abbiamo di conseguenza anche le percentuali dei vari Topic contenuti in quel documento.

2.1.2 Schema della macchina LDA

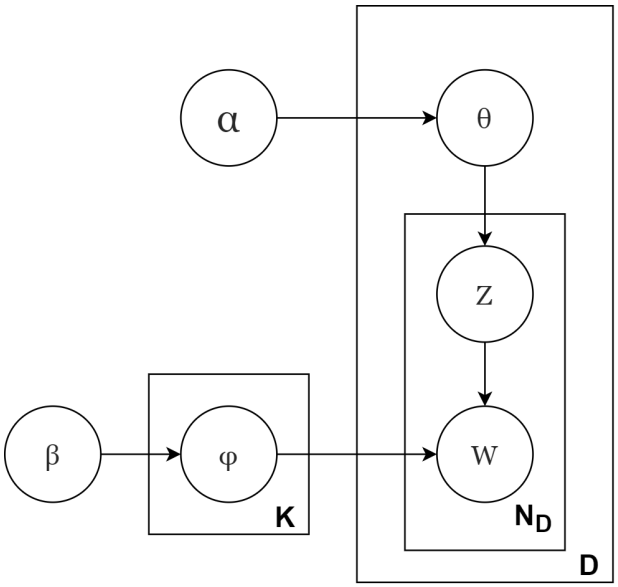

Figura 2.1: Schema della macchina LDA

Questo è lo schema completo della macchina LDA. Esaminiamo adesso le componenti di questa macchina. I parametri α e β rappresentano le distribuzioni di Dirichlet. θ e ϕ invece rappresentano delle distribuzioni multinomiali. Seguendo il percorso demarcato dalla macchina vediamo come a partire da queste distribuzioni multinomiali andiamo a creare dei Topic **z** associati ad ogni parola **w**. E' in questo modo che andiamo a creare i documenti, unendo queste parole insieme, dove ogni parola è associata ad un Topic.

$$P(W, Z, \theta, \phi, \alpha, \beta) = \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{i=1}^{K} P(\phi_i; \beta) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j)\ P(W_{j,t}|\phi_{Z_{j,t}})$$

Sulla sinistra abbiamo la probabilità che un documento appaia, perché, come abbiamo appena detto, questa macchina potrebbe generare qualsiasi documento vogliamo, con una certa (piccolissima) probabilità. Subito dopo il segno di uguaglianza troviamo due fattori di estrema importanza, ovvero i parametri α e β. Questi parametri vengono chiamati **iperparametri** ed è proprio tramite questi che è possibile calibrare opportunamente il funzionamento di LDA. Il parametro α nelle distribuzioni di Dirichlet può variare da <1, =1 e >1. Quando α è uguale ad 1, abbiamo una distribuzione uniforme dei punti/documenti nello spazio creato dai Topic, quando è inferiore ad 1 abbiamo che la distribuzione gravita di più verso i vertici, quando invece α è maggiore di 1, la distribuzione è concentrata in una regione molto stretta di questo spazio. In base al numero di Topic lo spazio generato può essere una retta, nel caso di soli 2 Topic, un triangolo nel caso di 3 Topic, o un **simplesso n dimensionale** nel caso

di **n Topic**. Nella formula sopra non abbiamo un parametro α ma più di uno, uno per ogni Topic. Tra le varie possibilità, quello che è di nostro interesse è il caso in cui α sia minore di 1, ovvero il caso in cui ogni documento avrà una certa percentuale che indica la sua vicinanza rispetto ad un determinato Topic invece che ad un altro. Come vedremo nel seguito di questa tesi, per trovare il settaggio ottimale per LDA è stato usato un Algoritmo Genetico il quale agisce proprio sugli iperparametri α e β, ma esamineremo questo aspetto più avanti nella trattazione. Moltiplicando tutti questi fattori infine abbiamo la probabilità di ottenere un dato documento.

la distribuzione di Dirichlet è sostanzialmente una distribuzione di distribuzioni, dato che ogni punto/documento ha una posizione specifica nello spazio e di conseguenza una determinata configurazione di percentuali di diversi Topic. Sono proprio queste diverse percentuali di Topic di cui è composto ogni documento che vengono usate come distribuzioni multinomiali.

Sino ad ora abbiamo esaminato la distribuzione di Dirichlet che si occupa di associare ad ogni documento determinati Topic, ma non dobbiamo dimenticare l'esistenza anche della distribuzione di Dirichlet che crea l'associazione tra i Topic e le parole. Come possiamo mettere adesso le due cose insieme? La cosa che dobbiamo ricordare è che i due settaggi α e β rappresentano proprio queste due distribuzioni. Di conseguenza, cambiare questi due iperparametri significa andare a cambiare la disposizione dei rispettivi punti all'interno dei rispettivi simplessi n dimensionali. E' chiaro dunque come questo mi permette di andare a produrre documenti diversi.

2.1.3 Esempio pratico

$$P(W, Z, \theta, \phi, \alpha, \beta) = \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{i=1}^{K} P(\phi_i; \beta) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j)\ P(W_{j,t}|\phi_{Z_{j,t}})$$

Questa formula ci è già familiare. Vediamo adesso come usarla in modo tale da produrre un documento. Il promo fattore di questa formula è la distribuzione di Dirichlet nella quale gli angoli del simplesso n dimensionale rappresentano i Topic. All'interno di questo spazio prendiamo dunque un punto. Il punto preso rappresenta il documento che vogliamo creare. In base alla posizione di questo punto all'interno del simplesso, sono in grado di esprimere con esattezza le percentuali di ciascun Topic che concorre a determinare la posizione del documento nello spazio. Il documento dunque, a questo stadio è soltanto un insieme di percentuali, dove ogni percentuale simboleggia quanto di quel Topic è presente nel documento. Queste percentuali sono estremamente importanti in quanto vengono impiegate all'interno della distribuzione multinomiale θ all'interno della prima probabilità condizionata nella terza produttoria. Se per esempio abbiamo le pecentuali 70% Topic scienza, 10% Topic politica, 20% Topic sport, tramite la distribuzione multinomiale vado a segnalare che il 70% delle parole devono appartenere al Topic scienza, il 10% al Topic politica e il 20% al Topic sport. In tal modo abbiamo definito le "etichette" delle parole che devono essere inserite nel documento, le quali possono essere di numero arbitrario. Il pescaggio dei Topic avviene in modo deltutto casuale. Man mano che estraiamo queste etichette ce le annotiamo da qualche parte. Questo crea una configurazione di Topic casuale ma che rispetta le proporzioni definite dalle percentuali ricavate dalla prima produttoria. un esempio di configurazione è:

- science
- science
- sports
- science
- science
- politics
- sports
- science

Ora, dato che queste etichette rapresentano i Topic per le nostre parole, dobbiamo andare a trovare le parole appartenenti a questi Topic che abbiamo pescato. Per fare questo usiamo la seconda distriuzione di Dirichlet, β. Questa distribuzione associa i Topic alle parole, in quanto per comporre il nostro documento abbiamo bisogno delle parole vere e proprie e non solo il Topic di appartenenza di ciascuna di esse. Dato un Topic dunque, questa distribuzione di Dirichlet permette di collocarlo all'interno del simplesso n dimensionale avente per vertici un insieme di parole legate a quel Topic. Questo Topic, in modo simile a come abbiamo ragionato nel caso di un documento nello spazio del simplesso n dimensionale relativo ai Topic, può essere descritto tramite una serie di percentuali relative a ciascuna parola appartenente al Topic.Prendiamo in considerazione l'esempio precedente. Prendiamo il Topic science. Vediamo che questo Topic, all'interno del simplesso relativo alle parole per questo Topic si colloca in modo tale da produrre le seguenti percentuali per le seguenti parole: 40% galaxy, 40% planet, 10% ball, 10% referendum. Facendo lo stesso anche con gli altri Topic otterremo grazie alla seconda probabilità condizionata nella terza produttoria della formula di cui sorpra una serie di distribuzioni di parole in base alle percentuali opportunamente ricavate nella distribuzione di Dirichlet per i Topic. Una volta fatto questo andiamo ad associare le parole che abbiamo trovato riguardo ogni Topic, ad ogni Topic della configurazione del documento precedentemente ottenuta. Per ogni Topic nella configurazione pescheremo una parola a caso dalle distribuzioni di parole ottenute tramite la seconda distribuzione multinomiale ϕ.

Il risultato finale sarà che ogni parola ha una etichetta che indica a quale Topic appartiene e per ogni parola etichettata con un determinato Topic possiamo trovare parole diverse, legate sempre a quel Topic, il tutto rispettando le percentuali che ci siamo imposti per i Topic e per le parole. In questo modo andiamo a generare una collezione di questi documenti, e una volta che ne abbiamo tanti, andiamo a verificare quanti somigliano al documento di partenza. Ovviamente la probabilità che otteniamo lo stesso identico documento è estremamente bassa, ma ci sono delle configurazioni peggiori che danno risultati ancora più negativi. E' chiaro quindi la possibilità di poter individuare una macchina LDA che ha performato meglio delle altre macchine. Costei sarà la "vincitrice" in quanto ha la più alta probabilità di restituirci il documento originale.

2.1.4 Formato interpretabile dal modello LDA

Nella implementazione in Python, il modello LDA per poter operare ha bisogno di maneggiare i documenti opportunamente convertiti in un formato numerico. Due sono i modelli che ci permettono di ottenere una tale conversione, il modello Bag of Words e il modello TF-IDF.

2.1.5 Bag of Words

Il Bag of Words a differenza del TF-IDF è orientato di più al capire la frequenza delle parole all'interno di un testo. BoW è un algoritmo che conta quante volte una parola appare in un documento. E' un conteggio. Questi conteggi di parole ci permettono di andare a comparare documenti e capire le loro similitudini per applicazioni quali la ricerca, classificazione di documenti e Topic Modelling. BoW è anche un metodo per preparare il testo per darlo in input ad una rete di deep-learning. BoW lista le parole in coppia al loro conteggio per documento. Nella tabella dove sono salvate le parole e i documenti che diventeranno vettori, ogni riga è un documento e ogni colonna è una parola con un conteggio. BoW è la forma più semplice di rappresentazione del testi, in numeri, sottoforma di un vettore. Prendiamo in esame queste frasi per fare un esempio:

- Review 1: This movie is very scary and long
- Review 2: This movie is not scary and is slow
- Review 3: This movie is spooky and good

Costruiremo prima un vocabolario a partire da tutte le parole uniche che ci sono tra queste frasi. Il vocabolario consisterà di 11 parole:

```
'This','movie','is','very','scary','and','long','not','slow','spooky','good'
```

Possiamo adesso prendere ognuna di queste parole e segnare con un 1 se è comparsa nella review i-esima oppure 0 se non è comparsa. Questo ci darà 3 vettori.

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

Tabella 2.1: Rappresentazione Bag of Words

Le Review possono essere dunque espresse in forma vettoriale:

- Vector of Review 1:[1 1 1 1 1 1 1 0 0 0 0]
- Vector of Review 2:[1 1 2 0 0 1 1 0 1 0 0]
- Vector of Review 3:[1 1 1 0 0 0 1 0 0 1 1]

Il vettore è lungo tante quante sono le parole presenti nel dizionario. Leggendo il primo vettore vediamo che gli 1 sono in corrispondenza delle parole *this,movie,is,very,scary,and,long*. Quindi sappiamo che la Review 1 contiene esattamente quelel parole.

2.1.6 Problemi del Bag of Words

Nell'esempio appena visto, possiamo avere vettori di lunghezza 11. Incominciamo inoltre a fronteggiare problemi, come quando in una frase:

- se una nuova frase contiene nuove parole, allora il nostro vocabolario crescerebbe di dimensione e quindi si allunga anche la dimensione dei vettori. Questo sta comunque molto alla implementazione del modello, in quanto in alcune implementazioni di BoW come quella fornita da Gensim, le parole che non sono nel dizionario non vengono assimilate.
- In aggiunta i vettori potrebbero contenere anche diversi 0, e quindi risultare in una matrice sparsa, che è quello che vogliamo evitare. Una matrice sparsa è una matrice i cui valori sono quasi tutti uguali a zero.

2.1.7 TF-IDF

TF-IDF a differenza del BoW è orientato a capire la rilevanza delle parole. Wikipedia dice infatti che TF-IDF è una statistica numerica che è intesa per riflettere quanto imporatnte una parola è relativamente ad un documento, in una collezione o corpus. Il problema con la rappresentazione Bag of Words words è che non c'è molta semantica legata alle parole, in quanto abbiamo soltanto 0,1, ovvero valori interi. TF-IDF invece considera sia la frequenza di una parola e sia la rarità di quella parola, ovvero quanto spesso è stata incontrata tra tutti i documenti. A differenza del Bag of Word, quindi, abbiamo dei valori reali, piuttosto che 0 ed 1.

Per fare un esempio pratico prendiamo in considerazione queste frasi:

- frase 1: good boy
- frase 2: good girl
- frase 3: boy girl good

Si va a performare dunque il conteggio della frequenza di ogni termine tra tutte le frasi/documenti:

- good 3
- boy 2
- girl 2

Adesso la prima cosa da fare per applicare TF-IDF, è applicare il TF ovvero il *Term Frequency*. La formula per il calcolo del Term Frequency è il:

$$tf_{t,d} = \frac{n_{t,d}}{Number\ of\ terms\ in\ the\ document}$$

Quindi per l'esempio corrente abbiamo:

	Frase1	Frase2	Frase 3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

Tabella 2.2: Calcolo del Term Frequency

Adesso bisogna calcolare la Inverse document frequency, la cui formula è:

$$idf_t = log\frac{number\ of\ documents}{number\ of\ documents\ with\ term\ 't'}$$

IDF misura quanto importante è un termine:

Parole	IDF
good	$\log(\frac{3}{3})$=0
boy	$\log(\frac{3}{2})$
girl	$\log(\frac{3}{2})$

Tabella 2.3: Calcolo dell'IDF

Adesso per calcolare TF-IDF basta moltiplicare i valori di TF*IDF:

	F1	F2	F3	O/P
	good	boy	girl	
Frase 1	$(\frac{1}{2} * log(\frac{3}{3})) = 0$	$(\frac{1}{2} * log(\frac{3}{2}))$	$0 * log(\frac{3}{2}) = 0$	
Frase 2	0	$log(\frac{3}{2}) * 0 = 0$	$\frac{1}{2} * log(\frac{3}{2}))$	
Frase 3	$(\frac{1}{3} * log(\frac{3}{3})) = 0$	$(\frac{1}{3} * log(\frac{3}{2}))$	$(\frac{1}{3} * log(\frac{3}{2}))$	

Tabella 2.4: Calcolo TF*IDF

Qui si può già evidenziare la differenza con Bag of Words. Se osserviamo la prima frase, l'importanza è stata data alla parola "boy" in quanto ha un valore più alto rispetto alle altre

parole nella stessa frase che hanno valore 0. Nella seconda frase, allo stesso modo, l'importanza è stata data alla parola "girl", mentre invece nell'ultima frase "boy" e "girl" hanno la stessa importanza.

TF-IDF da un valore maggiore alle parole che sono meno frequenti e questo valore sarà più alto quando i valori IDF e TF sono alti, ovvero quando le parole sono rare in tutti i documenti combinati, ma frequenti in un singolo documento.

Le considerazioni finali su BoW e TF-IDF sono dunque due:

- BoW crea un insieme di vettori contenenti il conto delle occorrenze delle parole nei documenti mentre TF-IDF contiene le informazioni sulle parole più importanti e meno importanti.
- I vettori Bag of Words sono facili da interpretare. Tuttavia, TF-IDF di solito performa meglio nei modelli di machine learning.

2.1.8 Come valutare le performance di LDA

Abbiamo appena detto che la funzione obiettivo che GA tenterà di ottimizzare sempre di più non è nient'altro che la misura di coerenza del modello di Topic modelling impiegato nel progetto. Prima di capire però cosa è la coerenza dei Topic, dobbiamo capire cosa è la misura di perplessità. Anche la perplessità è una delle metriche intrinseche di valutazione, ed è ampiamente usata per la valutazione di modelli basati sul linguaggio naturale. Essa cattura quanto un modello è "sorpreso" rispetto alla visione di nuovi dati che non ha già visto in precedenza, ed è misurata come la verosimiglianza normalizzata su un opportuno test set tenuto fuori dal processo di training. Focalizzandoci sulla verosimiglianza normalizzata, possiamo pensare alla metrica di perplessità come una misura di quanto è probabile che nuovi dati mai visti siano dati al modello allenato in precedenza. Ovvero, quanto bene il modello rappresenta o riproduce le statistiche dei dati appartenenti al test set. Tuttavia, studi recenti hanno mostrato che la probabilità predittiva (o in modo equivalente, perplessità) e il giudizio umano non sono spesso correlati e alcune volte sono persino divergenti. Ottimizzare la perplessità potrebbe non portare a Topics che sono interpretabili dall'essere umano. Questa limitazione della misura di perplessità è servita come motivazione per cercare di modellare il giudizio umano, dando vita alla coerenza dei Topic. Il concetto della coerenza del Topic combina un certo numero di misurazioni in un quadro per valutare la coerenza tra Topic

dedotti da un modello. Ma cosa è la coerenza del Topic?. Le misure di coerenza dell'argomento assegnano un punteggio ad un Topic misurando il grado di somiglianza semantica tra le parole con un punteggio più elevato nel Topic. Queste misure aiutano a distinguere tra Topic che sono semanticamente interpretabili e Topic che costituiscono artefatti di inferenza statistica. Cosa è la coerenza? Un insieme di frasi o fatti è detto "coerente", se essi si supportano a vicenda. In questo modo, un insieme di fatti coerente può essere interpretato in un contesto che copre tutti o la maggior parte dei fatti. Un esempio di un insieme di fatti coerenti è "the game is a team sport", "the game is played with a ball", "the game demands great physical efforts". Esistono diverse misure di coerenza. Citiamo:

- C_v - è la misura basata su una sliding window, ovvero la segmentazione di un insieme delle prime top words ed una misura di conferma indiretta che usa informazioni mutue puntuali normalizzate (NPMI) e la similarità del coseno
- C_p - è basata su una sliding window, ovvero la segmentazione uno-precedente delle prime top words e la misura di conferma della coerenza di Fitelson's
- C_uci - è basata su una sliding window ed una informazione reciproca punto per punto (PMI) di tutte le coppie di parole delle top word fornite.
- C_umass - si basa sui conteggi di co-occorrenza dei documenti, una segmentazione uno-precedente ed una probabilità condizionata logaritmica come misura di conferma.
- C_npmi - rappresenta una versione migliorata della coerenza C_uci. Essa usa la versione normalizzata della informazione reciproca punto per punto (NPMI)
- C_a - è basata su una finestra di contesti, una comparazione coppia per coppia delle top words ed una misura di conferma indiretta che usa la NPMI e la similarità del coseno.

Per questo progetto è stata presa in considerazione la misura di coerenza C_umass.

2.2 Algoritmi Genetici

Un algoritmo genetico è una variante della beam search stocastica. Esso inizia con un insieme di k stati generati casualmente. Questo insieme prende il nome di *popolazione*, mentre ogni stato prende il nome di *individuo*. Ogni individuo è rappresentato come una stringa su un alfabeto finito. Ogni individuo presente nella popolazione viene valutato da una *funzione di fitness* tramite la quale siamo in grado di capire la probabilità che l'individuo ha di essere

scelto per la riproduzione entrando di fatto nel *mating pool*. Dopo la fase della formazione delle coppie avviene la riproduzione, chiamata in questo caso *crossover*, dove vengono creati nuovi individui i quali avranno ciascuno parte del patrimonio genetico dei genitori. Infine c'è la fase della *mutazione* nella quale andiamo ad alterare secondo criteri che scegliamo noi, il patrimonio genetico degli individui al fine da scongiurare una convergenza prematura, esplorare meglio lo spazio di ricerca e mantenere allo stesso tempo la popolazione il più diversificata possibile.

Tutto questo procedimento ci porta ad una nuova generazione che è molto probabilmente migliore di quella che l'ha preceduta. Reiterando il processo otteniamo delle popolazioni sempre migliori, ma arrivati ad un certo punto possiamo notare un peggioramento. In questo caso potremmo decidere di terminare la ricerca. Si scarta dunque questa generazione la quale non ha fornito progressi e torniamo alla precedente dalla quale scegliamo l'individuo migliore.

Gli algoritmi genetici sono una procedura di alto livello ispirata alla genetica per definire un algoritmo di ricerca. Genetic Algorithms non è il nome di un algoritmo di ricerca specifico, essi ci permettono di definire un algoritmo di ricerca. Gli GA sono molto utili in scenari black box dove non conosciamo molti dettagli sulla funzione di fitness. Un GA potenzialmete potrebbe non fallire mai, nel senso che una soluzione, seppur non ottimale, la può sempre restituire. E' stato questo il motivo per il quale è sembrata sin da subito la soluzione migliore per trovare una configurazione sub-ottima per LDA.

C'è infine da menzionare anche la grande possibilità di personalizzazione degli algoritmi genetici. Siamo in grado infatti di specificare:

- Dimensione della popolazione
- Dimensione del mating pool
- Algoritmi di Crossover
  - Single Point
  - Two Point
  - K-Point
  - Uniform
  - Arithmetic

- Algoritmi di mutazione
  - Bit flip
  - Random resetting
  - Swap
  - Scramble
  - Inversione
- Inizializzazione
- Rappresentazione degli individui
- Algoritmo di selezione
  - Truncation
  - Roulette wheel
  - (K-way) Tournament
  - pressione di selezione
  - Truncation Selection
- Ordinamento degli individui
- Criteri di stopping conditions
  - Tempo di esecuzione
  - Costo
  - Numero di generazioni
  - Assenza di miglioramenti
  - Ibride
  - Problem specific

CAPITOLO 3

Stato dell'arte

3.1 Problema in esame

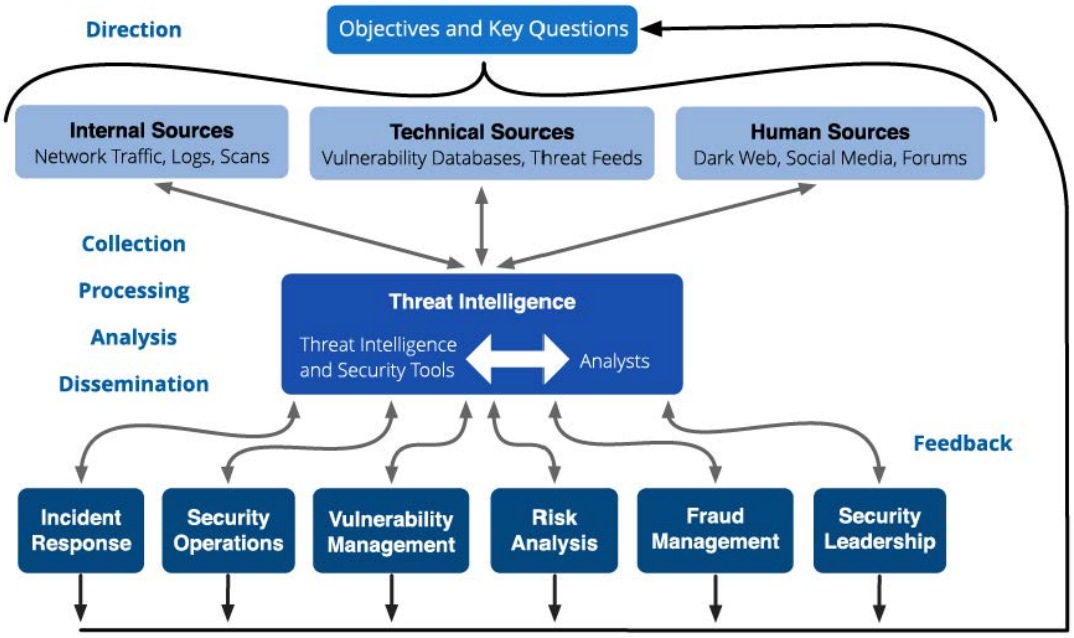

Figura 3.1: intelligence cycle

Nel corso degli anni gli attacchi informatici sono diventati sempre più ricorrenti ed elaborati. I danni che derivano da tali attacchi hanno motivato le aziende e i governi a

prendere il fenomeno sul serio Cascavilla et al. [2021], investendo parte dei propri ricavi nella cyber-security. Inoltre, Parallelamente, abbiamo dei danni che vengono sistematicamente inferti alla società tramite attività illecite che continuano a perpetrarsi e a proliferare sempre più velocemente grazie all'anonimato garantito dal Dark-web. Contrastare questo fenomeno non è facile, dato che collezionare e labellare manualmente contenuti appartenenti a questo strato nascosto del web è laborioso e richiede molto tempo, diventando una sfida per la ricerca. Attualmente troviamo diverse soluzioni per riuscire a fare ciò, le quali soluzioni verranno di seguito analizzate. Tali soluzioni costituiscono lo stato dell'arte corrente per la classificazione di siti web appartenenti al Deep/Dark-web.

Per contrastare la criminalità organizzata 2.0 nel corso del tempo sotto lo stesso ombrello della *threat intelligence* si è unita la *cyber threat intelligence*. Questa disciplina è nata con l'intento di fornire informazioni rifinite, analizzate ed organizzate su quelli che sono i potenziali attacchi informatici che possono danneggiare un organizzazione, sia essa governativa che non governativa. L'intelligence tradizionale si basa su sei fasi distinte. Tali fasi formano il cosiddetto *intelligence cycle*. Tali fasi sono:

- Direzione: In questa fase si cercano di prefissare quali sono gli obiettivi da raggiungere. Si valuta qual è l'impatto potenziale che si andrebbe a creare interrompendo il processo criminale e quali sono le priorità in termini di cosa proteggere, sia in termini di vite dei singoli individui, e sia protezione del patrimonio informativo e dei processi aziendali.

- Collezione: Questa è la fase in cui si fa information gathering. Si passa in rassegna ogni informazioni liberamente fruibile in rete, blog e market. Per esempio si possono trarre delle statistiche in merito al comportamento degli utenti presenti nei dark market o applicare tecniche di *Thematic Coding* ai blog dei terroristi. Infine si può effettuare lo scraping di siti web e forum o infiltrarsi in essi.

- Processamento: Questa è la fase in cui si trasformano le informazioni collezionate in un formato usabile.

- Analisi: La fase dell'analisi è critica in quanto il fattore umano è decisivo. Infatti sarà tramite un'attenta analisi delle forze dell'ordine o dell'organizzazione che si riesce a trasformare le informazioni in *intelligence* permettendo di prendere delle decisioni informate.

- Disseminazione: La disseminazione è la fase in cui la conoscenza derivata dalla fase di analisi viene condivisa con l'intera organizzazione.

- Feedback: infine troviamo la fase di feedback. Ovviamente per capire se si è presa la direzione giusta abbiamo bisogno di collezionare dei feedback. I feedback ci permettono di reinterpretare quali sono le proprità dell'intelligence, quali sono le decisioni più opportune da intraprendere, quali dati collezionare e quali sono le fasi per processare questi dati in modo tale da trasformarli in informazioni utili.

3.2 Tecniche attualmente disponibili

Vi sono alcune tecniche particolarmente interessanti appartenenti al momento a quello che viene considerato lo stato dell'arte della Cyber Threat Intelligence. Thorat et al. [2020] hanno pubblicato nel 2020 sull *International Research Jurnal of Engineering and Technology* (IRJET) una tecnica per classificare i siti web sfruttando il codice penele indiano (IPC-Indian Penal Code). La tecnica si basa sulla selezione in modo del tutto arbitrario e creativo, di una serie di leggi e regolameti inerenti ad ogni tipologia di attività illegali per allenare diversi classificatori. Dalle categorie come: droga, armi, pedo-pornografia, contraffazione, gioco d'azzardo, hanno scelto i documenti legali corrispondenti presenti nell'IPC per effettuare dell'allenamento supervisionato. Successivamente un algoritmo di classificazione come il Naive Bayes Classifier, si occupa di classificare il contenuto illegale sulle pagine web. L'architettura di questo sistema è divisa in tre parti principali. La prima parte la quale si occupa di collezionare delle leggi e regolamenti di rilievo per costruire il datast di training, la seconda parte la quale si occupa di estrarre effettivamente il contenuto da forum presenti nel dark web per costruire il dataset di testing ed infine l'ultima parte dove vengono usati i dati del dataset di training per allenare il modello e il classificatore per testare i dati appartenenti al dataset di testing. Su i due dataset così creati, viene applicato del pre-processing usando la possibilità di TF-IDF di poter estrarre delle feature per costruire uno modello spazio vettoriale del testo estratto.

Una tecnica molto simile basata invece sull'uso combinato di TF-DIF in aggiunta ad un algoritmo di regressione Logistica è stato presentato da Al Nabki et al. [2017]. In questo paper, oltre che a presentare la tecnica è stato fornito pubblicamente un dataset contenente diversi domini attivi risiedenti nel Dark-web. Il dataset in questione è il DUTA, acronimo di **"D**arknet **U**sage **T**ext **A**ddresses". Questo dataset è stato creato campionando la rete Tor sul periodo di due mesi e labellando manualmente ogni indirizzo in 26 classi distinte. Riprendendo quanto espresso nelle sezioni precedenti, labellare manualmente i siti web appartenenti al Dark-web è molto oneroso in termini di tempo, ma Al Nabki et al. hanno deciso di fare ciò per disporre

di un dataset su cui verificare l'accuratezza della tecnica proposta, e per fornire a chiunque volesse proseguire questo lavoro, un dataset labellato.

Una tecnica leggermente diversa viene invece proposta da De Fausti et al. [2019]. Il lavoro svolto da De Fausti et al. spinge sullo sfruttamento di enormi quantità di dati testuali automaticamente raschiati da siti web di imprese italiane in modo da predire un insieme di variabili selezionate che sono attualmente oggetto di studio da parte dell'ISTAT. E' ovvio come tale lavoro può essere esteso anche al contenuto testuale che è possibile estrarre dal Deep e Dark-web. Il problema è stato affrontato come problema di classificazione testuale, dove un algoritmo deve imparare ad inferire se una certa impresa italiana stia performando e-commerce a partire dal contenuto testuale presente sul sito dedicato. Per raggiungere questo obiettivo, De Fausti et al. hanno impiegato l'uso di Reti Neurali Convoluzionali e hanno fatto affidamento al Word Embedding per codificare testo grezzo in immagini a scala di grigi. Un'altra problematica affrontata in questa pipeline innovativa è stata quella della carenza di contenuto testuale utile rispeto alla quantità di rumore testuale presente sulle pagine. Per superare questo problema, De Fausti et al. hanno adoperato un framework conosciuto come Riduzione del Falso Positivo, il quale è stato raramente se del tutto mai usato in congiunta di un problema di classificazione testuale.

3.3 Studi condotti sull'argomento

Qui di seguito esaminiamo alcuni studi condotti recentemente pubblicati nel campo della cyber threat intelligece, e passiamo in rassegna quali sono stati i risultati e considerazioni emerse da tali lavori. Tounsi et al Tounsi and Rais [2018]. fornisce una overview di alcuni tool gratis open source e compara le loro caratteristiche con quelle di AlliaCERT TI. Attraverso la loro analisi, hanno scoperto che la rapida condivisione della threat intelligene, come incoraggiato da ogni organizzazione per cooperare, non è sufficiente per evitare attacchi mirati ed inoltre la fiducia si è rivelata essere estremamente importante per le compagnie che condividono informazioni personali. Un' altra problematica che è emersa a riguardo è quante informazione è necessaria condividere per prevenire gli attacchi e sopratutto in quale formato condividerle, al fine di evitare ogni perdita di informazione. Tounsi et al propone dunque una propria analisi per capire quale standard sia migliore e presenta una comparazione tra i migliori tool per la threat intelligence. troviamo poi il lavoro svolto da Toch et a Toch et al.

[2018]l. Questi autori pongono l'accento sul tipo di dati necessari ai sistemi di cybersecurity

che hanno l'obiettivo di proteggere la nostra provacy da occhi indiscreti. Nell'articolo viene illustrata una tassonomia la quale mostra che quasi tutte le categorie tecnologicehe inerenti alla cyber-security richiedono accesso alle informazioni personali e a dati sensibili. Questo risultato può essere da guida non solo nella selta di una tecnica rispetto ad un altra ma, più importante, nella progettazione di tecnologie per la cyber-sicurazza le quali non fanno compromessi sulla loro efficacia nella protezione di cyber attacchi. Gli studi sopra riportati

tentano di analizzare i sistemi e quelle che sono le good practice per mitigare le minaccie informatiche. In Chang et al Chang et al. [2013] abbiamo uno studio che riguarda lo stato dell'arte degli attacchi web basati sui malware e come difendersi da essi. Il paper inizia co uno studio del modello di attacco e le vulnerabilità che permettono questi attacchi, poi passa con l'analizzare lo stato corrente del problema dei malware ed infine investiga su quali sono i meccanismi di difesa. Come risultato, il paper da tre categoria di approcci in modo da analizzare, identificare e difendersi contro il problema dei malware web-based.Un altro

approccio ancora è presentato nel lavoro di Xu et alXu et al. [2013]. Lautore analizza il traffico nello strato di rete e nello strato di applicazione simultaneamente in modo da rilevare le web application malevole a run-time. Gli approcci che sono correntemente disponibili per rilevare siti web malevoli possono essere classificati in due categorie: approcci statici e approcci dinamici. I primi approcci analizzano le URL e i contenuti mentre gli altri usano honeypots client per analizzare comportamenti a run-time. L'esperimento con questo approccio ha mostrato che un rilevamento multi-lvello può ottenere la stessa efficacia di rilevamento come l'approccio dinamico, anche se quello dinamico si è mostrato essere molto più veloce.

3.4 MARK-V nello stato dell'arte

A differenza delle tecniche appena esaminate, MARK-V, ovvero la tecnica proposta in questa tesi si pone l'obiettivo di riuscire a classificare siti web appartenenti al Deep e Dark-Web concentrandosi di più su un approccio di *pattern matching*. Nei precedenti lavori si è data molta enfasi sul contenuto testuale per se, presente sulle pagine web ottenute mediante un web-crawler e sulla base di queste parole, in congiunta di altre parole prelevate da leggi e regolamenti o da target già ben definiti si è proseguito ad effettuare il procedimento di classificazione. Con MARK-V si tenta invece di estrapolare attraverso una rappresentazione vettoriale basata sui Topic latenti estratti grazie ad LDA quella che può essere considerata l'impronta digitale della pagina web esaminata. E' stato fatto ciò per dimostrare la congettura

che pagine web inerenti ad uno stesso argomento potessero condividere una distribuzione di probabilità in termini di Topic, simile.

3.5 Progetto ANITA

Il progetto ANITA ANI nasce per fornire alle forze dell'ordine un insieme di strumenti per cercare di contrastare attività criminali che si perpetrano grazie all'anonimato garantito dal Deep/Dark-web. In questi substrati dell'Internet visibile che tutti conosciamo, infatti, avvengono continuamente delle transazioni adoperanti cryptovalute, per esempio Bitcoin, che hanno come oggetto armi, droga, traffico di medicinali falsi, moneta contraffatta e molto altro ancora. Nel corso del tempo, le attività criminali si sono specializzate sempre di più nel garantirsi l'anonimato e nel trovare una piattaforma che permettesse loro di poter comunicare liberamente. ANITA si occupa di monitorare costantemente i cosiddetti Marketplace, ovvero luoghi in cui diventa possibile "fare la spesa" di ogni tipo di merce illecita. Il fine è di capire quali sono i trend e i pattern comportamentali che hanno luogo in tali Marketplace, in modo da aiutare le forze dell'ordine nell'identificazione delle persone coinvolte in questi traffici.

Gli obiettivi di ANITA sono dunque molteplici. Possono però essere riassunti nei seguenti due obiettivi:

- 1. aiutare il processo di investigazione delle forze dell'ordine in modo da aumentare le loro capacità operative,
- 2. diminuire significativamente la difficoltà per nuovi ufficiali nell'addestramento e ottimizzare la curva di apprendimento, collezionando e riutilizzando conoscenza.

Questa tesi mira ad essere un contributo al progetto ANITA, andando a sfruttare le informazioni contenute nel dataset fornitomi, per riuscire a trovare modi innovativi e più performanti rispetto a quelli proposti sin ora, al fine di analizzare le caratteristiche peculiari presenti in Marketplace. Quello che ci si aspetta è trovare un insieme di features caratterizzanti, a partire dai commenti lasciati dagli acquirenti della merce contraffatta sulle pagine di ogni Vendor presente su questi Marketplace.

CAPITOLO 4

MARK-V: Una tecnica di intelligenza artificiale per il riconoscimento dei siti del deep/dark web

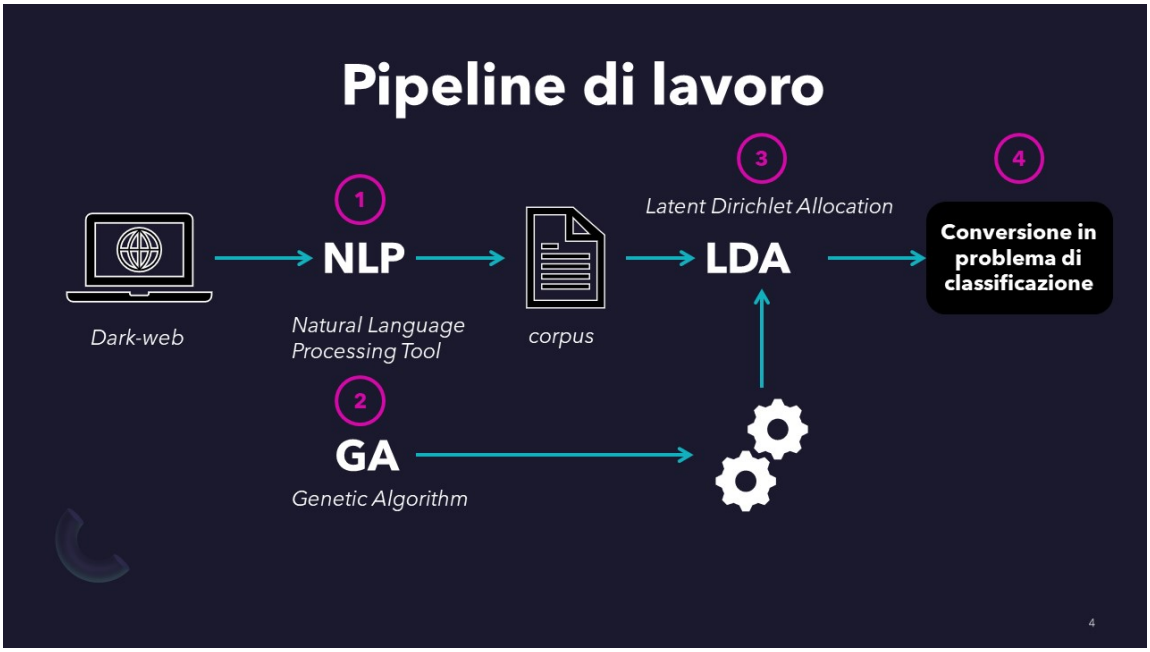

Figura 4.1: Schema della pipeline di lavoro

MARK-V costituisce il tentativo di creare una nuova pipeline di lavoro per riuscire nel problema della classificazione dei siti appartenenti al Dark/Deep-Web. Il numero dei siti risiedenti in questo sub-strato oscuro della rete è infatti in rapida espansione. Per questo motivo, cercare di contrastare questa crescita è diventata una sfida ancora aperta per la ricerca. L'approccio avanzato da MARK-V si articola su 4 punti:

1. Estrazione e Pre-processing del testo
2. Tuning degli Iper-parametri di LDA
3. Training di LDA
4. Conversione in problema di classificazione multiclasse

Nella fase 1. si punta, attraverso uno scraper costruito appositamente, di estrapolare il testo dai dump presenti nel dataset ANITA e successivamente a preprocessare il testo rimuovendo tutto ciò che non è essenziale. Verranno rimossi segni di punteggiatura, numeri, ogni verbo verrà portato alla sua forma base, ogni parola al plurale verrà portata al singolare etc... Successivamente, una volta che si dispone del corpus, nel punto 2. Troviamo quali sono i parametri sub-ottimali per allenare LDA. I parametri sub-ottimali vengono trovati grazie all'uso di un algoritmo genetico i quali considera ogni configurazione come un individuo di una popolazione che evolve fino ad un certo punto, restituendo così, una volta raggiunto il criterio di stopping condition la configurazione che ha dato prova di essere la migliore. trovata la configurazione più congeniale alleniamo nel punto 3. LDA sul corpus. Infine, nel punto 4. Una volta ottenuto il modello allenato e il corpus preprocessato, sfrutto LDA per esprimere ogni documento sottoforma di vettori n-dimensionali. Ogni componente di un vettore, rappresenta la probabilità che quel documento possa essere classificato sotto un certo Topic. Quello che viene fatto è sostanzialmente feature engineering al fine da poter ricavare delle feature da poter accoppiare a delle label per poter fare inferenza tramite un algoritmo di classificazione multi-classe, come può essere una regressione logistica, random forest o come nel mio caso un albero decisionale.

4.1 Introduzione

Avendo a disposizione i dump di diversi Marketplace del Dark/Deep-Web, forniti dal dataset ANITA, il primo passo da effettuare, per dirigerci verso l'utilizzo di LDA-GA è estrapolare il testo da questi dump. Quest'ultimo costituirà il corpus sul quale andremo ad

effettuare prima tutti i passi espressi nella pipeline di pre-processing e che poi daremo in pasto ad LDA, opportunamente calibrato, grazie all'utilizzo di un algoritmo genetico.

Il problema che è immediatamente emerso è come effettuare lo scraping delle sole recensioni, non potendo fare affidamento al modus operandi standard di quest'ultimo, ovvero quello di ricercare un pattern ricorrente nella pagina HTML. Tale procedura viene infatti usata per istruire lo scraper, permettendogli di reperire un sottoinsieme di contenuti presenti in determinati elementi della pagina HTML.

Piuttosto che affidarmi alla struttura HTML della pagina sotto esame sono partito col tentare di trovare dapprima un "discriminante" un *fil rouge* che accumunasse tutti i commenti presenti sui dump a mia disposizione. Inizialmente è stato osservato, mediante lo strumento "ispeziona" di Google Chrome, che le recensioni degli utenti sono sempre racchiuse tra delle virgolette (" "). Dunque si era ipotizzato l'uso di approcci quali regular expression, patter visitors, con le virgolette come discriminante per individuare le recensioni. Questi approcci si sono rivelati subito inadatti in quanto le virgolette non erano effettivamente presenti all'interno dell'HTML, ma costituivano soltanto un artefatto inserito dallo strumento ispeziona di Google Chrome (con amara sorpresa).

Non potendo fare affidamento alla struttura dell'HTML, in quanto diversa da dump a dump, e non potendo usare un discriminante in particolare, come le suddette virgolette, sono dovuto andare alla ricerca di un modo più generico per estrarre soltanto i commenti da questi dump. Ho costruito uno scraper.

4.2 Preprocessamento

4.2.1 preProcessing_Pipeline.py

Questo .py ha l'obiettivo di analizzare ogni singolo dump presente nel dataset ANITA, effettuare delle operazioni di estrazione e preprocessamento del testo e di salvare il tutto su disco tramite la funzione *pickle* in modo da non dover ripetere la fase di preprocessamento ogni volta. Effettuare il preprocessamento una volta soltanto ci permette di avere sempre un corpus già pronto su cui poter effettuare tutte le successive operazioni di training per LDA, velocizzando eventuali operazioni intermedie di testing, debugging e calibrazione di LDA.

La prima cosa che faccio è andare ad indicare qual è il percorso dove sono presenti i dump da reperire

```
#Prendo tutti i percorsi dei file di ratings a partire dalla root del dataset
RatingsPaths=getRatingsPaths("E:\ANITA-Dumps-Uncompressed")
```

In questo stralcio di codice faccio uso della funzione *getRatingsPaths*

```
def getRatingsPaths(DatasetFolder):

    Vendors_folders = []
    for root, subdirs, files in os.walk(DatasetFolder):
        for d in subdirs:
            if d == "vendor":
                Vendors_folders.append(os.path.join(root, d))

    Vendors_names = []
    for VendorPath in Vendors_folders:
        p = os.listdir(VendorPath)
        for file in p:
            if os.path.isdir(os.path.join(VendorPath, file)):
                Vendors_names.append(os.path.join(VendorPath, file))

    Final_Paths = []
    for Vendor in Vendors_names:
        if(((os.listdir(Vendor)).pop(0)).endswith(".html")):
            Final_Paths.append(os.path.join(Vendor, (os.listdir(Vendor)).pop(0)))
        else:
            Final_Paths.append(os.path.join(Vendor, (os.listdir(Vendor)).pop(1)))
    return Final_Paths
```

Questa funzione mi permette di andare a reperire tutti i percorsi inerenti alle recensioni lasciate dai clienti sulla pagina personale di ciascun vendor. Per riuscire a fare ciò, prendo tutti i percorsi di tutti i vendor contenuti nel dataset. Ogni cartella diversa rappresenta un vendor, il cui nome della cartella corrisponde al nome del vendor. Infine mi creo i percorsi che mi conducono fino ai *"rating.html"* che mi permettono, in fase di estrazione del testo, di andare a reperire i dump dove sono contenute le recensioni dei clienti.

Per riuscire adesso ad estrarre il testo ed effettuare le operazioni di scraping mi sono avvalso della libreria BeautifulSoup4 Richardson [2007]. Per prima cosa dunque, converto ogni file rating.html in un oggetto BeautifulSoup e lo salvo in una lista di oggetti BeautifulSoup che ho chiamato *SoupObjectsList*

```
SoupObjectsList=[]
for rating in RatingsPaths:
```

```
    f=open(rating, 'r',encoding='utf8')
    SoupObjectsList.append(BeautifulSoup(f, features="html.parser"))
    f.close()
```

Per un double check, controllo che il numero dei dump relativi alle recensioni che sono riuscito a reperire sia uguale al numero di vendor che sono stati trovati in fase di fetch dei percorsi.

```
if(len(SoupObjectsList)==len(RatingsPaths)):
    print("Sono stati rilevati "+ str(len(SoupObjectsList))+" venditori")
else:
    print("qualcosa andato storto nella conversione dei file di rating in
    BeautifulSoup objects")
```

Arrivati a questo punto ho tutto il necessario. Dispongo infatti di tutti i dump di tutte le pagine relative alle recensioni lasciate dai clienti nei confronti di ciascun vendor opportunamente convertite in oggetti BeautifulSoup da poter manipolare. Dunque, per ogni oggetto BeautifulSoup, vado ad eseguire tutti i passi della pipeline. Il testo preprocessato di ogni dump verrà salvato nella lista LISTA_CORPUS_PREPROCESSATI

4.2.2 Step 1 - estrazione del testo

Per reperire il testo dalla pagina dei feedback è stato necessario realizzare uno scraper apposito che, sulla base di alcune considerazioni che verranno elencate di seguito in questa tesi, è in grado di funzionare a partire da qualsiasi dump .html fornitogli. Un approccio di questo tipo è leggermente diverso dallo scraping classico. Nello scraping classico, quello che si faceva era andare ad esaminare la struttura HTML, quindi il suo DOM, per capire se esisteva un pattern o un percorso, dal nodo radice rappresentato dal tag <html>, al fine di individuare l'elemento contenente il testo della recensione da estrarre. In questo modo si poteva costruire uno scraper cucito su misura per la pagina HTML, contenente le informazioni di interesse da estrarre, ottenendo soltanto queste ultime senza ulteriori elementi di disturbo per le fasi successive di analisi. Nel mio caso però, il primo step è costituito dal reperimento delle recensioni **da ogni** dump di Marketplace a disposizione nel dataset ANITA. Questo ha fatto sì che un approccio di analisi della struttura del DOM HTML fosse fallimentare, in quanto è stato osservato che ogni pagina contenente le recensioni degli utenti ha una struttura diversa dall'altra. Questo ha reso molto difficile, se non impossibile, con gli strumenti attualmente a mia disposizione, automatizzare lo scraping sulla base dell'individuazione di un opportuno pattern che mi consentisse di individuare solo il testo dei singoli commenti, tra le pagine

a disposizione. Un primo tentativo per riuscire ad estrapolare il testo in modo automatico, senza legarsi ad elementi strutturali di una singola pagina web, è stato quello della ricerca di un discriminante. Si trattava di trovare un qualche elemento non legato alla struttura della pagina HTML ma alla recensione da estrarre, tramite il quale avrei avuto sempre la certezza di individuare la recensione all'interno della pagina, individuando prima il discriminante. Ispezionando manualmente tramite lo strumento "ispeziona" di Google Chrome la struttura dei diversi Marketplace di cui dispongo i dump, si era osservato che, a prescindere dal DOM della pagina in questione, i commenti erano sempre racchiusi tra " ". In primo luogo, questa è sembrata la soluzione al problema, ma successivamente, durante l'effettiva scrittura della funzione dedita allo scraping del testo del file .html, ci si è resi conto che queste " " erano presenti soltanto durante l'utilizzo dello strumento "ispeziona" e non all'interno della struttura del sito web. Questo ha portato di conseguenza ad un approccio più informale e meno deterministico alla costruzione del tool di scraping.

Per prima cosa, è stato necessario recuperare tutto il testo degli elementi HTML presente nei singoli file .html legati a recensioni lasciate dagli utenti nei diversi Marketplace. Ho aperto ogni file HTML, e tramite l'utilizzo della funzione BeautifulSoup fornita dalla libreria BeautifulSoup 4, ho convertito questo file .html in un oggetto BeautifulSoup, il quale mi permette di poter lavorare direttamente sul DOM estratto a partire da questa pagina html. Per l'estrazione del testo è stata creata una funzione di nome extractText, la quale prende come parametro un oggetto BeautifulSoup che estrapola tutto il testo presente nella pagina html, escludendo tutti i tag ed eventuali script o stili per la pagina.

```
1  def extractText(soup):
2      # kill all script and style elements
3      for script in soup(["script", "style"]):
4          script.extract()    # rip it out
5      # get text
6      text = soup.get_text()
7      # break into lines and remove leading and trailing space on each
8      lines = (line.strip() for line in text.splitlines())
9      # break multi-headlines into a line each
10     chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
11     # drop blank lines
12     text = '\n'.join(chunk for chunk in chunks if chunk)
```

Listing 4.1: - Funzione per l'estrazione del testo da ogni elemento HTML presente nell'oggetto soup passato come parametro.

Il testo estratto è rappresentato da un'unica stringa enorme, contenente sia le informazioni che ci occorrono, ovvero le recensioni, e sia parole che generano solo rumore per le successive fasi di analisi. Parole di questo genere sono, per esempio, legate a quegli elementi che sono sempre presenti in tutte le pagine, essendo elementi strutturali della pagina web stessa. Un esempio è la parola "rating", dove l'utente va ad esprimere il suo grado di soddisfacimento a seguito di un ordine. Questo rende difficile andare ad estrarre il testo legato alle sole recensioni degli utenti, a partire dal testo grezzo. Per risolvere questo problema, verranno messe in atto delle considerazioni che vedremo nelle fasi successive.

4.2.3 Step 2 - Splitting e lowercasing

Questa fase riceve in input un'unica stringa enorme, contenente tutto il testo che è stato possibile estrarre dalle pagine HTML. Adesso, per rendere questa stringa maneggevole per le successive fasi di pre-processing, è stata divisa in stringhe più piccole. Nello specifico, è stata creata una lista di stringhe, dove ogni stringa corrisponde ad una linea della stringa originaria, fornita in input. Questo viene realizzato chiamando la funzione splitAndLower, alla quale passo il corpus estratto nello step 1. In questa funzione viene creata una lista, dove andrò ad inserire diverse stringhe corrispondenti ad ogni linea della lista passata come parametro, preventivamente portata in minuscolo attraverso l'operatore lower( ). Portare tutto il testo in minuscolo si rivela essere una operazione fondamentale, in quanto permette di eliminare la possibilità di considerare una parola scritta tutta in minuscolo e la stessa parola con la prima lettera in maiuscolo, come due parole diverse. E.g. business e Business sono la stessa parola e devono essere considerate tali. Grazie alla funzione lower() riusciamo ad eliminare questa ambiguità. La funzione, una volta creata questa lista di stringhe portate in minuscolo, la ritorna al chiamante.

```
1 def slpitAndLower(corpus):
2     splitList=[]
3     splitList=txToList.split("\n")
4 
5     filteredCorpus=[]
6     for document in splitList:
7         filteredCorpus.append(document.lower())
8         return filteredCorpus
```

Listing 4.2: - Funzione per effettuare lo splitting del testo al fine di portarlo in minuscolo.

4.2.4 Step 3 - Tokenizzazione

Dalla fase precedente, riceviamo come input una lista di stringhe, dove ogni stringa corrisponde ad una riga del testo originale estratto dallo scraper, portata in minuscolo. Adesso, quello che bisogna fare è:

- tokenizzare le stringhe
- deaccentare le parole
- eliminare Token troppo corti o troppo lunghi
- eliminare liste vuote

Tokenizzare una stringa significa trasformarla in una lista di stringhe. Ogni stringa contenuta in questa lista è una singola parola della stringa originaria. La tokenizzazione è importante perché permette di poter operare sulle singole parole in modo molto più facile.

Per tokenizzare le parole ho creato la funzione TokenizingCorpus, la quale riceve come parametro il corpus da tokenizzare. In questa funzione mi sono avvalso della funzione di Gensim chiamata simple_preprocess, la quale effettua la tokenizzazione di ogni stringa, producendo una lista per ognuna, contenente i token. Questa funzione di Gensim, inoltre, mi permette anche di deaccentare le parole, durante la creazione dei Token, elimindo quelli con lunghezza minore di 4 caratteri e maggiore di 15. Ho effettuato questo filtraggio sui Token in quanto statisticamente tutte quelle parole che sono più corte di 4 caratteri sono parole legate a diversi Slang dell'inglese, ed in generale non trasportano alcuna informazione rilevante. Mentre le parole più lunghe di 15 caratteri sono dovute sicuramente a stringhe non legate a contenuti inerenti al parlato, per cui le rimuoviamo, togliendo un po' di rumore dal corpus originario, velocizzando questa fase di preprocessing. Una volta rimossi tutti i Token di lunghezza minore di 4 caratteri e maggiore di 15, ovviamente, si andranno a creare delle liste vuote. Questo è dovuto a tutte quelle liste che contenevano soltanto stringhe di dimensioni maggiori di 15 o inferiori di 4. Questa funzione, dunque, prima di concludere e restituire la lista delle liste di strighe, si assicura di eliminare ogni lista vuota che non ci è di alcun aiuto o interesse. Questo procedimento è fondamentale per avvicinarci sempre di più al nostro obiettivo principale, ovvero ottenere una rappresentazione del corpus in uno spazio vettoriale, sfruttando la rappresentazione Bag-of-words o TF-IDF.

Applicando questa funzione, avremo come risultato una lista di liste di stringhe dove non saranno presenti parole accentate e dove ogni parola costituisce una stringa a sé, contenuta ognuna in una opportuna lista rappresentante la frase di appartenenza.

```
def TokenizingCorpus(corpus):
new_corpus=[]
for document in corpus:
    new_corpus.append(gensim.utils.simple_preprocess(document,deacc=True,min_len
    =4max_len=15))

    #rimuovo le liste vuote che non servono a niente ed interferiscono con i
    calcoli
    TextExtracted = [x for x in new_corpus if x!=[]]
    return TextExtracted
```

Listing 4.3: - Funzione per effettuare la Tokenizzazione del testo.

4.2.5 Step 4 - Rimozione documenti corti

In questa fase creo una nuova lista nella quale tengo traccia soltanto dei documenti che contengono almeno 4 parole. Ho applicato l'osservazione che tutte le righe che hanno meno di 4 parole hanno un'alta probabilità di essere delle righe inutili. Diventa necessario dunque rimuovere queste righe in modo da riuscire ad estrarre il testo legato alle sole recensioni. Sono riuscito a fare ciò grazie alla funzione *delUnder4Words*.

```
def delUnder4Words(corpus):
    prettyList = []
    for document in corpus:
        if len(document) > 3:
            prettyList.append(document)
    return prettyList
```

4.2.6 Step 5 - Rimozione della punteggiatura

La punteggiatura cosi come i numeri è stato osservato non apportare informazioni critiche e di reale interesse, ed è per questo che durante questa fase mi occupo di eliminare tutto ciò. Per riuscirci mi sono avvalso della funzione *noPuncNoNumb*.

```
def noPuncNoNumb(corpus):
    List_No_punct_numb = [[strip_punctuation(stringa) and strip_numeric(stringa)
    for stringa in group] for group in
                            corpus]
    return List_No_punct_numb
```

4.2.7 Step 6 - Correzione dello spelling

Arrivati a questo punto, ho un testo che incomincia ad essere abbastanza pulito da poter essere soggetto ad una spelling correction. Spesso il contenuto testuale lasciato da un utente su un sito di e-commerce contiene degli errori grammaticali. Diventa cruciale quindi correggere tali errori ove possibile in modo da permettere successivamente ad LDA di poter capire che due parole apparentemente diverse in realtà sono la stessa parola e quindi esprimere performance migliori evitando ridondanze introdotte da tali errori. Questa fase è stata implementata dalla funzione *errataCorrige* la quale utilizza la libreria *TextBlob* e nello specifico la funzione *correct* di TextBlob.

```
1 def erratacorrige(corpus):
2     List_spellChecked = [[str(TextBlob(text).correct()) for text in document]for
      document in corpus]
3     return List_spellChecked
```

4.2.8 Step 7 - Espansione delle parole contratte

Allo stesso modo facciamo si che le parole presenti in forma contratta vengano portate alla forma non contratta. Il nostro obiettivo ultimo è ricondurre il testo nella forma più generale possibile. Infatti vedremo come gli step successivi non fanno nient'altro che avvicinarci sempre di più a questo obiettivo. Questo step è implementato dalla funzione *deContract* la quale utilizza a sua volta la funzione *expandContractions*. Tale funziona prende in considerazione una lista di parole contratte con relativa forma non contratta.

```
1 def deContract(corpus):
2     listExpanded = [[expandContractions(text) for text in document]for document
      in corpus]
3     # print("listExpanded")
4     # print(listExpanded)
5     return listExpanded
```

```
1 def expandContractions(text, c_re=c_re):
2     def replace(match):
3         return contractions[match.group(0)]
4     return c_re.sub(replace, text)
```

4.2.9 Step 8 - Singolarizzazione e lemmatizzazione

Proseguendo nella pipeline di preprocessamento del testo arriviamo a questa fase dove portiamo ogni parola espressa al plurale nella sua forma al singolare ed inoltre portiamo ogni verbo alla sua forma base. Questo step è stato implementato dalla funzione *singAndLemm*.

```
1 def singAndLemm(corpus):
2     newList = [[ (TextBlob(text)).words[0].singularize() for text in document]for
       document in corpus]
3     return lemmatize(newList)
```

Questa funzione come similmente fatto da alcune funzioni viste in precedenza, utilizza la versatile libreria TextBlob la quale tramite la funzione *singularize* permette di portare al singolare ogni parola. Per la lemmatizzazione invece mi avvalgo della funzione *lemmatize*.

```
1 def lemmatize(corpus):
2     newList = [[text.lemmatize("v") for text in document] for document in corpus]
3     return newList
```

nota:La lemmatizzazione di textBlob richiede di indicare se deve lemmatizzare un verbo o un sostantivo. dato che non ho previsto una fase di POS, ho deciso di lemmatizzare soltanto i verbi in quanto sembrano essere quelli che hanno più necessità e che racchiudono il significato maggiore.

4.2.10 Step 9 - Rimozione delle stopWords

L'ultimo step della fase di preprocessamento del testo è la fase di rimozione delle stopWords. Le stopWords sono essenzialmente delle parole che per ogni lingua vengono ripetute spesso e che non apportano nessun contenuto informativo e che di conseguenza è sicuro liberarsene. Ho deciso di effettuare questo step per ultimo in quanto se avessi tolto le stopWords in uno dei passi iniziali di preprocessing avrei rischiato di ottenere delle righe con meno di 4 parole facendo si che potessi perdere delle righe potenzialmente utili. Dato che le risorse testuali di partenza erano già scarse, si è reso necessario mettere in atto una politica atta a preservare ogni parola possibile.

4.3 Training di LDA

4.3.1 main.py

4.3.2 Operazioni preliminari

Il primo passo da fare dunque è quello di individuare il corpus precedentemente elaborato attraverso la pipline di pre-processing e di caricarlo in *LISTA_CORPUS_PREPROCESSATI*.

```
with open ('Serialized_Processed_Corpora', 'rb') as fp:
    LISTA\_CORPUS\_PREPROCESSATI = pickle.load(fp)
```

Salvare il lavoro svolto durante la fase di pre-processing mi ha permesso di poter velocizzare le fasi successive evitando i lunghi tempi di attesa che sono stati necessari attendere affinchè tutte le risorse testuali venivano estratte e processate, ad ogni run del progetto. In seguito, dato che LISTA_CORPUS_PREPROCESSATI potrebbe contenere qualche lista vuota, mi assicuro di eliminarle preventivamente

```
Corpora = [x for x in LISTA_CORPUS_PREPROCESSATI if x]
```

4.3.3 Creazione del dizionario

*scrivere cosa è e come è fatto il dizionario.

Creo un dizionario a partire dal corpora reperito in precedenza. Tale dizionario contiene la frequenza di ogni parola.

```
dictionary= gensim.corpora.Dictionary(Corpora)
count = 0
for k, v in dictionary.iteritems():
        #print(k,v)
        count +=1
```

4.3.4 Filtro dei Token

Giunti a questo punto filtro i Token che appaiono in meno di 15 documenti (numero assoluto) o in più del 50% dei documenti. Dopo i primi due step, mantengo soltanto i 600 token più frequenti. 600 è un parametro arbitrario, ma è stato scelto per tentare di mantenere quante più parole possibili dato che in totale sono state rilevate circa 700 parole uniche.

```
dictionary.filter_extremes(no_below=15,no_above=0.5,keep_n=600)
```

4.3.5 creazione Bag of Words

Siamo giunti dunque ad un punto importante, infatti abbiamo a disposizione una serie di parole estratte e preprocessate, le quali costituiranno la base su cui baseremo il training di LDA. c'è però un problema. Non possiamo semplicemente fornire delle tringhe ad un modello di machine learning e aspettarci una risposta. Dobbiamo prima processare queste stringhe andandole a convertire in un formato numerico, maneggiabile da un computer. Il modello Bag-of-Word (BoW) e Term Frequency - Inverse Document Frequency (TF-IDF) ci permettono di fare questo.
In questo progetto è stato adottato il modello Bag Of Words in quanto LDA non necessita di conoscere l'importanza di una parola, ma ha bisogno di effettuare dei conteggi sulle frequenze per performare i suoi calcoli. In questa fase dunque converto ogni documento nella sua rappresentazione Bag of Words sfruttando il dizionario creato in precedenza. Questo è importante perché i computer o meglio, i modelli matematici utilizzati nel contesto di algoritmi di Machine Learning, per poter funzionare hanno bisogno di basarsi su dei numeri. Tramite la rappresentazione Bag of Words andiamo a rappresentare ogni documento come un vettore ad N componenti dove ogni numero corrisponde alla chiave della parole presente nel dizionario che è stato creato in precedenza. In tal modo riusciamo ad esprimere un documento con soltanto dei numeri e allo stesso tempo non perdiamo traccia del contenuto del documento che risulta così sempre ricostruibile. Questa fase è implementata, così come le fasi che abbiamo visto e che vedremo, grazie alla libreria Gensim Rehurek and Sojka [2011]la quale è una libreria atta al Topic Modelling racchiudendo in se tutti i principali strumenti utili per estrarre conoscenza a partire da fonti testuali. In questa fase è stata usata la funzione *doc2bow*.

```
1 bow_corpus = [dictionary.doc2bow(doc) for doc in Corpora]
```

4.3.6 Training di LDA

Questa è la fase dove viene effettuato l'allenamento di LDA.

```
lda = LdaModel(bow_corpus,id2word=dictionary,passes=92, num_topics=28)
```

Per l'implementazione del modello ho impiegato la libreria Gensim e nello specifico *LdaModel*. Vale la pena menzionare che Gensim dispone anche del modello *LdaMulticore* il quale permette tramite la computazione parallela di velocizzare i tempi di attesa necessari per la fase di training del modello. Nel mio caso, ho avuto difficoltà ad utilizzare la versione Multicore e dunque ho eseguito il tutto sfruttando la versione non parallelizzata. A questo modello passo il dizionario creato in precedenza, tramite il paramtetro *id2word* il quale permette al modello di capire quale parola è associata ad un determinato id. Con il parametro *passes* invece vado a specificare quante volte bisogna analizzare l'intero corpus mentre con il parametro *num_topics* vado a specificare quanti Topic voglio che vengano estratti dall'intero corpus. Questi parametri sono sub-ottimali e sono stati trovati grazie all'applicazione di un algoritmo genetico. Analizzerò la ricerca dei parametri ottimali a breve. Una volta estratti i

Topic avrò una situazione del genere:

```
Topic: 0
Words: 0.282*"market" + 0.242*"real" + 0.079*"grade" ...
Topic: 1
Words: 0.684*"come" + 0.083*"week" + 0.057*"soon" ...
Topic: 2
Words: 0.294*"ship" + 0.127*"free" + 0.111*"arrive" ...
Topic: 3
Words: 0.279*"tablet" + 0.209*"bar" + 0.186*"canal" ...
Topic: 4
Words: 0.178*"cocaine" + 0.160*"price" + 0.137*"gun" ...
```

Ovvero per ogni Topic vediamo la distribuzione delle parole ritenute appartenenti a quel Topic con una certa probabilità Li. Tramite questa distribuzione di Topic ricavata in automatico da LDA possiamo classificare un documento e capire in quale Topic ricade più probabilmente tramite lo score più alto. Un esempio è il seguente:

```
docuemnto che sto cercando di classificare
['money', 'pick', 'deal', ... , 'vendor', 'come', 'acros', 'cheer']
```

```
Score: 0.41401582956314087
Topic: 0.441*"deal" + 0.101*"package" + 0.090*"leave" ...

Score: 0.254073828458786
Topic: 0.168*"like" + 0.168*"money" + 0.098*"pick" ...

Score: 0.12159229815006256
Topic: 0.684*"come" + 0.083*"week" + 0.057*"soon" ...

Score: 0.11507955193519592
Topic: 0.733*"vendor" + 0.076*"reliable" + 0.072*"trust" ...
```

In questo esempio il documento che volevamo classificare è molto verosimilmente classificabile nel Topic 16.

4.4 Tuning degli iperparametri

4.4.1 Hyperparameters_Estimation.py

E' risaputo che LDA per poter funzionare al meglio delle sue capacità ha bisogno di essere opportunamente calibrato. Data la mole molto ristretta di testo a mia disposizione sul quale sono andato ad applicare LDA, la necessità di disporre si una configurazione sub-ottima si è manifestata ancora più evidente. Piuttosto che procedere per tentativi ho voluto impiegare gli algoritmi genetici Banzhaf et al. [1998] tramite la libreria *geneticalgorithm 1.0.2* .

Ritornando a questo modulo del progetto, la prima cosa che faccio è aprire il corpora che ho serializzato su disco nella fase di pre-processing del testo affrontata nel modulo *preProcesing_Pipeline.py*.

```
1 with open ('Serialized_Processed_Corpora', 'rb') as fp:
2     LISTA_CORPUS_PREPROCESSATI = pickle.load(fp)
```

Adesso, dato che *LISTA_CORPUS_PREPROCESSATI.py* potrebbe contenere qualche lista vuota, mi assicuro di eliminarle preventivamente in quanto intralcerebbero la ricerca della configurazione sub-ottima per LDA.

```
1 Corpora = [x for x in LISTA_CORPUS_PREPROCESSATI if x]
2 print("Corpora su cui applichero' il Bag of words")
```

```
print(Corpora)
print("\n")
```

Sostanzialmente quello che farò da qui in poi è molto simile a quanto visto in *main.py* per l'allenamento di LDA in quanto dobbiamo sostanzialmente allenare il modello diverse volte con settaggi sempre diversi. Per tali ragioni quindi, sorvolerò sui passaggi intermedi di creazione di un dizionario e conversione dei documenti in Bag Of Words in quanto sono aspetti già affrontati nei capitoli precedenti. Il punto fondamentale di questo modulo è la funzione obiettivo da ottimizzare chiamata *CalculateCoherence*.

4.4.2 Funzione da ottimizzare

```
def CalculateCoherence(X):
    x = int(X[0])
    y = int(X[1])
    print("LDA parameters:\npasses:"+str(x)+"\nnum_topics:"+str(y)+"\n")
    # Alleno il modello LDA
    lda = LdaModel(bow_corpus, id2word=dictionary, passes=x, num_topics=y)
    # Compute Coherence Score
    coherence_model_lda = CoherenceModel(model=lda, texts=Corpora, dictionary=
    dictionary, coherence='u_mass')
    coherence_lda = coherence_model_lda.get_coherence()
    print('Coherence Score: ', coherence_lda)
    return coherence_lda
```

Questa funzione prende in considerazione due parametri, x e y i quali rappresentano rispettivamente il numero associato al parametro *passes* e il numero dei Topic associato al parametro *num_topics* che bisogna definire prima della fase di training di LDA. Questi valori x e y, verranno forniti dall'implementazione dell'algoritmo genetico ed è tramite la manipolazione di questi due parametri che è possibile ottenere un modello sempre più preciso. Il calcolo della bontà del modello allenato con una certa configurazione al tempo Δ sarà dato dalla misura di coerenzaKapadia [2020]. La misura di coerenza è il valore che l'algoritmo genetico cercherà di diminuire il più possibile.

4.4.3 Settings dell'algoritmo genetico

Gli algoritmi genetici godono della possibilità di una grande personalizzazione in base al problema che si sta affrontando. Per questo motivo il primo passo da fare prima di usare un algoritmo genetico è quello di deciderne i settaggi.

```
varbound=np.array([[1,100],[2,100]])
algorithm_param = {'max_num_iteration': 5,\
                   'population_size':5,\
                   'mutation_probability':0.1,\
                   'elit_ratio': 0.01,\
                   'crossover_probability': 0.5,\
                   'parents_portion': 0.3,\
                   'crossover_type':'uniform',\
                   'max_iteration_without_improv':3}
```

Come prima cosa vado a stabilire il range dei valori assumibili dai parametri x e y in modo tale da poter circoscrivere i valori per i quali l'algoritmo genetico tenterà di ottimizzare la funzione CalculateCoherence. Ho stabilito dunque la possibilità di effettuare da 1 fino a 99 *passes* e la possibilità di utilizzare da 2 fino a 99 *num_Topics*. Successivamente troviamo il parametro *max_num_iteration*. Tramite questo parametro è possibile definire il criterio di stopping condition. In questo caso la condizione di stop è raggiunta dopo 5 iterazioni dell'algoritmo. Definisco anche la dimensione della popolazione di cui è composta una generica generazione, tramite il parametro *population_size* il quale è settato a 5. Una piccola nota. Questi parametri insieme a quelli che vedremo fra poco sono stati scelti arbitrariamente in modo tale da venire incontro alle capacità di calcolo a disposizione e cercando di rimanere con un settaggio dell'algoritmo genetico che sia quanto più generale e standard possibile. Proseguendo dunque con la spiegazione del settaggio dell'algoritmo genetico troviamo i parametri:

- *mutation_probability*, il quale determina la possibilità di ogni gene in ogni singola soluzione di essere rimpiazzato da un valore totalmente random.
- *elit_ratio*, il quale determina il numero di *elites* nella popolazone. Questo parametro è impostato di default a 0.01 il che indica che su una popolazione di 100 individui abbiamo 1 elites. Se il parametro viene impostato a 0 allora si implementa l'algoritmo genetico classico senza elitismo.
- *crossover_probability*. Esso determina la possibilità di una soluzione esistente di passare il suo genoma (caratteristiche) ai nuovi discendenti. Il valore di default è 50 percento.

- *parents_portion*. Esso determina la porzione della popolazione riempita dai membri appartenenti alla generazione precendente. Per esempio 0.3 indica che il 30 percento della popolazione attuale è rappresentata da individui appartenenti alla popolazione immediatamente precedente.

- *crossover_type*. E' stata scelto un crossover di tipo univorme piuttosto che a singolo punto o multipunto.

- *max_iteration_without_improv*. Esso costituisce un ulteriore criterio di stopping condition. Tramite questo parametro stiamo dicendo che se l'agoritmo non migliora la funzione obiettivo su un numero di 3 iterazioni, allora l'algoritmo genetico si arresta e ritorna la migliore soluzione trovata prima della iterazione numero 3. Di default questo parametro è settato a *None* ma ho preferito usarlo per evitare iterazioni troppo lunghe.

Dopo aver settato l'algoritmo genetico non resta che avviarlo passandogli la funzione obiettivo da ottimizzare.

```
Genetic_Algorithm=ga(function=CalculateCoherence,dimension=2,variable_type='int',
    variable_boundaries=varbound,algorithm_parameters=algorithm_param)
```

La soluzione la leggiamo tramite il comando:

```
solution=Genetic_Algorithm.ouput_dict
```

Un esempio di output di questa fase è:

```
Coherence Score:  -8.484003954674659
LDA parameters:
passes:7
num_topics:88

Coherence Score:  -7.423403241409457
LDA parameters:
passes:7
num_topics:34

Coherence Score:  -7.639967055687163
LDA parameters:
passes:50
num_topics:32

Coherence Score:  -8.149792159810666
LDA parameters:
passes:44
num_topics:89

Coherence Score:  -7.232784068825673
||||||||||_____________________________________________ 20.0% GA is running...LDA
    parameters:
passes:92
num_topics:33

...
```

Alla fine otteniamo i valori per *passes* e *num_Topics* che ci permettono di ottenere un valore di coerenza sub-ottimo.

4.5 Conversione in problema supervisionato

Sino ad ora abbiamo visto come a partire dal dataset ANITA composto principalmente da dump di pagine legate a marketplace presenti nel Dark/Deep-web siamo stati in grado di pre-processare il testo, di allenare il modello LDA opportunamente settato tramite GA e di ricavare i Topic latenti in ciascun documento Kelechava [2020]. Quanto ottenuto sino ad ora è un ottimo punto di partenza, ma non è abbastanza. Quello che idealmente si avrebbe voluto

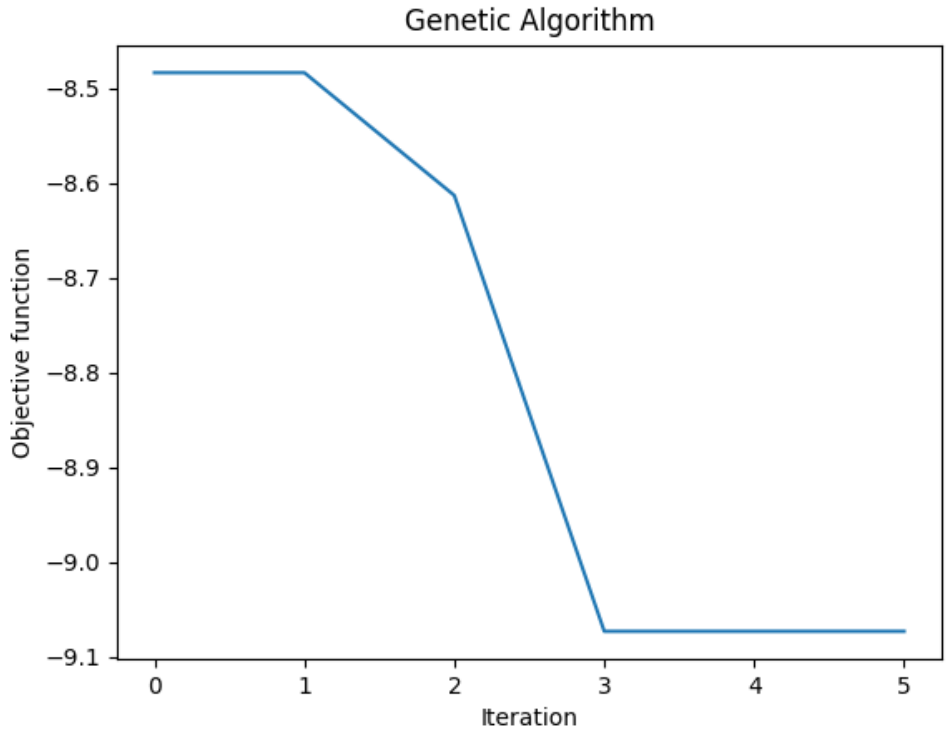

Figura 4.2: Progressi nella ricerca della soluzione sub-ottima da parte di GA

fare è, tramite LDA, trovare un modo per predire, a partire dal testo estratto da un sito web nel Deep/Dark-web il crimine perpetrato su quel sito.

4.5.1 idea di fondo

Partiamo con l'analisi di quanto abbiamo ottenuto sino ad adesso. Abbiamo un modello di Machine Learning che è in grado, a partire da un documento, di poter ricavare una *distribuzione di Topic latenti* in esso presenti. Questo significa che ogni documento può essere espresso come una distribuzione di probabilità dove ogni valore della distribuzione corrisponde alla probabilità di quel commento x di appartenere al topic 1, 2, 3, ecc... . Tenendo questo in mente dunque, quello che restava da fare era convertire ogni commento in distribuzioni di probabilità e creare così un nesso tra le distribuzioni di probabilità e le label che già disponevamo per ogni commento. Questi risultati li ho opportunamente salvati in un nuovo dataset contenenti soltanto le colonne di mio interesse, ovvero:

- Review
- Category
- Topic Distribuition

Avendo un Dataset così composto ho potuto allenare un algoritmo di Decision Tree e misurare in questo modo quanto le predizioni di questo algoritmo si siano rilevate precise, sulla base delle distribuzioni di probabilità di ciascun documento.

4.5.2 pre-processing delle recensioni

Per validare la bontà del modello ho applicato due metodologie diverse, dapprima ho diviso il dataset in due porzioni. Una porzione dedicata al training del modello ed una porzione dedicata al testing del modello, e poi ho applicato una 10-Fold Cross Validation. Essa ci consente di partizionare il dataset in k partizioni, allenare ogni volta il modello sulle k-1 partizioni e usare la k-esima partizione come partizione di validazione, e ripetere il procedimento per tutte le k partizioni.

Le fasi iniziali per pre-processare il testo delle recensioni delle applicazioni sono deltutto uguali a quello che abbiamo già affrontato durante il pre-processing del testo estratto dai dump del dataset ANITA. L'unica differenza è che essendo un dataset differente è stato necessario modificarlo, eliminando colonne che non sono interessanti per questo esperimento. Per prima cosa dunque apro il dataset grezzo così come mi è stato fornito:

```
data=pd.read_csv("reviews.csv")
```

Proseguo quindi con l'eliminazione delle colonne che non mi sono di aiuto:

```
print("\nI remove the unwanted columns 'Unnamed: 0','id', 'package_name', 'date
    ','star', 'version_id'")
data = data.drop(columns=['Unnamed: 0','id', 'package_name', 'date',
                          'star', 'version_id'], axis=1)
```

Prendo in questo modo tutte le recensioni e le metto in una lista. Queste recensioni costituiscono il testo oggetto di pre-processamento.

```
temp_data = data
temp_data = temp_data.drop(columns=['category'],axis=1)
r=temp_data.reset_index().values.tolist()
print(r)
reviews=[]
for lista in r:
    temp_list=[lista[1]]
    reviews.append(temp_list)
```

Dato che il dataset di partenza non aveva una colonna *Topic_Distribuition* l'ho creata cosicchè potessi salvare in secondo momento le distribuzioni di probabilità relative ai Topic latenti per ciascun documento/recensione.

```
data['Topic_Distribuition']=""
```

Una volta fatto ciò eseguo la pipeline di pre-processing esattamente come definita in precedenza per i dump del dataset ANITA e serializzo il testo pre-processato su disco con il nome di *Serialized_Processed_Corpora*.

A questo punto per poter effettuare gli step successivi limitando il carico di lavoro che il dispositivo sul quale stavo lavorando era costretto a subire, e per dimezzare i tempi mi sono avvalso della piattaforma *Google Colab* per effettuare della computazione in cloud.

4.5.3 TopicDistribuitions.ipynb

Questo notebook permette, preso il csv delle recensioni, di allenare LDA su di esse e di ricavarne la dstribuzione di probabilità in termini di Topic. Sulla base di questi risultati ho generato una nuova colonna nel dataset di partenza chiamata TopicDistribuition dove ho inserito per ogni recensione la corrispondente distribuzione di Topic, ed ho effettuato un drop di tutte le altre colonne che non mi servivano. Alla fine ottengo un csv chiamato FinalDataset contenente soltanto le colonne review per le recensioni, category ovvero le nostre label e TopicDistribuition.

Apro per prima cosa il corpus delle recensioni già pre-processato in locale e il Dataset con le colonne già ripulite e sistemate.

```
with open ('/content/drive/MyDrive/Colab Notebooks/Serialized_Processed_Corpora',
    'rb') as fp:
    LISTA_CORPUS_PREPROCESSATI = pickle.load(fp)
df= pd.read_csv("/content/drive/MyDrive/Colab Notebooks/reviewsincsvconExcel.csv"
    ,error_bad_lines=False, sep=';')
print(df)
```

Per quanto riguarda l'allenamento di LDA si procede come abbiamo già visto più volte, dunque non riportedò dinuovo gli stessi passaggi qui di seguito. Il punto più interessante di questo modulo è la generazione della distribuzione di probabilità per ogni recensione. Per ogni riga del csv prendo il testo della review, lo converto in BoW e lo do in pasto ad lda.get_document_topics. Tramite questo metodo riesco ad ottenere la distribuzione di probabilità in termini dei Topic per il documento i-esimo. Questa distribuzione di probabilità la vado a scrivere nella colonna TopicDistribuition. Una volta finito tutto salvo il lavoro in un nuovo csv

```
def func(row):
  return lda.get_document_topics(dictionary.doc2bow(gensim.utils.tokenize(row.
    review)), minimum_probability=0.0)

df['TopicDistribuition']=df.apply(func, axis = 1)
print("\n")
print(df)
df.to_csv("/content/drive/MyDrive/Colab Notebooks/FinalDataset.csv",index=False)
```

4.5.4 Predizione.ipynb

Questo codice prende in input un Csv che è stato ricavato opportunamente a partire da un un dataset contenente recensioni che degli utenti hanno rilasciato al momento del download di una applicazione. Su questo Csv è stato applicato del semplice preprocessing in modo da ottenere un corpus più o meno omogeneo da dare in pasto ad LDA. Una volta allenato il modello è stato usato per ricavare la distribuzione dei Topic associata ad ogni recensione. Questa distribuzione costituisce sostanzialmente la "feature" aggiuntiva del dataset su cui andiamo ad allenare il classificatore, in questo caso un albero decisionale. Questo è in sostanza l'obiettivo di questo modulo, ovvero preso il dataset così descritto e fare classificazione. Per aumentare l'accuratezza si potrebbe pensare di migliorare le fasi precedenti, sopratutto quelle relative alla pipeline di pre-processing e magari il tuning dell'algoritmo genetico.

La prima cosa che faccio è leggere il csv finale ottenuto dalle fasi precedenti

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/FinalDataset.csv",
    error_bad_lines=False, sep=',')
```

Leggo le distribuzioni di probabilità in termini di Topic ottenute per ogni review e salvo il tutto in temp.

```
temp=data.TopicDistribuition.tolist()
```

Dato che avere una lista di vettori contenenti le distribuzioni di probabilità costituisce un formato più maneggevole rispetto ad un singolo mega vettore, creo una nuova lista di liste. Ogni lista in questa grande lista è la distribuzione di probabilità di un documento.

```
x = [strToList(x) for x in temp ]
```

Faccio la stessa cosa anche per le categorie.

```
y = data.category.tolist()
```

Adesso la cosa che bisogna fare è splittare il dataset in due set, uno di training ed uno di testing. Le percentuali che sono state scelte sono 70% training e 30%testing.

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
    random_state=1) # 70% training and 30% test
```

Dopo aver fatto ciò creo un oggetto di tipo DecisionTree classifier

```
clf = DecisionTreeClassifier(criterion="gini",max_depth=10)
```

Alleno a questo punto il Decision Tree

```
clf = clf.fit(X_train,y_train)
```

Testiamo le risposte sul test set tenuto da parte in precedenza

```
y_pred = clf.predict(X_test)
```

Ed infnfine misuro l'accuratezza, ovvero quante volte il classificatore fa una predizione corretta

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

CAPITOLO 5

Valutazione Preliminare

5.1 Obiettivi dello studio empirico

L'obiettivo di questa tesi è investigare se e quanto possibile è utilizzare il solo contenuto testuale presente all'interno di siti provenienti dal Deep/Dark-Web per determinare i crimini perpetrati in esso. L'obiettivo è quello di fornire alle Law Enforcement Agency un Tool che possa aiutare loro in questa impresa, data la grande quantità di siti presenti in questo substrato della rete che tutt'ora sono ancora non classificati. Avere delle metriche tramite le quali si possa percepire la pericolosità di un sito web, costituirebbe un grande aiuto nell'individuazione di crimini in corso e di prendere immediate conseguenze.

5.1.1 Domande di ricerca

Questa tesi è guidata da 3 domande di ricerca che puntano ad esplorare il problema della classificazione di siti nel Deep/Dark-Web. La prima domanda che mi sono posto è:

> *RQ_1 Qual è l'accuratezza di un modello di Machine Learning basato su metriche testuali nella classificazione di siti del Deep e Dark web?*

Come ogni nuova tecnica che non è stata ancora estensivamente testata, nasce spontanea la domanda di quanto essa possa essere effettivamente una via percorribile. Nel corso degli anni, nello stato dell'arte si sono susseguite diverse tecniche per stabilire la pericolosità di un

sito web, come per esempio le software quality metrics intrinseche nella struttura del sito stesso come indicatore di pericolosità. Soltanto recentemente, concordemente con l'aumento esponenziale delle pagine risiedenti nel Deep/Dark-Web si è pensato di usare il testo presente in queste pagine per poter allenare un algoritmo di Machine Learning. E' doveroso però fare una precisazione. Sebbene sono diverse le tecniche nate con lo scopo di sfruttare le informazioni testuali per trarre conoscenza, esse si sono sviluppate in modo diverso le une dalle altre. Infatti l'utilizzo del testo è costituito quasi sempre soltanto il punto di partenza per poi proseguire in modi diversi. Dato che l'approccio proposto in questa tesi è leggermente diverso questo mi ha condotto ad una seconda domanda di ricerca.

RQ_2 Utilizzare la distribuzione di probabilità dei Topic latenti nel contenuto testuale di un sito per una sua classificazione, è possibile?

A differenza dell'approccio utilizzato dai miei predecessori, l'obiettivo perseguito in questa tesi è stato quello di sfruttare il testo presente nelle pagine web per ricavare a sua volta delle feature attraverso un processo di feature engineering. Nelle ricerche che abbiamo esaminato il testo è stato spesso accomunato ad informazioni aggiuntive immesse dal programmatore per tentare di effettuare una corretta classificazione. Per esempio nel lavoro di Thorat et al. [2020] il testo fu accoppiato a quelle che sono le leggi indiane in modo da individuare quali crimini fossero perpetrati su un sito web. L'appoccio invece che ho voluto tentare in questa tesi trascende dal significato vero e proprio delle parole estratte dal sito web, per andare alla ricerca di un pattern identificabile come nella distribuzione di probabilità dei topic latenti in qualsiasi testo.

RQ_3 Qual è l'accuratezza di un modello di Machine Learning basato su distribuzione di probabilità di Topic latenti nella classificazione di siti del Deep e Dark web?

Sebbene questo nuovo approccio si sia rivelato essere fattibile, non sono state poche le difficoltà per far funzionare il tutto. Il problema di un approccio di classificazione text-based è il numero di passi presenti nella pipeline di pre-processamento del testo e nella scelta delle feature da considerare per il successivo problema di classificazione. Il corretto pre-processamento è un'arte a-sè-stante che richiede una minuzia nella calibrazione di tutti i passaggi che può derivare soltanto da una conoscenza del dominio applicativo molto estesa. Diventa fondamentale dunque l'intervento umano, in un modo o nell'altro, e di

conseguenza una reale pipeline automatica per risolvere il problema della classificazione dei siti nel Deep/Dark-Web non credo sia possibile, a patto di scendere a compromessi con i risultati. La pipeline di pre-processamento è però soltanto una delle componenti della macro-pipeline di MARK-V. Infatti, ogni settore di questa pipeline necessita di una calibrazione, ed è dal corretto setup di tutte queste componenti che si potrebbe raggiungere risultati più soddisfacenti. L'accuratezza del 46% raggiunta in questa tesi dunque deve essere letta come un primo tentativo nella direzione di questo nuovo approccio ed un invito nella identificazione di quali possono essere ulteriori punti critici nella pipeline che necessitano di una riconfigurazione.

5.2 Contesto di studio e preparazione del Dataset

Nel corso del lavoro svolto in questa tesi, sono stati due i Dataset impiegati. Ho impiagato dapprima il Dataset ANITA per sviluppare la pipeline fino a giungere alla fase di classificazione vera e propria e poi il Dataset APPREVIEW per la fase di classificazione. I motivi per il quale sono stati impiegati due Dataset diversi sono molteplici. Il primo fattore che ha influenzato questa scelta, è dovuto al fatto che il Dataset ANITA non vi erano delle label che mi permettessero di operare della classificazione supervisionata. E' stato dunque necessario andare a considerare un nuovo Dataset, APPREVIEW dove invece vi era disponibile sia del testo, che delle label associate a quel testo. L'idea di fondo inoltre è stata che MARK-V, essendo una tecnica che è possibile applicare in più contesti, se avesse dato prova di essere accurata sul Dataset APPREVIEW utilizzando i commenti come testo di partenza, questo avrebbe in qualche modo portato a pensare che una simile accuratezza possa essere ottenuta anche sul Dataset ANITA. il Dataset ANITA è composto da diversi dump di Marketplace presenti nel Deep/Dark-Web ottenuti mediante un crawler, su un periodo di diversi mesi. Di ogni pagina dunque è possibile osservare come sia cambiate nel corso del tempo. Il Dataset così come mi è stato fornito non disponeva di una struttura ben definita, ed è stato mio compito andare a selezionare soltanto un sottoinsieme di questi dump per poter applicare MARK-V per la classificazione. Ho selezionato infatti soltato quei dump per i quali erano effettivamente disponibili delle recensioni sulle pagine dei venditori di merce illecita, trascurando gli altri nei quali invece non c'era nulla. Il problema di questo Dataset però è stato quello di non disporre delle label tramite il quale io potessi poggiarmi per effettuare l'ultimo step di MARK-V, ovvero quello della classificazione del sito basandomi sulle distribuzioni di probabilità dei Topic latenti. per questo motivo ho dovuto disporre di un altro Dataset,

strutturato in maniera completamente differente, ovvero APPSREVIEW. Questo Dataset si è presentato come un CSV nel quale vi erano riportate le recensioni rilasciate da utenti sulle pagine presenti su uno store di applicazioni. Per ogni recensione vi era associata la categoria dell'applicazione per la quale l'utente stava rilasciando la recensione, permettendomi di disporre dunque della combinazione test/label di cui necessitavo. Questo Dataset ha necessitato una leggera elaborazione prima di poter essere usato. Ho notato infatti che diverse label di applicazioni erano composte e non apportavano nessun contributo al Dataset. Data anche la grande inferiorità numerica di queste label rispetto ad altre molto simili, ho deciso di effettuare un drop delle righe di questi dati per pulire il Dataset. Sono stati effettuati anche dei drop ulteriori per tutte quelle colonne contenenti informazioni di cui non avevo bisogno, come l'id dei dati, il nome del package, la data, il voto dell'applicazione e la versione.

5.3 Metodologia di valutazione

Il training del modello è stato condotto prima con un approccio semplice di divisione del Dataset in test-set e training-set, per poi effettuare un training tramite 10-fold cross validation per una verifica più approfondita. Di conseguenza è stato necessario scegliere quale metrica scegliere per valutare le prestazioni dell'algoritmo. La scelta delle metriche influenza come le performance degli algoritmi di Machine Learning sono misurati e comparati. Esse influenzano come pesiamo l'importanza delle diverse caratteristiche nei risultati, e la scelta su quale algoritmo scegliere. Le metriche usate per problemi di classificazione sono essenzialmente le seguenti:

1. Accuracy
2. Log Loss
3. Area sotto la curva ROC
4. Confusion Matrix
5. Report di classificazione

Tra tutte queste metriche è stata adottata l'accuracy. L'accuracy viene definita formalmente come:

$$Accuracy = \frac{Numero\ delle\ predizioni\ corrette}{Numero\ totale\ delle\ predizioni}$$

La misura di accuracy è espressa come un numero reale tra 0 ed 1, o alternativamente in percentuali comprese tra 0% e 100%. Percentuali vicine al 100% e numeri reali vicini ad 1, rappresentano l'accuracy desiderata, in quanto indica che il modello ha un'ottima capacità predittiva.

5.4 Risultati ottenuti

5.4.1 Risultati RQ.1

Queste tecniche text-based comunque hanno dato già prova di essere efficaci, infatti troviamo che nel lavoro di Thorat et al. [2020] essi sono riusciti a raggiungere una accuracy dell'88.889% in media, utilizzando tre diversi algoritmi di classificazione, come Naive Bayes, Support Vector Machine e Random-Forest. Nel lavoro invece di Al Nabki et al. [2017] l'accuratezza è salita fino al 93.7% utilizzando una combinazione di rappresentazione delle parole in TF-IDF e un classificatore di regressione logistica.

5.4.2 Risultati RQ.2

Il processo di feature engineering mi ha portato a considerare come feature, con le quali poter allenare un algoritmo di classificazione (un decision tree) i Topic latenti che sono riuscito ad estrarre impiegando un algoritmo di classificazione testuale non supervisionato, LDA. Un'approccio dunque che si basa su tali distribuzioni di probabilità si è rivelato essere dunque praticabile.

5.4.3 Risultati RQ.3

In fase di testing è stata ottenuta l'accuracy del 46%. Questa accuracy, come già fortemente evidenziato nella RQ_3, deve essere interpretata come un invito a continuare la ricerca della giusta calibrazione di ogni modulo della pipeline di MARK-V. Vorrei inoltre aggiungere una considerazione. Il Dataset sul quale è stato tentato questo approccio di classificazione in base ai Topic latenti nel testo non è stato, per cause di forze maggiori, quello realmente oggetto della ricerca portata avanti in questa tesi. Infatti il dominio applicativo è cambiato dal dominio composto dai siti web presenti nel Deep/Dark-Web a quello delle recensioni lasciate dagli utenti su uno store di applicazioni. E' possibile che questo cambio di dominio abbia potuto inficiare i risultati ulteriormente. L'ipotesi è che i commenti su diverse applicazioni appartenenti a categorie diverse potessero somigliarsi al punto da non portare ad una

classificazione accurata. Inoltre, il linguaggio contenuto nelle recensioni rilasciate da utenti su un store di applicazioni è meno inquadrabile in un contesto specifico come potrebbe essere il linguaggio usato su un sito dove si vendono per esempio armi. In questi siti infatti, si potrebbero riscontrare delle parole inquadrabili in quello che è il contesto della vendita di armi, contesto in cui possiamo trovare un numero limitato di termini e tutti molto specifici. Di conseguenza sarebbe più facile per un algoritmo di Machine Learning creare una associazione tra un testo così composto e una eventuale label ad essa associata.

Il Dataset ANITA inoltre ha dimostrato di avere una grossa limitazione. Ogni dump non ha associata una label che indicasse che tipo di crimine si stesse perpetrando correntemente su quel determinato sito. Questo ha portato il progetto a considerare un altro Dataset appartenente ad un altro progetto. E' stato possibile fare ciò in quanto la tecnica innovativa oggetto di questa tesi non è legata alla provenienza del testo di partenza, sia essa proveniente da un sito web oppure da recensioni lasciate su uno store di applicazioni. Quello che si vuole dimostrare infatti è l'esistenza di una relazione tra i Topic latenti in un testo e la label per la quale quel determinato testo è stato classifiato. Il progetto in questione da cui è stato preso in prestito il Dataset è *AppReview* Palomba et al. [2017] nel quale sono state racimolate migliaia di recensioni che gli utenti hanno rilasciato sulle pagine di diverse applicazioni in uno store di app. Dato che ogni recensione è legata ad un'applicazione appartenente ad una certa categoria ho potuto disporre di un testo e di una label per ogni recensione, grazie a questo Dataset.

CAPITOLO 6

Conclusioni

Dal lavoro di tesi e dalle analisi effettuate, è emerso che un approccio di Cyber Threat Intelligence basato sul solo testo, è una via plausibile. Al contrario di altre tecniche, come quelle basate sulle software quality metrics, un approccio basato sul testo richiede una calibrazione molto più capillare per poter funzionare al meglio. Questo di conseguenza comporta un'adattabilità della tecnica molto alta, ma con il rovescio della medaglia di correre un rischio più concreto nello scegliere dei settaggi non ottimali. Una possibile soluzione dunque per sfruttare questo approccio al massimo è avare una conoscenza del dominio applicativo maggiore, necessaria per poter estrarre e preprocessare il testo. Dal lavoro svolto in questa tesi è emersa una correlazione tra un documento, la distribuzione di probabilità dei Topic attraverso la quale il documento può essere espresso e una "categoria" tramite la quale possiamo classificare il documento. In questa tesi sono state incontrate delle limitazioni nel dataset (ANITA) fornitomi in partenza, in quanto i dump delle pagine dei marketplace non disponevano di una label. Questo mi ha forzato a considerare un nuovo dominio applicativo, ovvero quello delle recensioni di utenti rilasciate sulle pagine di applicazioni scaricate da uno store. Come possibili sviluppi futuri sarebbe interessante applicare questa tecnica a dei blog i quali per natura dispongono già di una struttura a "Topic" e di una quantità di testo maggiore, e sarebbe interessante applicare questa tecnica ad un dataset creato a partire da questi blog, provvisto di label. Non si può non menzionare tutti gli sviluppi che è possibile portare avanti in merito alla correzione o aggiunta di moduli alla pipeline di pre-processing del testo, in quanto è quella la fase cruciale dell'intero processo. La qualità delle predizioni di

LDA infatti è direttamente proporzionale a quanto pulito e "informativo" è il testo datogli in pasto. Infine proporrei di provare ad usare diversi algoritmi di apprendimento per verificare quali tra questi produce un'accuratezza maggiore. In questa testi è stato sperimentato un albero decisionale in quanto mi sono trovato difronte ad un problema di classificazione multi-classe, ma si potrebbero provare anche Random Forest e altri sistemi di classificazione multi-classe, sia one vs one e sia one vs rest.

Bibliografia

[1] ANITA Project. URL https://www.anita-project.eu/. (Citato a pagina 24)

[2] Mhd Wesam Al Nabki, Eduardo Fidalgo, Enrique Alegre, and Ivan de Paz. Classifying illegal activities on tor network based on web textual contents. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 35–43, 2017. (Citato alle pagine 21 e 55)

[3] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. ISBN 155860510X. (Citato a pagina 40)

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003. (Citato a pagina 5)

[5] Giuseppe Cascavilla, Damian A Tamburri, and Willem-Jan Van Den Heuvel. Cybercrime threat intelligence: a systematic multi-vocal literature review. *Computers & Security*, page 102258, 2021. (Citato a pagina 20)

[6] Jian Chang, Krishna K. Venkatasubramanian, Andrew G. West, and Insup Lee. Analyzing and defending against web-based malware. *ACM Comput. Surv.*, 45(4):49:1–49:35, 2013. doi: 10.1145/2501654.2501663. URL https://doi.org/10.1145/2501654.2501663. (Citato a pagina 23)

[7] Fabrizio De Fausti, Francesco Pugliese, and Diego Zardetto. Toward automated website classification by deep learning. *arXiv preprint arXiv:1910.09991*, 2019. (Citato a pagina 22)

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[9] Siyu He, Yongzhong He, and Mingzhe Li. Classification of illegal activities on the dark web. In *Proceedings of the 2019 2nd International Conference on Information Science and Systems*, pages 73–78, 2019.

[10] Shashank Kapadia. Evaluate topic models: Latent dirichlet allocation (lda), Dec 2020. URL `https://towardsdatascience.com/evaluate-topic-model-in-python%20latent-dirichlet-allocation-lda-7d57484bb5d0`. (Citato a pagina 41)

[11] Marc Kelechava. Using LDA Topic Models as a Classification Model Input, 08 2020. URL `https://towardsdatascience.com/unsupervised-nlp /-topic-models-as-a-supervised-learning-input-cf8ee9e5cf28`. (Citato a pagina 44)

[12] Susan Li. Evaluate topic models: Latent dirichlet allocation (lda). `https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-/9bf156893c24`. (Citato a pagina 39)

[13] Steven Loria. textblob documentation. *Release 0.15*, 2, 2018.

[14] Fabio Palomba, Pasquale Salza, Adelina Ciurumelea, Sebastiano Panichella, Harald Gall, Filomena Ferrucci, and Andrea De Lucia. Recommending and localizing change requests for mobile apps based on user reviews. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 106–117, 2017. doi: 10.1109/ICSE.2017.18. (Citato a pagina 56)

[15] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011. (Citato a pagina 38)

[18] Leonard Richardson. Beautiful soup documentation. *April*, 2007. (Citato a pagina 28)

[19] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[20] Ewan Klein Steven Bird, Edward Loper. NLTK 3.6.2 documentation, 2021. URL `https://www.nltk.org/`.

[21] Hrushikesh Thorat, Shubham Thakur, and Amit Yadav. Categorization of illegal activities on dark web using classification. 2020. (Citato alle pagine 21, 52 e 55)

[22] Eran Toch, Claudio Bettini, Erez Shmueli, Laura Radaelli, Andrea Lanzi, Daniele Riboni, and Bruno Lepri. The privacy implications of cyber security systems: A technological survey. *ACM Comput. Surv.*, 51(2), February 2018. ISSN 0360-0300. doi: 10.1145/3172869. URL `https://doi.org/10.1145/3172869`. (Citato a pagina 22)

[23] Wiem Tounsi and Helmi Rais. A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Comput. Secur.*, 72:212–233, 2018. doi: 10.1016/j.cose.2017.09.001. URL `https://doi.org/10.1016/j.cose.2017.09.001`. (Citato a pagina 22)

[24] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. Cross-layer detection of malicious websites. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, page 141–152, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318907. doi: 10.1145/2435349.2435366. URL `https://doi.org/10.1145/2435349.2435366`. (Citato a pagina 23)