Università degli Studi di Salerno



DIPARTIMENTO DI INFORMATICA

**CORSO DI LAUREA IN INFORMATICA**

TESI DI LAUREA

Profilazione attività illecite nel Deep/Dark-web mediante LDA ed Algoritmi Genetici

**RELATORE CANDIDATO**

Prof. Fabio Palomba Emanuele Fittipaldi

Matr. N. 5816

**Anno Accademico 2020/2021**

**Indice**

[Panoramica del dataset ANITA 2](#_Toc66176382)

[Pipeline di pre-processing… 3](#_Toc66176383)

[Panoramica dell’approccio usato 4](#_Toc66176384)

[Step 1 – Reperimento del testo 6](#_Toc66176385)

[Step 2 – Splitting del testo 9](#_Toc66176386)

[Step 3 – Tokenizzazione 11](#_Toc66176387)

[Step 4 – Eliminazione documenti corti 13](#_Toc66176388)

[Step 5 – Rimozione punteggiatura e numeri 18](#_Toc66176389)

[Step 6 – Correzione dello spelling 19](#_Toc66176390)

[Step 7 – Espansione delle contrazioni 20](#_Toc66176391)

[Step 8 - Singolarizzazione delle parole 21](#_Toc66176392)

[Step 9 – Lemmatizzazione delle parole 21](#_Toc66176393)

[Step 10 – Rimozione delle Stopwords 21](#_Toc66176394)

[Valutazione dello scraper 22](#_Toc66176395)

[Statistiche estratte dal dataset ANITA 22](#_Toc66176396)

# Panoramica progetto ANITA

Il progetto ANITA 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.

# Pipeline di pre-processing.

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. In questo modo si può istruire lo scraper, permettendogli di reperire un sottoinsieme di contenuti presenti in determinati elementi.

Inizialmente è stato osservato, mediante lo strumento “ispeziona” di Google Chrome, che le recensioni degli utenti sono sempre racchiuse tra delle virgolette (“ “). Dunque si erano avanzate delle ipotesi su come poter estrarre soltanto i commenti, usando 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.

Non potendo fare affidamento alla struttura dell’HTML, diversa da dump a dump, e non potendo usare un discriminante in particolare, come le suddette virgolette, si è dovuto andare alla ricerca di un modo più generico per estrarre soltanto i commenti da questi dump.

# Panoramica dell’approccio usato

Lo scraper costruito fa parte di una pipeline di pre-processing, la quale prende in input il testo grezzo, estratto grazie allo scraper, e attraverso diversi passi mi permette di avere in output un modello con il quale posso istruire LDA, per poter fare inferenza sul testo analizzato.

Questi passi sono stati divisi in *passi principali* e *passi di raffinamento*.

**Passi principali**

L’insieme di questi passi principali costituiscono la fase di *Scraping*.

I passi che sono stati individuati sono i seguenti:

1. reperimento del testo
2. splitting del testo in righe

2.1) lowercasing delle parole

1. tokenizzazione delle righe

3.1) tokenizzazione delle stringhe

3.2) eliminazione Token troppo corti e Token troppo lunghi

3.3) rimozione dell’accento

3.4) eliminazione liste vuote

1. eliminazione dei documenti corti.

A fase di *Scraping* terminata, otteniamo il testo dei commenti lasciati dagli utenti su quel determinato Marketplace di cui è presente il dump nel dataset ANITA. Avendo, come vedremo, adottato delle scelte che hanno permesso di salvare più parole possibili, ciò ha introdotto la presenza di alcune non appartenenti ai commenti, che da qui in poi chiameremo parole *spurie*. Quest’ultime verranno eliminate durante la fase di *raffinamento*.

**Passi di raffinamento**

L’insieme di questi passi di raffinamento costituiscono la *fase di raffinamento*.

I passi che sono stati individuati sono i seguenti:

1. eliminazione della punteggiatura e dei numeri
2. correzione dello spelling
3. espansione delle contrazioni
4. singolarizzazione delle parole
5. lemmatizzazione delle parole
6. rimozione delle StopWords.

A fase di *raffinamento* terminata, otteniamo del testo senza punteggiatura e senza numeri, con spelling corretto, con la forma estesa di eventuali parole contratte (e.g. da *don’t* a *do not*) e dove tutte le parole sono state portate alla loro forma canonica e al singolare. Inoltre, in questo testo finale, non ci saranno neanche le cosiddette *stopwords*, ovvero parole usate frequentemente nella lingua inglese che non aggiungono informazioni rilevanti al testo.

Questo testo, una volta finite le fasi di *Scraping* e *Raffinamento,* è nella cosiddetta forma “Tokenizzata”, ovvero ogni parola è una stringa a sé stante.

Dopo aver applicato queste fasi, si possono usare queste parole per creare una rappresentazione *Bag of Words* *(BoW)* dei vari documenti componenti il corpus. Questa rappresentazione ci permette di convertire ogni documento che fa parte del corpus in un formato comprensibile dal computer. Più precisamente, tramite questa rappresentazione, siamo in grado di esprimere ogni documento come un vettore n- dimensionale, le cui componenti sono dei numeri naturali.

In alternativa, possiamo anche usare un altro tipo di rappresentazione, quella TF-IDF, la quale si poggia su altri principi.

La comparazione tra questi due modi di rappresentare i documenti verrà fatta in seguito, in modo da capire come questo attecchisca sul funzionamento e resa di LDA.

# Step 1 – Reperimento 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 trovareun 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.

Figura 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.

**Esempio di output di questa fase**:

*“…*

*Ratings for AmazingShop*

*All Recent Feedback for AmazingShop:*

*Rating*

*Feedback*

*Product*

*Type*

*Created*

*User*

*5 ⁄ 5*

*AMAZINGSHOP BEST COUNTERFEIT ESCROW VENDOR THANKS I RECEIVED MY PACKAGE AND I FINALIZED BILLS WHERE EXCELLENT IN BY PASSING ALL PEN TESTS TOP QUALITY BILLS I WILL BE A REGULAR CUSTOMER*

*Physical*

*5 days ago*

*R\*\*\*\*e*

*10 - 25*

*deals*

*…”*

# Step 2 – Splitting del testo

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.

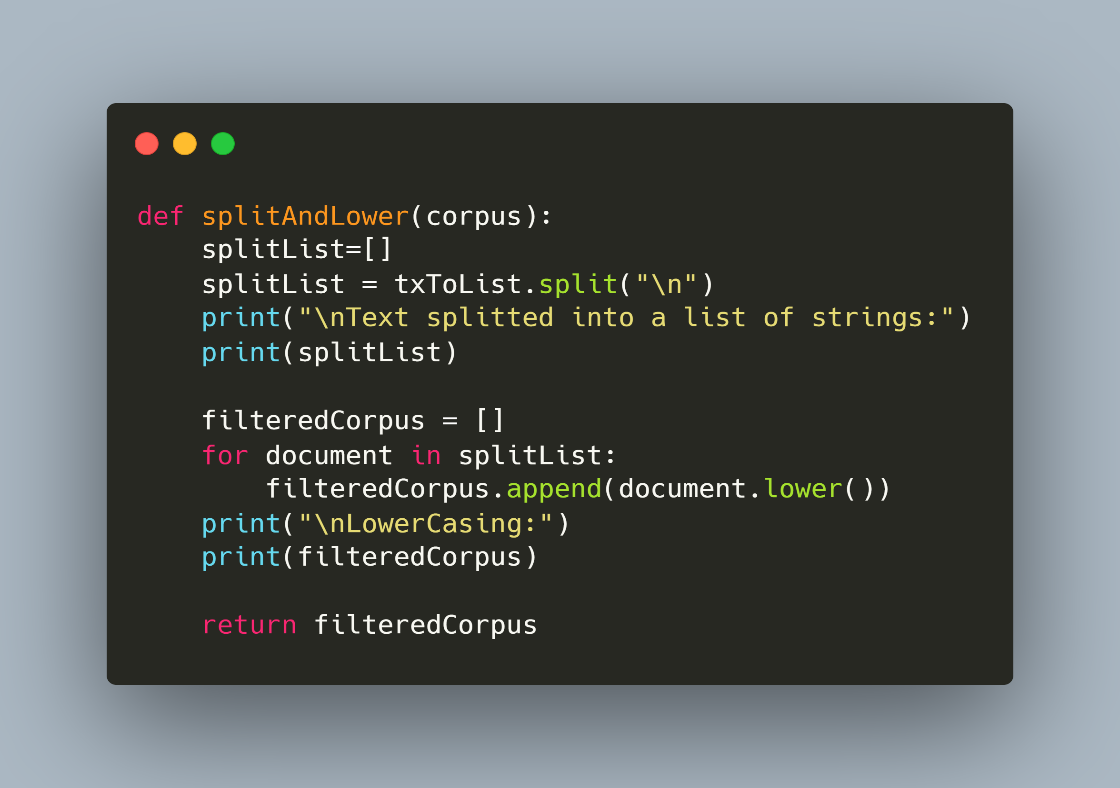


Figura 2 - Funzione per effettuare lo splitting del testo, al fine di portarlo in minuscolo.

**Esempio di output di questa fase**:

['ï»¿', 'agartha : ratings for amazingshop', 'ag', 'agartha underground oasis', 'listings', 'profile', 'wallet', 'orders', 'forums', 'info/help', 'welcome', 'anna\_sani23', '::', 'wallet', 'à¸¿', '0', '(0.00000000)', 'dash:', 'à¸¿', '0.0', '(0.00000000)', 'ltc:', 'à¸¿', '0.0', '(0.00000000)', 'bch:', 'à¸¿', '0.0', '(0.00000000)', 'vtc:', 'à¸¿', '0.0', '(0.0000000), …]

# 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.

**Esempio di output di questa fase:**

[[ ], ['agartha', 'ratings', 'amazingshop'], [ ], ['agartha', 'underground', 'oasis'], ['listings'], ['profile'], ['wallet'], … ]



Figura 3 - Funzione per effettuare la Tokenizzazione del testo

# Step 4 – Eliminazione documenti corti

L’input di questa fase è costituito dunque da una lista di liste contenente un insieme di Token, rappresentanti ognuno le parole della frase, rappresentata a sua volta dalla lista stessa.

Questa operazione di Tokenizzazione e di creazione di una lista di questi Token per frase è stata fatta per uno scopo preciso. Come avevamo evidenziato all’inizio, lo scraping che è stato necessario mettere in atto non legato strettamente alla struttura del DOM della pagina HTML presa in considerazione. Questo significa che non è stato possibile disporre fin da subito, pur attraverso un’ attenta analisi della struttura del sito, del testo legato ai soli commenti lasciati dagli utenti sulle pagine dei venditori. Questo ha reso necessario mettere in atto delle considerazioni che saranno qui elencate e spiegate.

Prima di tutto, osservando il corpus grezzo estratto da diversi Marketplace di cui disponiamo il dump nel dataset ANITA, è emerso che questo corpus estratto era formato più o meno sempre nello stesso modo. Si è osservato che tutte le righe inerenti ai commenti hanno almeno 4 parole, mentre tutte coloro che ne hanno di meno rappresentano testo spurio da eliminare.

Quello che è stato fatto, dunque, è eliminare ogni riga ottenuta dalla fase precedente che abbia meno di 4 parole.

**Considerazioni**

Su questa fase ci sono state alcune considerazioni da fare, che hanno permesso di delineare meglio la pipeline di sviluppo.

Sin dall’inizio, sono stati oggetto di studio: la scelta dei passi da prendere in considerazione per effettuare il preprocessing nel migliore dei modi; come arrangiare questi passi, in modo da preservare la maggior parte delle parole (perchè non ne disponiamo molte) che siano inerenti esclusivamente alle recensioni che intendiamo estrarre. Questo step si è rivelato essere fortemente influenzato dallo step di raffinamento di rimozione delle stopwords.

La spiegazione a ciò è dovuta al fatto che, in diverse occasioni, dei commenti che visualizzati nel browser sembravano essere omogeneamente distribuiti su più linee, in fase di analisi e di preprocessing sono risultati invece spezzati su più righe differenti, senza un ordine preciso. Questo ha fatto sì che successivamente, nel corpus, fossero presenti delle righe inerenti ad un commento, che quindi voglio preservare, che avessero meno di 4 parole, oppure esattamente 4. Questo avrebbe significato che, applicando la funzione *delUnder4Words,* avrei perso tutte quelle righe contenenti esattamente 4 parole legate ad un commento, tra le cui parole ci fosse stata almeno una stopword. Se avessi condotto la rimozione delle stopword prima di questa fase, avrei perso delle parole inutilmente. Questo rappresenta un problema, dato che stiamo operando in un contesto in cui già di partenza non disponiamo di molto materiale, e quindi un approccio orientato al “conservare” è più indicato (tutto deve essere misurato alla perfezione).

Per risolvere questo problema si è deciso di rimandare la rimozione delle stopwords come ultimo step di raffinamento. Ovviamente, conservando in questo modo anche le righe contenenti esattamente 4 parole, vado ad introdurne alcune spurie, le quali saranno eliminate successivamente.

Per capire l’efficacia di questo metodo, sono stati condotto dei piccoli studi statistici sul corpus estratto e preprocessato fino allo step 3. Da questi piccoli studi è emerso il seguente risultato:

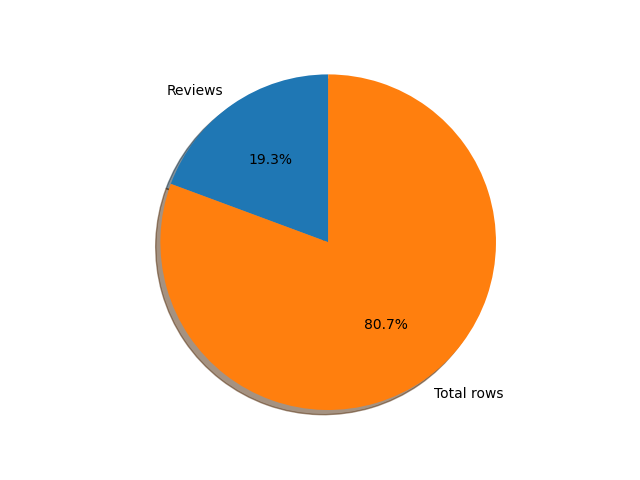


Figura 4 - Percentuali delle righe estratte inerenti alle recensioni sul totale delle righe estratte dal Marketplace Agartha.

Le percentuali di testo legato ai soli commenti, presenti in una pagina web di qualsiasi Vendor, non supera mai il 20%.

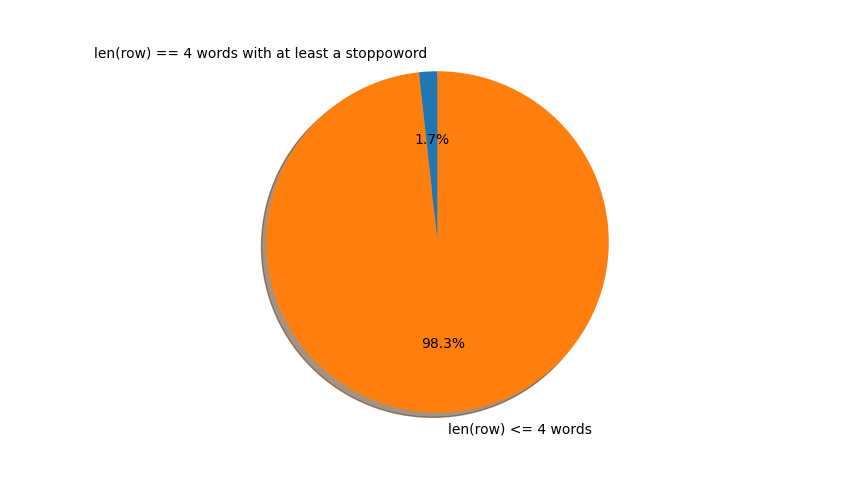


Figura 5 - Diagramma che mostra la percentuale delle righe con 4 parole ed almeno una stopwords, rispetto a tutte le righe di lunghezza minore o uguale di 4.

Se avessimo condotto la rimozione delle stopwords prima di questo step avremmo perso in media 3% del testo. Anche se sembra poco, è bene ricordarsi che ogni parola è importante, soprattutto con un corpus così scarno come quello delle recensioni su un Marketplace del deep/dark-web.



Figura 6 - Funzione per l'eliminazione dei documenti che contengono meno di 4 parole.

# Step 5 – Rimozione punteggiatura e numeri

Una volta che si ha questa lista di documenti contenenti almeno 4 parole, andiamo a “pulire” ogni parola da eventuali simboli di punteggiatura e numeri. Per fare ciò mi avvalgo della funzione noPuncNoNumb, la quale prende in input il corpus dalla fase immediatamente precedente ed applica a ciascuna parola presente in ogni documento del corpus ricevuto come parametro la funzione strip\_punctuation(str) di Gensim. Quest’ultima rimpiazza tutti i simboli di punteggiatura con uno spazio, e la funzione strip\_numeric(str) la quale piuttosto che rimpiazzare un numero con uno spazio, rimuove il numero, e concatena le due stringhe che erano presenti alla sinistra e alla destra del numero.

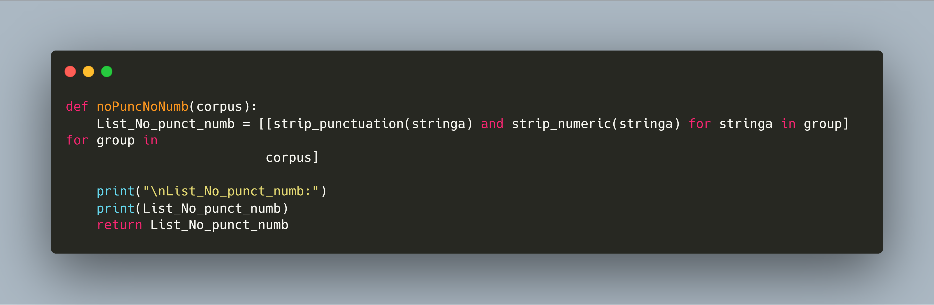


Figura 7 - Funzione che si occupa di rimuovere la punteggiatura e ogni numero da una data parola.

# Step 6 – Correzione dello spelling

Spesso nelle recensioni lasciate da utenti sulla pagina di un venditore si possono osservare degli errori ortografici. Questo fa sì che una parola scritta correttamente e la stessa scritta con qualche errore ortografico vengano considerate come due parole distinte. Per evitare, quindi, di introdurre successivamente nel **dizionario delle parole** due parole diverse ma che hanno lo stesso significato, e quindi lo stesso contenuto informativo, dobbiamo procedere alla correzione dello spelling. Adesso qui si potrebbero aprire delle speculazioni per quanto riguarda l’*illiteracy as a feature,* ovvero che anche gli errori ortografici possono essere una espressione caratterizzante di una comunità. Se l’errore ortografico fosse sistematico, allora potrebbe essere considerata una caratteristica peculiare degli individui. Si potrebbe immaginare che su un sito appartenente al Dark/Deep-web ci sia più probabilità di trovare degli errori piuttosto che su un Forum di storia della letteratura inglese. In merito al seguente aspetto, si condurranno degli esperimenti per verificare la veridicità dell’intuizione. Per correggere lo spelling mi sono avvalso della funzione errataCorrige(corpus), la quale riceve come parametro in input il corpus proveniente dalla fase immediatamente precedente. In questa funzione viene utilizzato il metodo *correct* appartenente alla libreria di NLP *TextBlob,* la quale si poggia a sua volta sulla rinomata libreria di Natural Language Processing *NLTK*. Per ogni parola presente nel corpus, dunque, viene corretto lo spelling e viene inserita in una nuova lista. Come output, da questa funzione avremo una nuova lista di liste contenenti tutte le parole corrette.

In merito alla correttezza si dovrebbe fare un appunto. Ogni libreria di spelling correction, infatti, è risultata soffrire di *Over correction*, ovvero quel fenomeno in cui viene corretta una parola che è già corretta. Questo significa che, per esempio, se un testo presenta un errore del 30% sulla totalità delle parole, applicando la correzione dello spelling non è possibile arrivare ad una percentuale di errore dello 0%.

# Step 7 – Espansione delle contrazioni

In questa fase viene esaminata ogni parola e, se quest’ultima contiene una contrazione, viene riportata nella sua forma non contratta.

E.g. Se abbiamo la parola *don’t* questa deve essere portata alla sua forma non contratta *do not*.

# Step 8 - Singolarizzazione delle parole

In questa fase si esamina ogni parola per portarla alla sua versione singolare, in modo che la versione singolare e la versione plurale possano così essere considerate come la stessa cosa. Questo step è fortemente collegato al successivo step di *Lemmatizzazione delle parole,* in quanto la funzione singAndLemm(corpus), usata per portare le parole al singolare, prima di ritornare la lista delle parole processate, applica immediatamente lo step successivo, chiamando nella fase di return la funzione lemmatize(corpus). L’output, quindi, è disponibile direttamente alla fine della fase 9.

# Step 9 – Lemmatizzazione delle parole

La lemmatizzazione consiste nel portare ogni verbo e ogni sostantivo nella sua forma base, e.g. playing 🡪 play.

# Step 10 – Rimozione delle Stopwords

Avendo a disposizione finalmente il testo estratto, vado a rimuovere le stop words da ogni stringa. Le stop words sono tutte quelle parole che vengono usate spesso con un’alta frequenza in un testo scritto di una determinata lingua, che non trasportano informazioni significative, per cui la rimozione è giustificata. Questo favorisce la riduzione di elementi spuri presenti nel corpus, a favore di spazio, tempo di processamento e contenuto informativo. Nel caso in questione, è stata presa in considerazione la lingua inglese, ed in particolare, sono state eliminate tutte le stop words definite da Gensim.

Una volta eliminate le stopwords da ogni stringa, possiamo procedere con la fase successiva.

**Calcolo delle frequenze:** A questo punto posso andare a calcolare ogni parole con che frequenza compare corpus, salvando questa informazione in un dizionario, dove ogni elemento è individuato da una coppia chiave/valore. La chiave è rappresentata dalla parola, mentre il valore è rappresentato dalla frequenza di quella parola all’interno del corpus.

**Assegnazione di un id intero, unico per ogni parola**

Ogni parola viene contrassegnata da un id intero unico, in modo da poter successivamente rappresentare ogni documento come un *bag of word*.

**Conversione in Bag-of-words:** scrivere la teoria\*

# Valutazione dello scraper

# Statistiche estratte dal dataset ANITA