Università degli studi di Milano-Bicocca

Big Data in Public and Social Services

Progetto Finale

AN ALGORITHM TO MANIPULATE DOCUMENT SENTIMENT

Autori:

Luca Gabellini	777786	l.gabellini1@campus.unimib.it
Matteo Provasi	782922	m.provasi2@campus.unimib.it
Pierluigi Tagliabue	835211	p.tagliabue7@campus.unimib.it

1. Obiettivi del progetto e utilità

L'obiettivo del progetto è la creazione di una funzione che, dato in input un documento testuale, sia in grado di trovare il minor numero di termini da aggiungere al documento tale per cui il sentiment sia, in valore assoluto, superiore ad una soglia prefissata dall'utente. Come concordato con il relatore la frase ottenuta non deve essere semanticamente corretta in quanto sono richieste delle competenze che esulano i requisiti del corso ma d'altra parte l'algoritmo non si deve limitare ad aggiungere parole appartenenti ad un'unica parte del discorso.
Poste le limitazioni sopracitate, il progetto si pone in un'ottica di Natural Language Generator (NLG), ovvero quei processi applicabili per esempio alle chatbot o ad altri strumenti in cui si cerca di simulare una conversazione.

2. Architettura e strumenti utilizzati

I dati sono stati presi da una competizione di Kaggle[1] i cui obiettivi erano però differenti da quelli del nostro progetto, dunque non esiste un metro di comparazione con i risultati che saranno illustrati in seguito. I dataset presenti sul sito sono anche le uniche fonti dati utilizzate, in particolare i dati sono stati ottenuti mediante la libreria nyt-comments[2] che contiene al suo interno funzioni che si comportano da API wrapper per ottenere articoli e commenti dal New York Times.
I dataset sono catalogati in due tipologie: articoli e commenti. Per gli articoli le variabili tenute in considerazione sono l'ID, una chiava univoca che identifica l'articolo e il set di keyword fornito. Tramite il codice identificativo è stato possibile effettuare un join con i dataset relativi ai commenti di cui si è tenuta la variabile contenente il testo completo del commento.

3. Upload

Per caricare i dati sono state create due funzioni, `get_articles` e `get_comments`, entrambe dato un path in input iterano all'interno della cartella, leggono tutti i file presenti con una determinata estensione e si concatenano i dati in un dataframe.
Una volta ottenuti i dati si effettua il join mediante l'ID identificativo in modo da avere in un unico elemento i commenti e le keywords dell'articolo di riferimento.

4. LDA

Prima di procedere con la classificazione è stato necessario definire una variabile da prendere come target. L'idea originale era quella di utilizzare le keywords, tuttavia si sono trovate circa 13500 singole keywords ed oltre 7000 gruppi distinti, molti dei quali con un'unica occorrenza. Per ridurre il numero di classi si è scelto di implementare una Latent Dirichlet Allocation (LDA) per estrarre un numero ragionevole di topic a cui saranno successivamente assegnati i commenti. L'idea di base della LDA è quella che esistano dei topic non osservati (latenti appunto) di cui si osservano parole e documenti. L'obiettivo è di connettere al meglio le parole ai topic in base a quanto coerentemente le parole si dovrebbero trovare in quel topic ed infine connettere i topic ai documenti in base ai loro argomenti. La funzione `prepare_lda` prende in input la lista di keywords e da quelle crea un dizionario e la bag of words, questi elementi sono passati alla funzione LDA. Per trovare il miglior modello LDA si è effettuata una grid search creando modelli con diverse numerosità per i topic. Per ogni modello si sono calcolate due metriche per la qualità dei modelli ottenuti, la perplexity e la coherence.

[1] Kaggle, Comments on articles published in the New York Times, https://www.kaggle.com/aashita/nyt-comments
[2] Github, nyt-comments, https://github.com/AashitaK/nyt-comments

La perplexity è definita come il reciproco della media geometrica delle verosimiglianze dei token nel corpus dato il modello creato[3], in formula:

$$p(\vec{W}|M) = exp - \frac{\sum_{m=1}^{M} \log p(\vec{\widetilde{w}}_{\widetilde{m}}|M)}{\sum_{m=1}^{M} N_m}$$

Il valore della perplexity deve essere il più basso possibile che si ottiene quando $\log p(\vec{\widetilde{w}}_{\widetilde{m}}|M)$ è piccolo $\forall m$ ovvero quando le probabilità tendono ad uno. Per ragioni computazionali si utilizza la `log_perplexity` che, a differenza del nome, non è una semplice trasformazione logaritmo della perplexity ma indica il bound della disuguaglianza per la stima della perplexity:

$$p(\vec{W}|M) \leq e^{-bound}$$

La coherence è invece una misura che indica quanto sono coesi i topic estratti[4]. Definito $D(w_i)$ il numero di documenti che contengono la parola w_i il coherence score è calcolato come:

$$Coherence_{score} = \sum_{i<j} score(w_i, w_j)$$

Dove con il metodo UMASS utilizzato lo score è:

$$score_{UMASS}(w_i, w_j) = \log \frac{D(w_i, w_j) + 1}{D(w_i)}$$

Per costruzione lo score sarà negativo in quanto la frequenza congiunta di due token non potrà mai essere mai superiore alla frequenza di un singolo token (il valore 1 aggiuntivo serve come smoothing nel caso non ci siano documenti comuni evitando quindi valori indefiniti del logaritmo) di conseguenza il logaritmo sarà negativo. Con questa metrica il valore maggiore (minore in valore assoluto) indica un modello che ha generato topic più coesi.

Per la scelta del numero ottimale di topic le metriche precedentemente descritte non hanno permesso l'identificazione di un unico modello migliore degli altri, infatti i valori minimi di perplexity e coherence erano relativi a due modelli differenti. Di conseguenza si è optato per una scelta dettata dal giudizio umano, si è così deciso di scegliere un numero di topic pari a nove in quanto presentavano tra loro una buona distinzione e al contempo erano informativi.
I topic trovati sono i seguenti:

- US_Elections_2016
- Environment
- International
- US_Politics
- Guns
- Ethics
- Laws
- (Social)_Media
- News

Trovato il modello migliore la funzione `lda_to_topic` assegna ad ogni commento il topic a cui più verosimilmente appartiene mediante il modulo `get_document_topics`. Dato che una keyword potrebbe essere contenuta in più di un topic, l'algoritmo implementato è di tipo softmax, quindi restituisce già un vettore a somma unitaria. Questa funzione è contenuta all'interno di un'altra funzione `lda_to_topic`, che restituisce il dataset con le keyword cambiate.

[3] Si Chen and Yufei Wang: Latent Dirichlet Allocation, http://acsweb.ucsd.edu/~yuw176/report/lda.pdf
[4] qpleple, Topic Coherence To Evaluate Topic Models, http://qpleple.com/topic-coherence-to-evaluate-topic-models/

5. Pre-processing

Lo step successivo consiste nel normalizzare il testo dei commenti. Sono state effettuate tutti gli step classici del pre-processing con la maggior parte delle funzioni già implementate nella libreria gensim. Particolare attenzione è stata rivolta alla rimozione degli URL, in quanto gli eventuali link ipertestuali nei commenti sono visualizzati come URL completi. Dopo un primo pre-processing sono state applicate ulteriori modifiche ai commenti per aggiustare i risultati ottenuti negli step successivi: uno stemming per portare alla radice le parole e la rimozione dei commenti con meno di dieci parole in quanto troppo corti e difficili da analizzare per i modelli classificativi.

6. Word2Vec & T-SNE

Il Word2Vec è un modello predittivo utile per fare word embedding.
Dato il corpus di interesse, Word2Vec utilizza una rete neurale finalizzata a generare un vettore numerico per ciascuna parola presente nel corpus. Lo spazio vettoriale corrispondente presenterà vettori semanticamente simili vicini tra loro e viceversa.
Esistono 2 approcci alternativi utili per generare il vettore, CBOW (Countinuous Bag-of-Words) e Skip-gram. Il primo ha lo scopo di predire la parola dato il suo contesto, mentre il secondo al contrario, predice il contesto data la parola.

Come input per la funzione `word2vec()` del package gensim è stato selezionato un campione casuale di 40000 commenti preprocessati.
È stata inoltre condotta una grid search per individuare quali fossero i valori migliori dei seguenti parametri:

- size: dimensione del vettore
- sg: approccio utilizzato, 0=CBOW, 1=Skip-gram
- perplexity: parametro incluso nella funzione relativa alla T-SNE

La rappresentazione ottenuta con la procedura Word2Vec è stata poi utilizzata come input per l'algoritmo TSNE, utile per visualizzare graficamente uno spazio vettoriale a D dimensioni in sole d dimensioni, 2 nel nostro caso.
Dato uno spazio d-dimensionale $X \in R^D$, t-SNE produce uno spazio $Y \in R^d$ dove $d \ll D$. Se due punti x_i e x_j saranno vicini nello spazio X, un'elaborazione ottimale prevederà i corrispondenti punti y_i e y_j altrettanto vicini.
Per fare ciò, l'algoritmo modella le similarità tra punti come probabilità. Nello spazio originale le similarità sono date da:

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}$$

Con probabilità congiunte:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2}$$

Nello spazio a dimensionalità ridotta viene utilizzata una distribuzione T-student per modellare le probabilità:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)}$$

Da qui il nome t-SNE. Il nostro obiettivo sarà quello di rendere Q il più simile possibile a P, utilizzando la divergenza di Kullback-Leibler:

$$C = KL(P||Q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

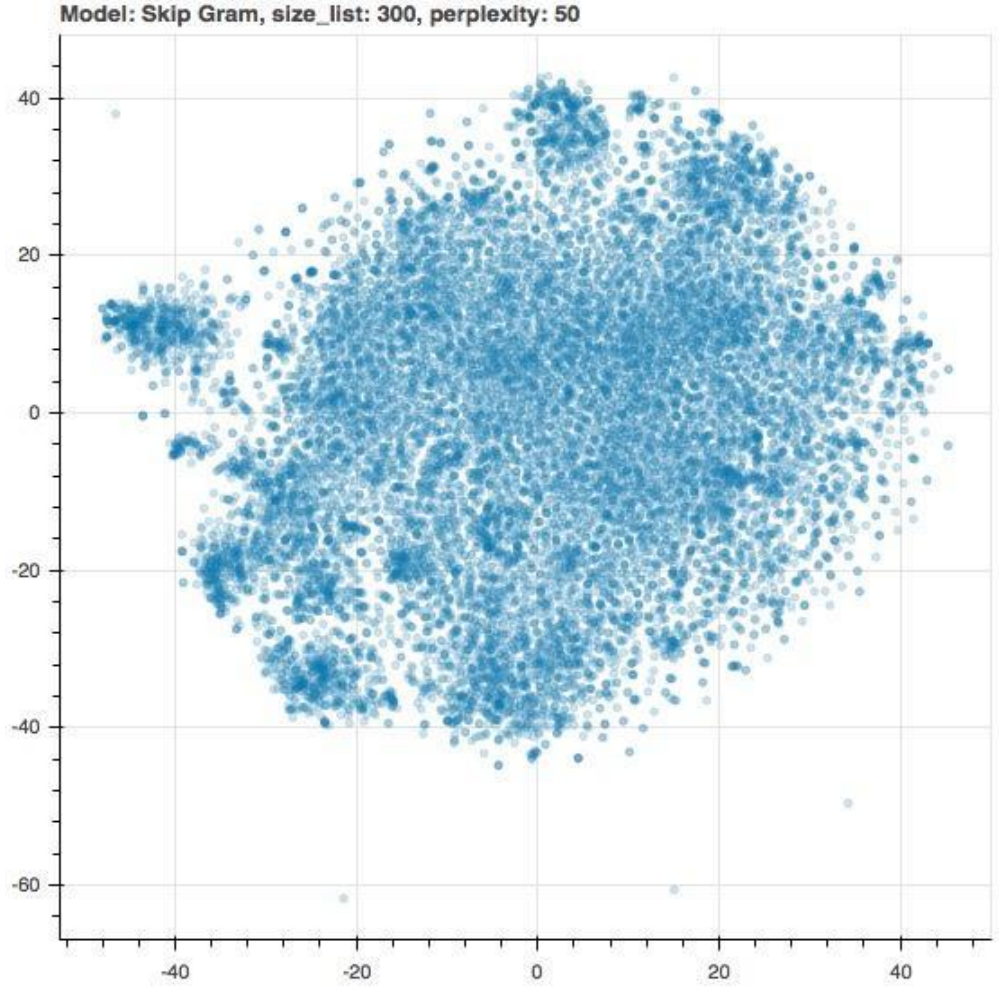
La perplexity in questo contesto è definita come: $Perplexity(p_i) = 2^{H(p_i)}$. Dove H è l'entropia di Shannon di una distribuzione discreta:

$$H(p_i) = -\sum_i p_{j|i} \log_2(p_{j|i})$$

Il valore della perplexity è, in sostanza, una previsione del numero di vicini stretti che un singolo punto avrà nel plot finale. Usualmente si scelgono valori di perplexity compresi tra 5 e 50.
La rappresentazione visivamente migliore è risultata quella con:

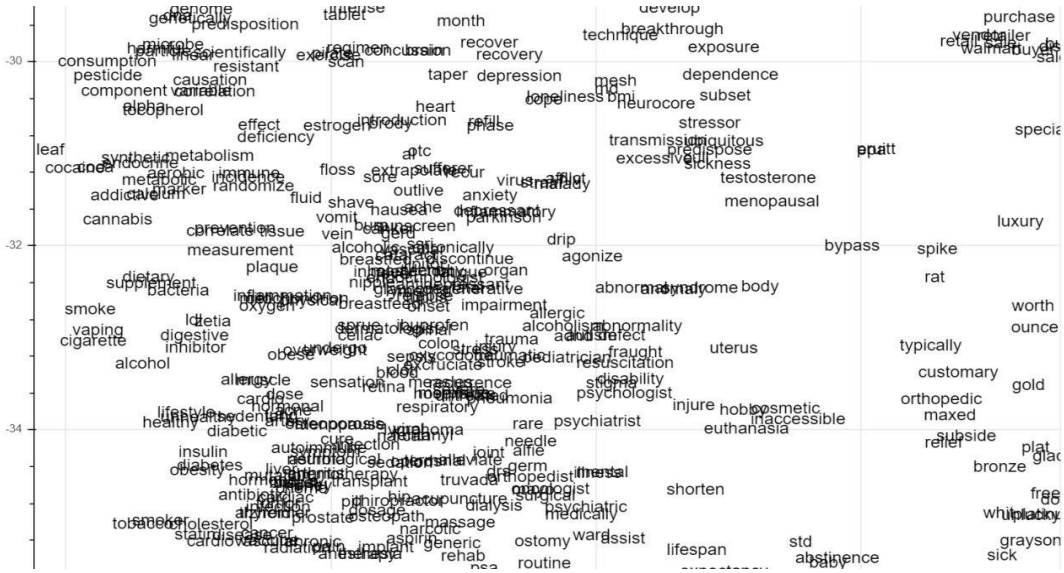
- size = 300
- sg = Skip gram
- perplexity = 50

Tutte le combinazioni di parametri non portavano ad una rappresentazione con dei cluster distinti. Nell'immagine seguente è riportato il plot del modello migliore in cui si possono distringere degli agglomerati distanti dalla giant component centrale.



Rappresentazione in due dimensioni dell'output della t-SNE

È stato poi possibile, grazie anche ai plot interattivi, andare a visualizzare quali fossero i termini che formassero i cluster distinti. Nella seguente immagine è riportato uno zoom del grafico che riporta i termini nel cluster in basso sinistra che si possono identificare come topic *Health*.



t-SNE con i termini al posto dei punti

7. Modelli di classificazione

L'assegnazione dei topic ha portato ad avere delle classi non bilanciate all'interno del dataset. Dai primi modelli creati si è subito notato che le classi meno numerose avevano un discreto valore di precisione ma un basso livello di recall. Questa situazione avviene quando si hanno poche osservazioni in una classe e i modelli fanno fatica ad assegnare osservazioni a quella classe. Il problema si è risolto utilizzando congiuntamente un undersampling ed un oversampling. Preso il valore medio di commenti appartenenti ad una classe, per i topic più frequenti si è scelto un sample senza reinserimento, mentre per i topic meno numerosi si sono riestratti dei commenti fino ad arrivare alla numerosità pari alla media originale.
Prima di partire con i modelli di classificazione è stato necessario trasformare i commenti in modo da renderli utilizzabili da parte dei modelli. Il metodo tradizionale per i testi crea dei vettori con la metrica TF-IDF (Term Frequency-Inverse Document Frequency). L'algoritmo di sklearn utilizza come document frequency grezza il numero di volte in cui il termine t compare nel documento d. L'inverse document frequency è calcolata come:

$$idf(t, D) = ln\left(\frac{N+1}{1+|\{d \in D: t \in d\}|}\right)$$

Dove N indica il numero di documenti totali e al denominatore il numero di volte che il termine t compare in un documento d; viene aggiunto infine uno smoothing unitario per evitare valori indefiniti.
Ad ogni parola viene quindi associata un valore numerico. Questo valore aumenta alla frequenza della parola e quanto meno diffusa è nei documenti, la logica dietro a questa metrica è di "premiare" le parole con alta frequenza ma concentrata in pochi documenti. Per la selezione dei modelli si sono scelte delle tecniche e valutati i parametri di accuracy nel test set al variare dei parametri di input tramite una grid search.
La maggior parte dei modelli provati (Reti Neurali, Random Forest) erano dispendiosi a livello computazionale data la grossa mole dei dati e pertanto una ricerca tramite grid search era impraticabile. Si sono quindi provati inizialmente alcuni parametri e alla fine si è scelto il modello migliore per continuare una grid search più intensiva, ovvero il modello Naive Bayes Multinomiale. Questo modello è risultato il migliore sia in termini di accuracy che in termini di performance.
La funzione utilizzata, `MultinomialNB()` implementa l'algoritmo Naive Bayes per target multinomiali.
La distribuzione dei dati è parametrizzata dai vettori $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$ per ogni categoria del target y, dove n è il numero di features TF-IDF e θ_{yi} è la probabilità $P(x_i|y)$ che la feature i appaia nel campione appartenente alla classe y.

I parametri θ_y vengono stimati con una versione smoothed della massima verosimiglianza, ovvero la frequenza relativa

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Dove N_{yi} è il numero di volte in cui la feature i appare in un campione di classe y nel training set e N_y è il conteggio totale di tutte le features della classe y.
Il parametro α se fissato maggiore di zero permette di fare uno smoothing, aggiustando per feature presenti solo nel test e non nel train, ed evitando quindi una stima nulla delle probabilità[5].

Per migliorare le performance di classificazione nel TF-IDF si è inserita una soglia minima di frequenza, al di sotto della quale le parole non venivano considerate in quanto ritenute eccessivamente rare e quindi potenzialmente inefficaci per la classificazione. La funzione dà inoltre la possibilità di considerare bigrammi o un misto con le parole. Dalle analisi si è ottenuto che la rappresentazione con solamente i bigrammi è stata la migliore con un'accuratezza di 0.5427, leggermente superiore a quella ottenuta con parole singole e bigrammi.
Per ogni modello si è costruita la matrice di errata classificazione ed una rappresentazione tramite heatmap. La classe con l'accuratezza maggiore è stata `International` con valori di 0.66, mentre la peggiore con 0.40 è stata `Social Media`. I valori di recall non si distanziavano significativamente da quelli di accuracy e anche controllando la matrice si è notato che non ci fossero degli errori sistematici di classificazione.

Per formulare la frase finale non si è solamente fatto riferimento ai testi dei commenti che abbiamo ottenuto, ma è stata implementata anche una comparazione con un corpus di default preso dalla letteratura, nello specifico il corpus denominato "Brown" presente all'interno del pacchetto ntlk.
Questo corpus fornisce oltre un milione di parole provenienti da oltre cinquecento fonti diverse. Ogni parola è etichettata con una parte del discorso. Per semplificare il matching e non renderlo troppo restrittivo, è stato selezionato il tagset universale: le diverse tipologie di una parte del discorso (nome proprio, nome comune ad esempio) sono raggruppate nella tag principale (nell'esempio "nome").
Successivamente si è partiti con l'implementazione dell'algoritmo di Viterbi. Questo algoritmo, basato un processo markoviano (la probabilità di essere in uno stato in un determinato istante dipende solo dallo stato all'istante precedente), permette la creazione di una sequenza di tag a partire da una o più parole. Dato in input il corpus con i tag l'algoritmo stima le probabilità di ogni singola parola di essere etichettata con un determinato tag (una parola può essere classificata in maniera differente a seconda del contesto) e genera delle sequenze di parole osservando quelle precedenti. In questo modo data in input una nuova sequenza di parole, l'algoritmo restituisce la sequenza di tag che più verosimilmente dovrò susseguirsi alle parole date.

8. Generazione frase

Come precedentemente detto l'obiettivo del progetto è la creazione di una funzione che, dato in input un documento testuale, sia in grado di trovare il minor numero di termini da aggiungere al documento tale per cui il sentiment sia, in valore assoluto, superiore ad una soglia prefissata dall'utente.
Questo obiettivo viene raggiunto attraverso la funzione `final_function` al cui interno si trovano diverse sotto-funzioni per eseguire i diversi step:

1. La prima funzione è `random_select` che prende in input una lista di commenti, nel nostro caso si è creata un'unica lista con tutti i commenti presenti nel dataset semi-processato, ed estrae una sequenza di parole di lunghezza predefinita dall'utente. La frase estratta viene salvata e sarà quella a cui verranno aggiunte passo per passo le nuove parole, una copia di questa frase è invece passata al pre-processing in modo da normalizzare il più possibile la frase.
2. L'utente può decidere tramite un input quale sia il sentiment da raggiungere prima che l'algoritmo si concluda. Una volta ottenute le due frase, quella con il pre-processing viene passata alla funzione `category`. Questa funzione prende in input la frase ed un modello di classificazione e viene stimata

[5] Scikit Learn, Naive Bayes https://scikit-learn.org/stable/modules/naive_bayes.html

la probabilità di appartenenza ad ogni classe, viene quindi stampato a video il dataframe con queste probabilità ordinate in modo decrescente e le rispettive classi. Si crea un nuovo dataframe che avrà come nome il nome della classe selezionata e come elementi solamente i commenti relativi a quella classe. Il concetto di base dietro a questo passaggio è che per aggiornare la frase scelta si debba fare riferimento solamente ai commenti e al lessico utilizzato nei commenti di quella classe.

Pred	class
0.67452	Laws
0.127029	International
0.0588022	Ethics
0.0436148	Environment
0.0308774	US_Politics
0.0181738	News
0.0178023	(Social)_Media
0.0155433	US_Elections_2016
0.0136375	Guns

Esempio di output della funzione `category`.

3. La funzione successiva è inserita all'interno di un loop che continuerà a ripetersi fino a quando non si sarà raggiunto il sentiment desiderato o si arriverà ad una condizione di stop (non è possibile aggiungere nessuna parola). Il dataset e la frase estratta sono passati alla funzione `words_dataframe`. L'obiettivo di questa funzione è quello di trovare le sequenze di parole adatte per aggiornare la frase. In realtà questo processo è suddiviso in due step, il primo dei quali è eseguito da una sotto-funzione:
    a. La funzione `follow` prende la frase e cerca in tutti i commenti che contengono la parte finale di essa. Per i commenti che la contengono si effettua uno split e si prendono tutti gli elementi di indice 1 e successivi (in quanto interessa solo la parte di commento che segue la frase in input). I pezzi di commenti che contengono almeno un numero predefinito di parole sono inseriti come chiavi di un dizionario; dopo aver passato tutti i commenti si passa al conteggio della frequenza. Dato che l'obiettivo principale dell'analisi è modificare il sentiment, questi pezzi di frasi sono temporaneamente divisi in singole parole e per ognuna di esse viene calcolato il sentiment. Nel caso il sentiment sia nullo viene mostrato a video un avviso ma non implica la conclusione dell'algoritmo.
    b. Ottenuto il dizionario di frequenza come output si creano tante liste quante sono le parole che compongono le sequenze (per costruzione tutte avranno la stessa lunghezza): la prima lista conterrà dunque le prime parole di ogni sequenza, la seconda le seconde e così via. Si creano inoltre due liste aggiuntive: una per il conteggio di quante volte quella sequenza è stata trovata nei commenti e il sentiment.

    Per trovare la sequenza più adatta si è creato un indice che fosse una media tra il conteggio e il sentiment della sequenza. È possibile che una determinata sequenza abbia un sentiment molto elevato ma compaia una singola volta e allo stesso tempo ci sia una sequenza con basso sentiment ma che sia molto frequente. Questo indice cerca di ottenere un trade-off tra questi due parametri per selezionare la sequenza più adatta. Viene creato un dataframe a partire da tutte le liste create che viene messo in output.
4. A questo punto tutte le sotto-funzioni sono terminate e si ritorna alla `final_function`. Il dataset trovato viene filtrato per valori positivi o negativi a seconda del sentiment dato in input e ordinato conseguentemente per l'indice identificato al punto precedente. Come ultimo passaggio viene applicato l'algoritmo di viterbi ad ogni parola della sequenza per identificarne la parte del discorso più probabile e alla fine della frase ottenuta all'inizio dell'algoritmo. Si confronta la sequenza più probabile di tagset con quella ottenuta da ogni sequenza nel dataframe e si tengono solamente le osservazioni che rispettano il criterio. La frase viene quindi aggiornata con la prima riga del

dataframe, se il nuovo sentiment è superiore alla soglia prefissata, la frase è completata, altrimenti si procede per via iterativa.

Riassumendo, la funzione `final_function` prende in input i seguenti parametri:

- **Dataframe**: un dataframe con semi pre-processing che sarà utilizzato come input per trovare le parole da aggiungere alla frase selezionata casualmente.
- **Model**: classificatore per individuare la classe di appartenenza più probabile della frase generata casualmente.
- **Lenght**: il numero di parole da cui deve essere composta la frase generata casualmente.
- **Comment_List**: una lista di tutti i commenti da cui deve essere generata la frase.
- **Threshold**: non è un parametro della funzione ma viene richiesto tramite input all'inizio della funzione.
- **End_Seq**: il numero di parole prese dalla fine della frase che devono essere utilizzate per ricercare una sequenza adatta. Nel caso non ci siano sequenze adatte, il numero diminuisce progressivamente. Nel caso sia la prima iterazione il parametro viene ignorato e si prende la lunghezza della frase stessa.
- **n_match**: è il numero minimo di sequenze distinte che devono essere trovate prima di passare ad una sequenza più ristretta.
- **Alpha**: è un parametro di smoothing per il calcolo del nuovo indice di trade-off tra frequenza e sentiment. Nel caso il sentiment di una sequenza sia nullo viene aggiunta una quantità ragionevolmente piccola in modo da non avere un indice sempre nullo.
- **Method**: per rendere l'algoritmo flessibile sono state implementate due filosofie:
    - Quanto questo valore è 0 l'algoritmo rifiuterà le sequenze trovate (anche se rispettano tutti i criteri sopra elencati) se tutte hanno sentiment nullo. La logica dietro a questo metodo è di perdere in parte il livello semantico della frase per cambiare il sentiment nel minor numero di parole.
    - Quando questo valore è 1 l'algoritmo è più flessibile, ritiene più importante la creazione di una parola a livello semantico e quindi accetta eventuali sequenze con sentiment nullo.

9. Conclusioni

L'algoritmo non dà errori con nessuna delle combinazioni dei parametri testati e restituisce una frase in un tempo ragionevole (ogni iterazione dura in media un secondo). A livello semantico la frase finale non ottiene grandi performance se non in alcuni casi: generalmente ci sono delle porzioni ottimali ed altre che non hanno un gran significato. Questo potrebbe essere dato dal fatto che il numero di corrispondenze che si trovano è abbastanza esiguo nonostante l'elevato numero di commenti e quindi l'algoritmo spesso aggiunge sequenze di parole composte da pochi elementi.
Un'idea per rendere migliore l'algoritmo era quello di prendere dei nuovi dati utilizzando l'API citata nel secondo paragrafo. Per motivi ignoti non è possibile installare il pacchetto e anche copiando le singole funzioni dalla documentazione GitHub, utilizzando le credenziali appropriate, la funzione restituisce sempre zero elementi in output. Possibili miglioramenti riguardano un algoritmo di classificazione più efficacie e cercare di ottenere una frase più sensata a livello semantico.