Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

DATA ANALYTICS

Progetto d'esame

Luca Poli 852027 l.poli6@campus.unimib.it

Sommario

Introduzione

Per questo progetto di Data Analytics affronterò il problema espresso nella traccia 3, ovvero la rilevazione del sarcasmo. Questo problema è uno dei più interessanti nell'ambito di NLP (Natural Language Processing), per la sua complessità; causata della sua natura soggettiva e delle sfumature linguistiche coinvolte; infatti, spesso non è semplicissimo neanche per un essere umano. In particolare, il sistema dovrà esaminare un testo, e aiutandosi con degli elementi esterni (il contesto), dovrà capire se esso è sarcastico o no.

Nella ricerca sono state adottate diverse strategie per questo task, divisibili in due categorie:

- Basati sulle caratteristiche linguistiche: in cui si analizzano i testi alla ricerca di pattern ligustici legati al sarcasmo; come l'uso di più parole contradditorie nella stessa frase, o rispetto al contesto, l'uso di contrasti tra parole con connotazione positiva e negativa oppure l'uso di esagerazioni.
- Basati su modelli di Machine Learning o Deep Learning: in cui si sfruttano tecniche di apprendimento supervisionato (o semi) per addestrare modelli statistici con dati etichettati per rilevare i testi sarcastici; generalmente i modelli posso essere più "semplici" come SVM, Random Forest Naive Bayes, o piccole reti neurali, fino ad arrivare ai modelli di Deep Learning, come RNN (Recurrent Neural Network) o i Transfomer che possono raggiungere e superare i milioni di parametri.

In questo progetto, la strategia risolutiva adottata si baserà sul training supervisionato con i Transformer; in quanto sono un modello che sta rivoluzionando il settore, e sarà interessante applicarli in primo piano (seppur in una scala ridotta) a questo difficile task.

La relazione sarà divisa in tre capitoli principali: nel primo descriverò l'analisi dei dati effettuata, da cui si baseranno le scelte adottate; nel secondo tratterò il modello usato e il relativo addestramento; nel terzo discuterò i risultati prodotti.

Capitolo 1: Analisi dei dati

In questo capitolo descriverò l'analisi dei dati di addestramento, esaminando gli elementi del dataset, il target della classificazione, i testi e gli elementi del contesto. Con cui poter trarre delle conclusioni per poter processare i dati e addestrare il modello con una maggior accortezza.

1.1 Descrizione generale dei dati

1.1.1 Introduzione ai dataset e Splitting

Alla traccia, sono associati due dataset, uno con i dati di addestramento (train) e uno per il testing. Essi sono due insiemi di commenti di Reddit dal 2009 al 2016; in particolare, per ogni istanza abbiamo:

- Text: Testo del commento da classificare;
- Sarcastic: Label booleana che indica se il commento è sarcastico (True) o no (False);
- Author: Nome dell'autore del commento;
- Subreddit: Nome del subreddit in cui è stato pubblicato il commento;
- Date: Data di pubblicazione del commento;
- Parent: Testo della discussione a cui il commento si riferisce;

Il dataset di training originale, con un milione di righe (precisamente 1010822), sarà diviso casualmente in due parti: quello di training definitivo (con il 95%), su cui verranno effettuate tutte le analisi e assunzioni; quello di validation (con il restante 5%), usato durante il processo di training per misurare le performance del modello.

Il dataset di training è molto completo, perché il numero di righe duplicate e non valide è relativamente irrilevante (rispettivamente 200 e 53); dunque non è necessario attuare alcuna strategia particolare per recuperare i dati mancanti o nulli (come l'imputazione), e si può procedere con la loro rimozione. Inoltre, sia la codifica dei testi che delle date rispetta gli standard e non sono dunque presenti errori di conversione.

Sarcastico	Testo	Autore	Data	Subreddit	Parent
false	i was surprised that i've never seen this either... i too am afflicted by an obsession with the her mit kingdom.	RebeccaTwatson	2013-04-01T0 0:00:00	Documentaries	i have an unhealthy obsession with north korea and i'm not sure how i haven't seen this one. i'm super excite to see thi s!
true	he must have a good reason not to give the ba ll to his top-5 rb.	ChocoPeant	2014-12-01T0 0:00:00	fantasyfootball	andy reid is stupid
false	you can run out of ammo if they plant the bo mb (and thereby extending the timer), therefor e it's not impossible.	GhostCalib3r	2015-02-01T0 0:00:00	GlobalOffensive	til it is impossible to run out of ammo in the scout in comp etitive matchmaking due to the high ammo count, low rate of fire and reload speed

Figura 1.1: Alcune righe di training di esempio

1.1.2 Descrizione della metodologia di analisi

Ogni riga è composta da una label e dalla relativa istanza, i cui attributi possono essere divisi nel testo fondamentale da classificare (l'attributo Text), e una serie di attributi contestuali che aggiungono informazioni per il testo (i restanti attributi).

Vista questa divisione ho strutturato il processo di analisi in due fasi:

- Nella prima fase esaminerò gli attributi del contesto, calcolando per ogni elemento unico di ogni attributo qual è il rateo con cui compare in testi sarcastici e non (detto Rateo Informativo), in modo da individuare gli elementi (e quindi i relativi attributi) più rilevanti ad un modello.
- Nella seconda fase esaminerò i diversi modi processare il testo, calcolando quale sistema produce il testo con un maggior contento di rateo informativo.

Nel dettaglio il rateo informativo di un elemento unico ($e \in Feature$) verrà calcolato come:

$$IR_e = \left| \frac{n_{sc}}{n_{sc} + n_{nsc}} - S_{IR} \right| * 100$$

Dove: n_{sc} e n_{nsc} sono rispettivamente il numero di commenti sarcastici e non sarcastici associati ad e; mentre, S_{IR} è la probabilità della classe più probabile.

1.1.3 Analisi del target

Il target, ovvero l'attributo Sarcastic, è la Label che identifica la natura del commento, e sarà dunque l'obbiettivo del modello di NLP.

Per la buona riuscita del training è importante che la sua distribuzione sia bilanciata, ovvero che il numero di istanze sarcastiche e non sia approssimativamente simile. Come sostenuto nell'articolo "A Large Self-Annotated Corpus for Sarcasm" (allegato alla traccia), e dopo una fase di verifica, posso affermare che il dataset presenta questa caratteristica e quindi non è necessario applicare tecniche per migliorare il bilanciamento.

Figura 1.1: Distribuzione del target ottenuta dopo la fase di verifica, come si può notare il bilanciamento è quasi perfetto

Inoltre, questa analisi mi è servita per calcolare con precisione $S_{IR} \simeq 0.5001$.

1.2 Analisi delle feature contestuali

Come affermato precedentemente si definiscono tali: Parent, Date, Subreddit e Author. Esse potrebbero essere utili per la predizione aggiungendo delle informazioni di contesto al commento. Nel dettaglio:

- Il Parent: codificata come una stringa a più parole, rappresenta il testo della discussione a cui il commento è stato lasciato; potrebbe essere molto utile per rilevare significati contrastanti con il testo (caratteristica principale dell'ironia), oppure per individuare discussioni polarizzate sarcastiche o no.
- Data: codificata come una stringa "anno-mese", potrebbe avere rilevanza nel trovare date altamente sarcastiche (o no).
- Subreddit: codificata come una stringa ad una parola, rappresenta il nome del subreddit in cui è stato postato il commento; potrebbe essere utile per trovare subreddit tendenzialmente più o meno sarcastici.
- Author: codificata come una stringa ad una parola, rappresenta il nome dell'autore del commento; potrebbe essere utile per trovare autori più o meno sarcastici.

A partire da queste premesse ho effettuato un'analisi per avere un reale riscontro sui dati.

1.2.1 Analisi del contesto rispetto al numero di commenti

In questa fase ho analizzato per ogni feature quanti sono gli elementi che si ripetono (prendendoli una sola volta) e ad ognuno quanti commenti sono associati.

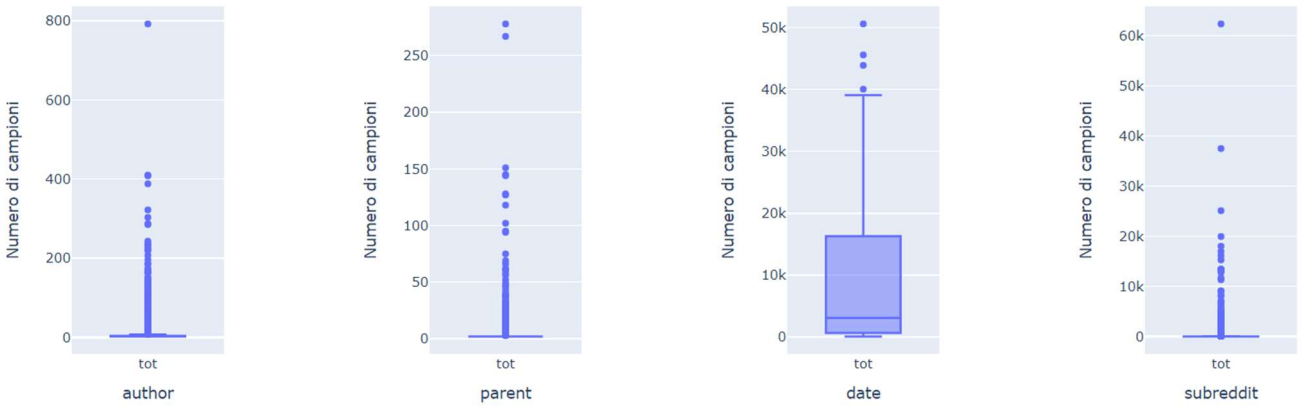


Figura 1.2: Distribuzione, tramite boxplot, degli elementi unici del contesto rispetto al numero di commenti associati

feature	Elementi unici (%) associati ad un unico testo	Elementi unici totali (%)	Elementi unici totali
author	7.03	26.67	256071
parent	98.38	97.28	933914
date	0	0.01	96
subreddit	40.65	1.52	14545

Figura 1.3: Tabella mostrante il numero di elementi unici (e che si ripetono quindi in più commenti), e il numero di testi associati

Da Figura 1.2 e Figura 1.3, si nota particolarmente che:

- Quasi ogni commento si riferisce ad un Parent unico; infatti, nel boxplot la distribuzione è "schiacciata" verso l'1 (il 1°, 2° e 3° quartili sono a 1), e nella tabella il 97.3% dei Parent si ripete almeno una volta, di cui il 98.4% si ripete esattamente una volta.
- Le date, come atteso essendo solo 96, si ripetono con un'alta frequenza nei commenti; infatti, nel boxplot i valori si distribuiscono principalmente in un range dai 668 a 16000, con mediana a 3108.
- La feature autore potrebbe essere rilevante; in quanto il numero di autori unici rispetto ai commenti è del 26.7% di cui solo il 7% ha un solo commento, come mostrato nella tabella; il boxplot conferma la tesi, infatti, la distribuzione è posta nel range 1, 4
- La feature subreddit, con una distribuzione nel range 1, 9 e un numero di elementi unici a 1.5% (14545 rispetto ai 960000), risulta ancora più interessante; vale la pena notare però che il 40.6% di essi ha un solo commento.

1.2.2 Analisi del rateo informativo del contesto singolarmente

Dopo aver analizzato come si distribuiscono gli elementi unici del contesto, procederò con l'analisi del contenuto informativo di quegli elementi che hanno almeno più di un commento associato. Approfondendo ogni feature separatamente, tramite degli istogrammi che presentano, per un range di commenti associati, la media del rateo.

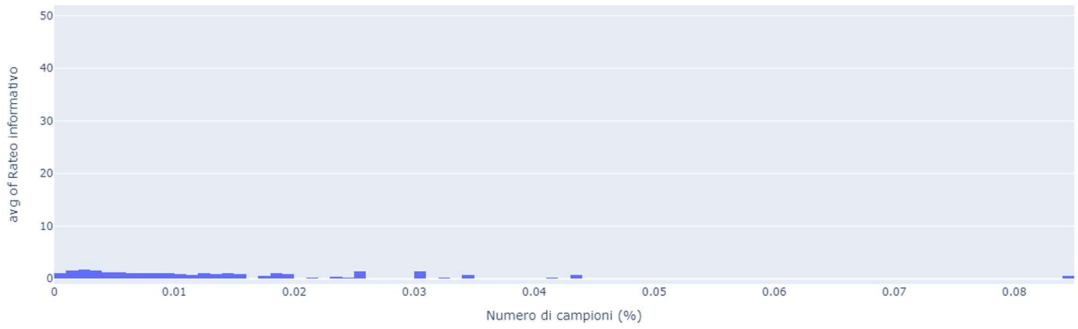


Figura 1.4: Rateo informativo per gli elementi di Autore.

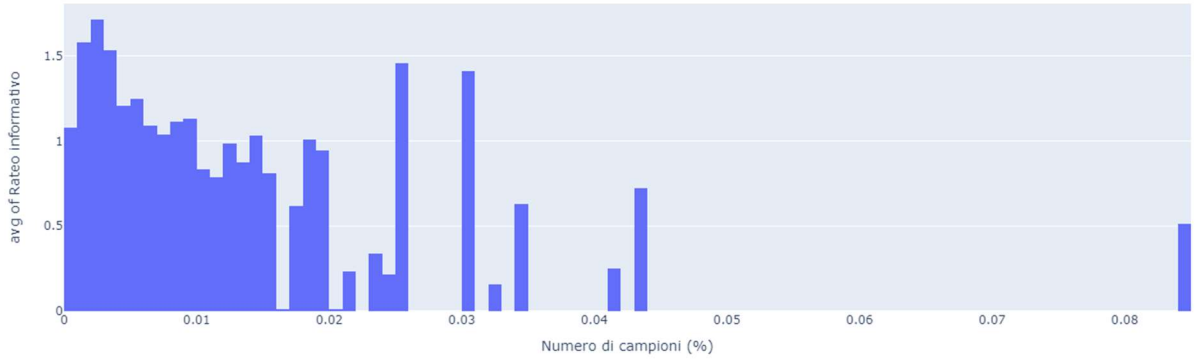


Figura 1.5: Rateo informativo per gli elementi di Autore. Con y scalata

Come si può notare in Figura 1.4 e Figura 1.5 il rateo informativo di autore è molto basso; infatti, è necessario scalare l'asse y tra 0 e 1.8 per poter distinguere gli istogrammi. Ne consegue che non possiamo distinguere molti autori a maggioranza sarcastica o no, e quindi che questa feature potrebbe essere irrilevante.

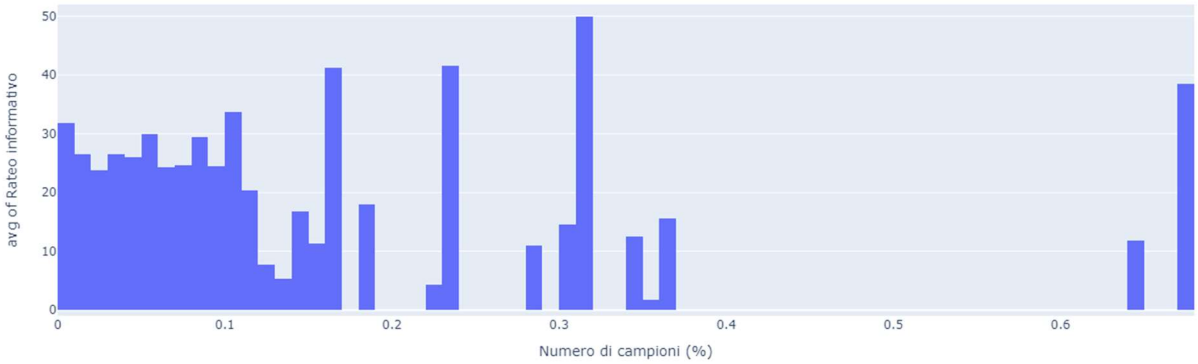


Figura 1.6: Rateo informativo per gli elementi di Parent

Dalla Figura 1.6: Notiamo che il rateo informativo per gli elementi di Parent è alto; tuttavia, come si vede nell'asse x (e come conferma la Figura 1.3) il numero di commenti associato è basso.

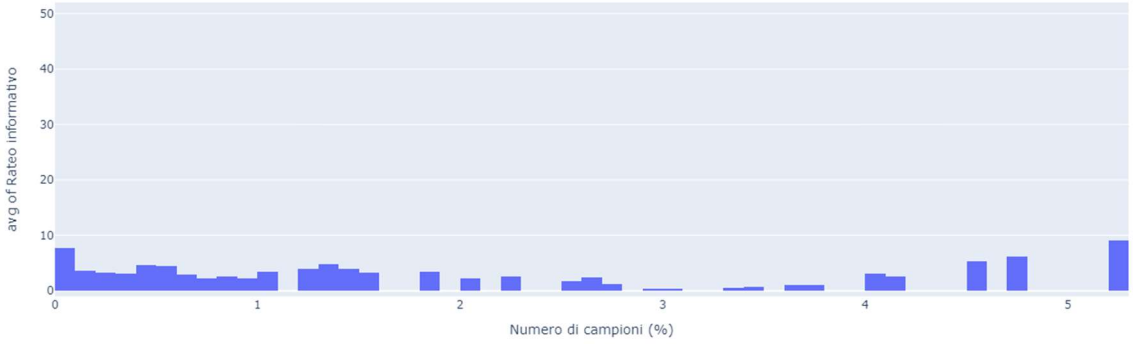


Figura 1.7: Rateo informativo per gli elementi di date

Come si può notare dalla Figura 1.7 il rateo informativo è abbastanza scarso, anche se il numero di campioni per elemento è abbastanza alto.

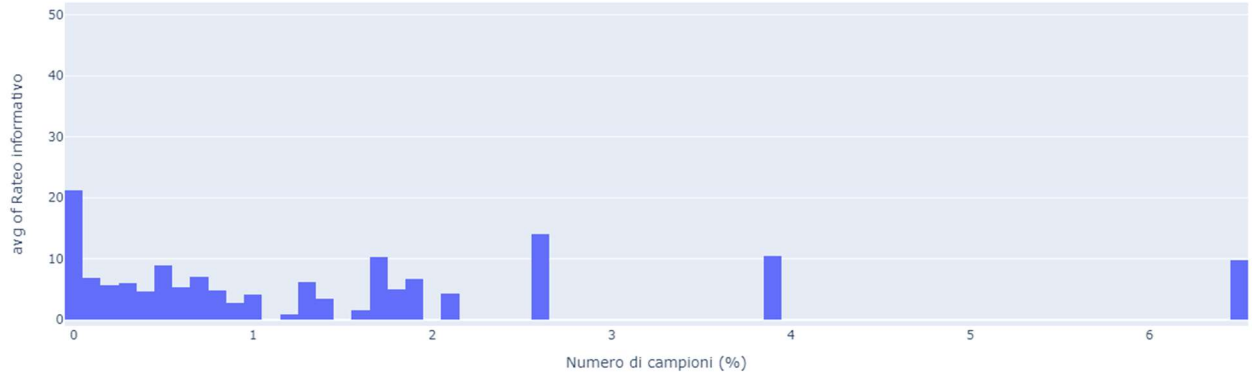


Figura 1.8: Rateo informativo per gli elementi di Subreddit

Dalla Figura 1.8 possiamo notare che il rateo informativo per Subreddit è decisamente migliore rispetto a quello di date, raggiungendo e superando anche il 10% in molti casi; come atteso, notiamo però, che il rateo cala all'aumentare del numero di commenti.

1.2.2 Analisi aggregata del rateo informativo

Dopo aver visto le distribuzioni singolarmente del rateo informativo, è utile aggregare i risultati in modo da poterli confrontare in modo efficiente. Per farlo ho seguito due approcci:

- Threshold: in cui ho calcolato la percentuale degli elementi unici che supera delle soglie (10%, 20%, 30%, 40%) di rateo informativo.
- Media: in cui ho calcolato la media e deviazione standard pesate sul numero di commenti associati.

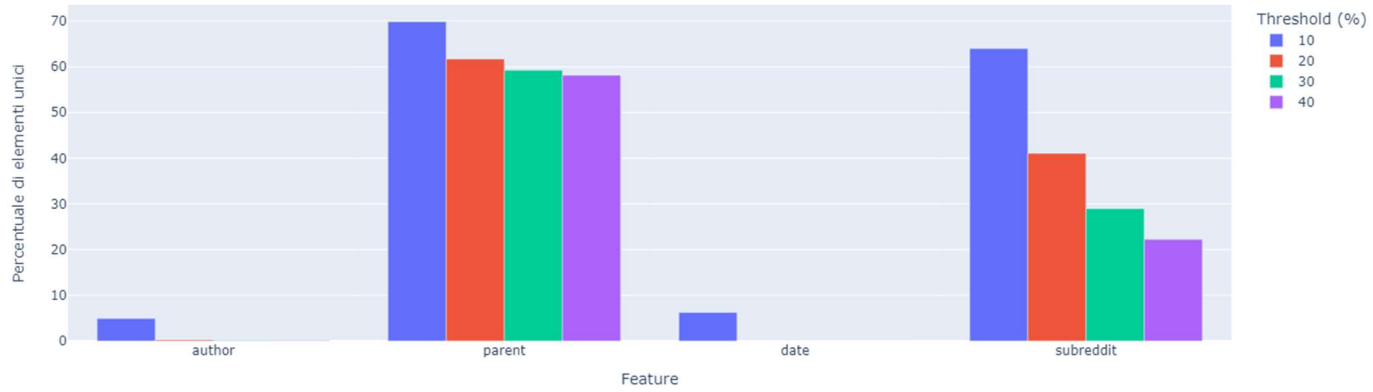
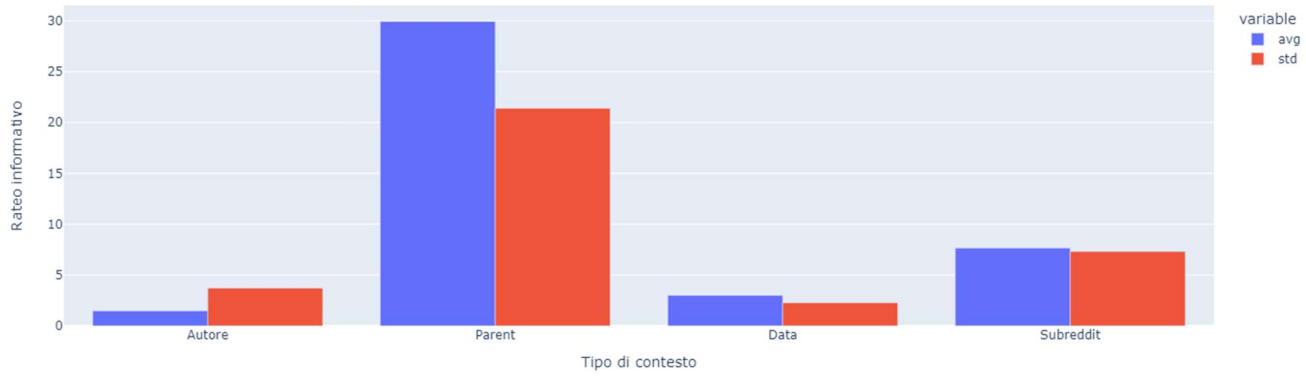


Figura 1.9: Threshold: Percentuale di elementi unici che supera determinate soglie, per ogni feature del contesto

Figura 1.10: Media e STD pesate del rateo informativo per i tipi di contesto



Possiamo notare da Figura 1.9 e **Error! Reference source not found.**:

- Data e Autore hanno un basso rateo informativo medio, e pochi superano la soglia del 10%.
- Parent ha il rateo informativo medio più alto, e quasi tutti quelli che superano la soglia del 10%, superano anche quella del 40%; questo, significa che i pochi elementi ripetuti sono molto significativi.
- Subreddit ha un rateo informativo accettabile, con ben il 64% degli elementi che supera il 10% e il 22% che supera la soglia del 40%.

1.2.3 Conclusioni delle analisi sul contesto

Unendo le analisi fatte precedentemente possiamo giungere a delle conclusioni utili per il modello:

- Autore presenta non troppi elementi unici (associati quindi ad abbastanza commenti), tuttavia essi non hanno abbastanza rateo informativo; perciò, introdurrebbero troppo rumore nell'addestramento del modello.
- Data come autore, ha davvero pochi elementi unici ma questi non hanno un rateo informativo significate; perciò, introdurrebbe anche lui troppo rumore, e sarebbe totalmente inutile con commenti in un periodo temporale successivo.
- Parent al contrario ha un alto rateo informativo, ma il numero di elementi unici è pari quasi al numero di commenti; quindi, l'uso dell'intera frase in modo statistico non avrebbe alcun senso, però potrebbero essere utili le associazioni delle parole fra di loro o con il testo del commento.
- Subreddit presenta un buon rateo informativo e un numero adeguato di elementi unici; perciò, potrebbe essere una feature utile al modello.

In conclusione, la feature Subreddit verrà usata dal modello, Data e Autore non verranno usate e Parent verrà usata solo in legame con il testo.

1.3 Analisi del testo

Per l'analisi del testo del commento ho adottato una logica simile a quella per il contesto, ma al posto di paragonare le diverse feature, ho confrontato i vari modi di elaborare il testo.

Ho fatto tale analisi, con quattro grafici prendendo singolarmente ogni tipo di testo:

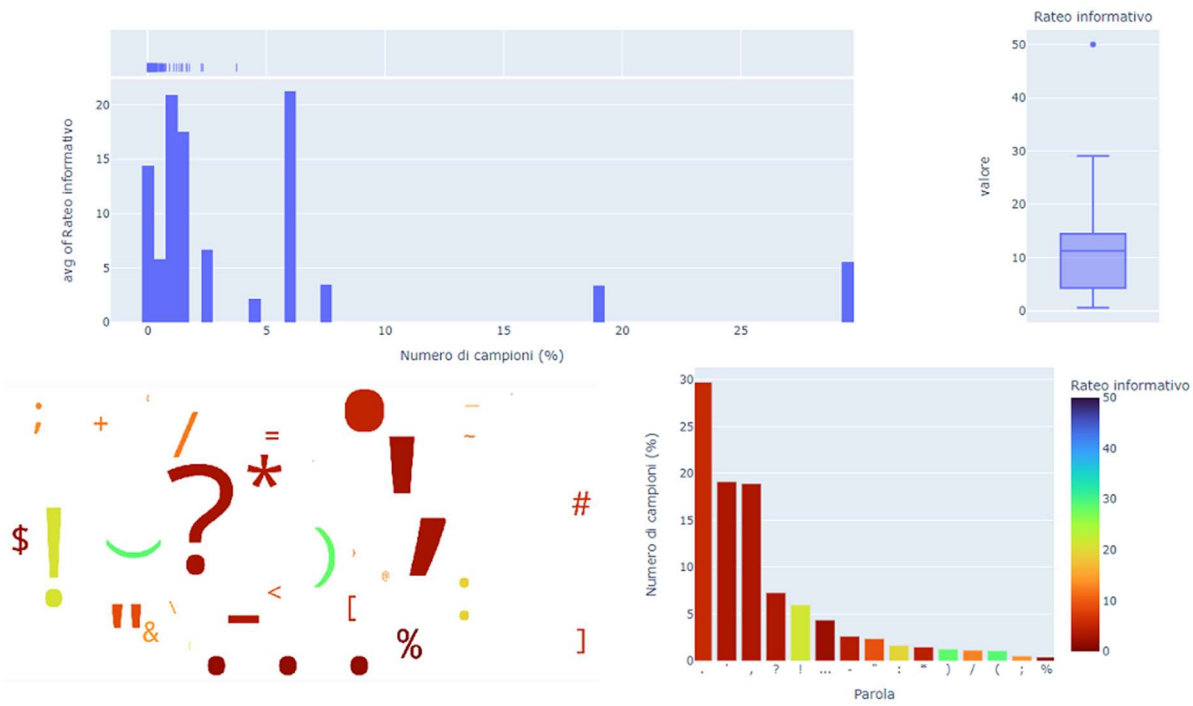
- Istogramma: come quello usato nell'analisi del contesto, mostra il rateo informativo associato ad un range di frequenze dei token nei commenti.
- Boxplot: mostra in sintesi la distribuzione del rateo informativo.
- Wordcloud: Grafico che mostra i 200 token più frequenti; la grandezza indica la frequenza, mentre il colore indica il rateo informativo.
- Barplot: mostra in ordine i token più frequenti e il loro rateo informativo.

I vari tipi di testo, elaborati ed analizzati, sono:

- Normale
- Senza stopwords: a cui sono state rimosse le parole non chiave; come articoli, congiunzioni, ecc...
- Con stemming: in cui le parole derivate e flesse vengono ridotte alla loro forma radice.
- Senza stopwords e con stemming: in cui si applicano simultaneamente i due approcci precedenti.

Però prima dell'analisi è necessario tokenizzare il testo, ovvero convertirlo da una stringa unica in una lista di parole (token); per poi rimuovere la punteggiatura, in quanto spesso non è significativa e comporta solo del rumore. Tuttavia, al posto di rimuovere tutta la punteggiatura ho effettuato l'analisi del rateo informativo dei punti per mantenere quelli con un rateo significativo, in modo da non avere eccessive perdite di informazione.

Figura 1.11: Distribuzione del rateo per i simboli punteggiatura



Come si può osservare dalla **Error! Reference source not found.** l'unico segno di punteggiatura significativo è il "!"; perché presenta un rateo di 21 (infatti le frasi con almeno un "!" sono sarcastiche il 71% delle volte), con una frequenza circa del 6%. Dunque, ho

Figura 1.12: Distribuzione del rateo per i token del testo normale

deciso di mantenere questo punto e rimuovere gli altri, per poi procedere all'analisi singola dei tipi di testo.

Come si può notare dalla Figura 1.12, la maggior parte del rateo informativo è distribuito sugli elementi a bassa frequenza, e quindi non molto usabili; al contrario, gli elementi più frequenti, che sono principalmente stopwords non portano molte informazioni, anche se sono distinguibili sullo sfondo della wordcloud dei token utili.

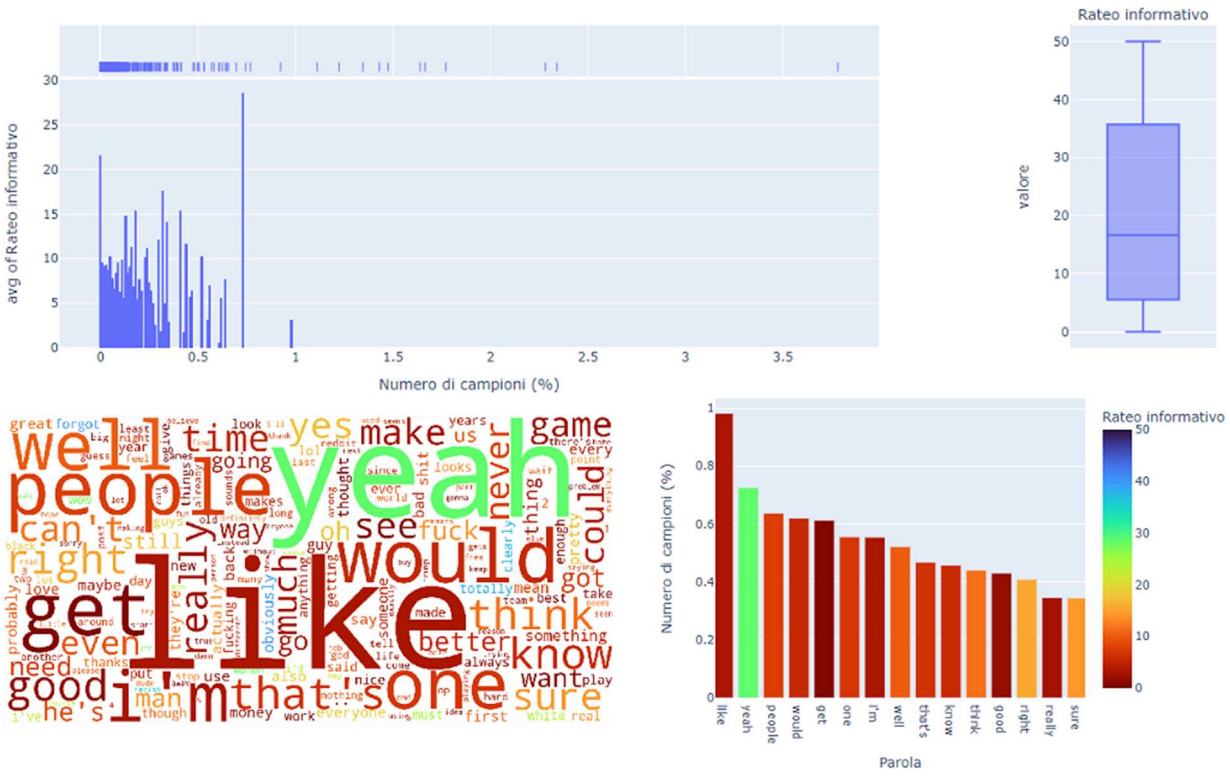
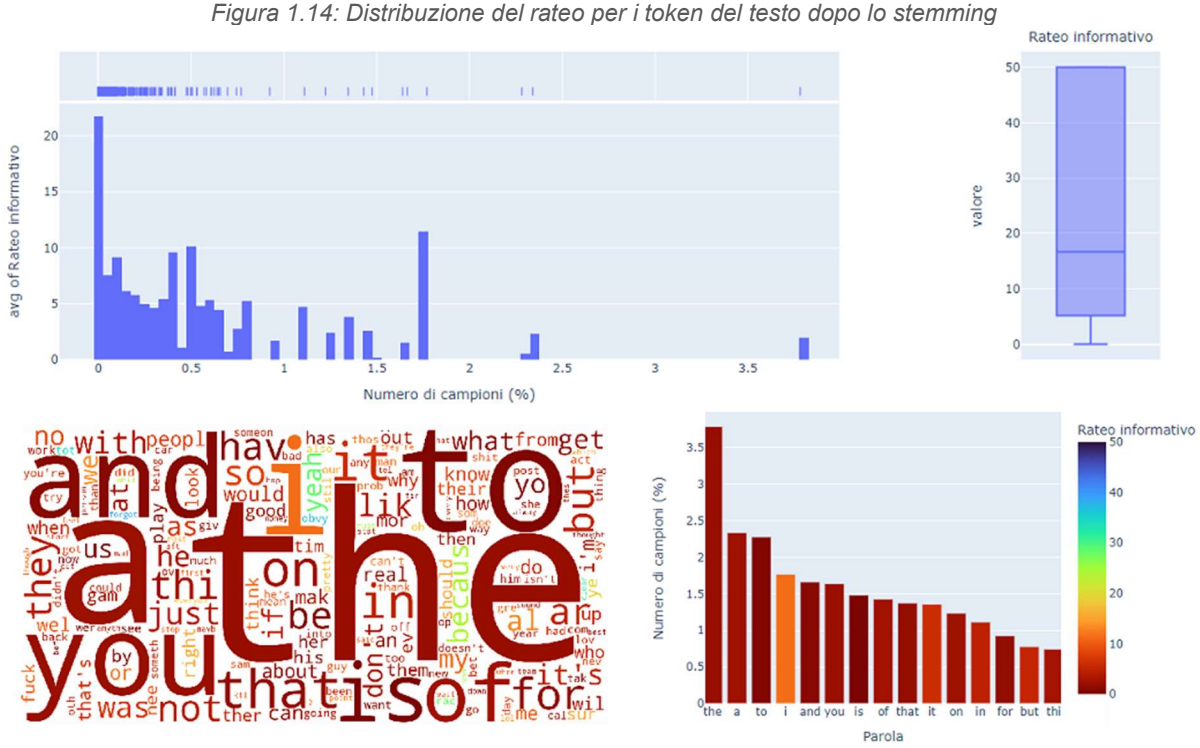


Figura 1.13: Distribuzione del rateo per i token del testo senza stopwords

Osserviamo che in Figura 1.13 la frequenza dei token più comuni è scesa notevolmente, passando da più di 3.5% a quasi 1%, ma il rateo di questi token è decisamente maggiore; infatti, notiamo sia nel wordcloud sia nel barplot che i colori sono meno tendenti al rosso scuro (e quindi hanno un maggiore rateo).

Figura 1.14: Distribuzione del rateo per i token del testo dopo lo stemming



Analizzando meglio la Figura 1.14 si può notare come lo stemming non abbia avuto un buon impatto e infatti il rateo informativo negli elementi più frequenti è sceso notevolmente; invece il boxplot mostra che il range si è allargato, questo a causa del fatto che il numero complessivo di elementi è diminuito.

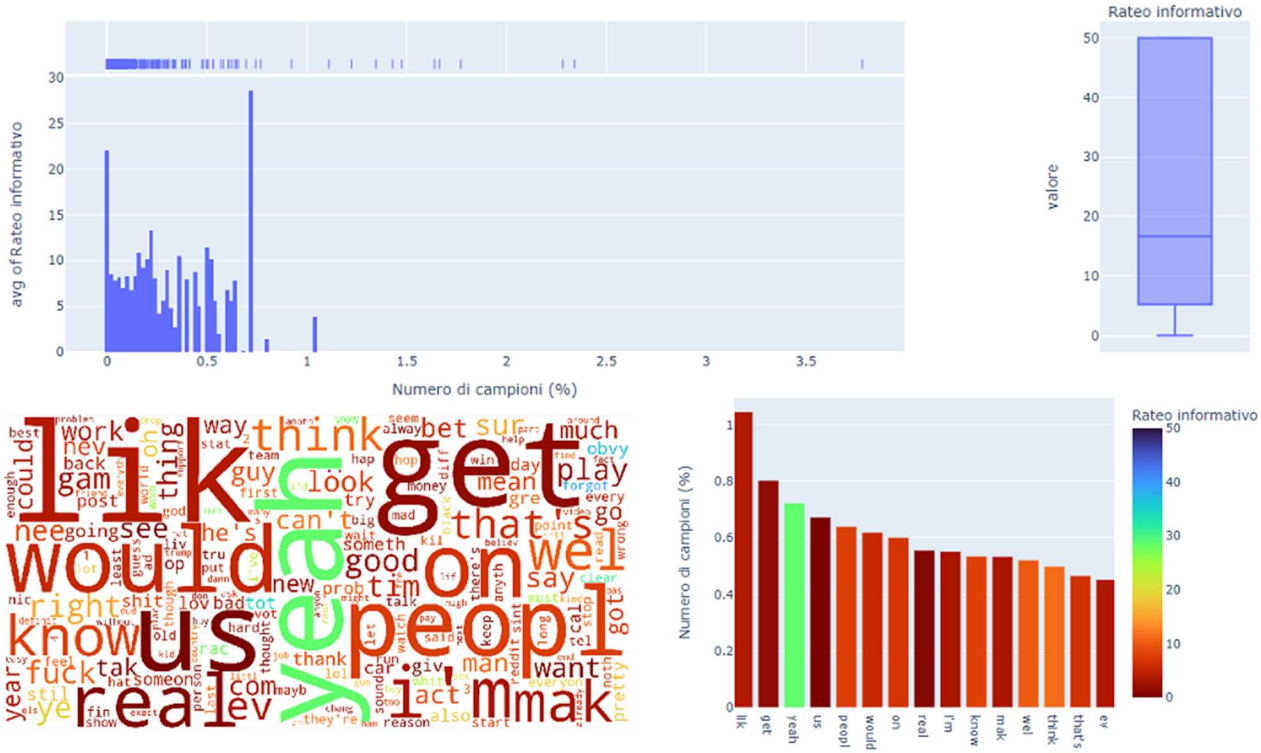


Figura 1.15: Distribuzione del rateo per i token del testo senza stopwords e con stemming

In Figura 1.15 notiamo infine che l'applicazione dello stemming al testo senza stopword non produce un risultato sensibilmente diverso; tuttavia, nella wordcloud sembra che il rateo sia sceso nei token a maggiore frequenza; invece, nel boxplot sembra globalmente aumentato (perché il terzo quartile si trova più in alto).

Dopo aver analizzato in modo singolo i tipi di testo, è utile effettuare un'analisi aggregata, calcolando media e deviazione standard pesate (come precedentemente fatto per il contesto).

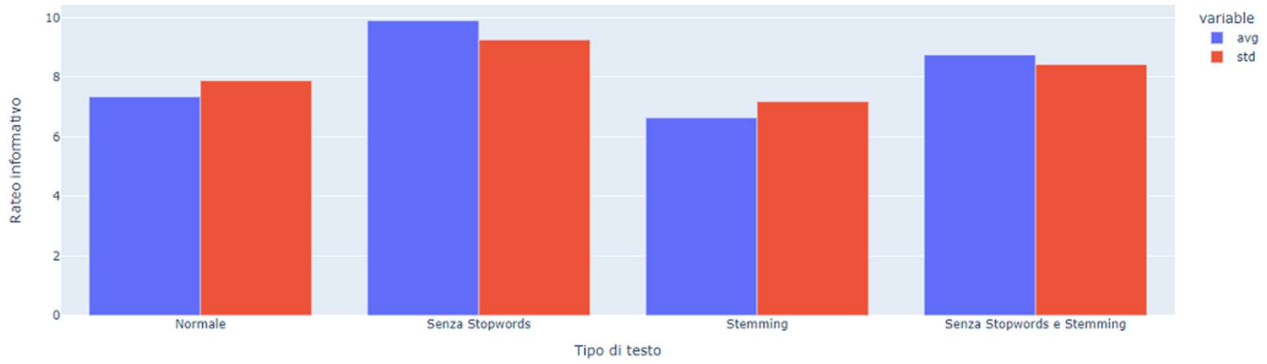


Figura 1.16: Media e STD pesata del rateo informativo dei tipi di testo

La Figura 1.16 ci fornisce la vista d'insieme utile per decidere quale tipo di testo usare. Come visto nelle analisi singole la rimozione delle stopwords aumenta la qualità del testo, mentre, l'uso dello stemming è più relativo; infatti, i tipi "Senza stopwords" e "Senza stopwords e Stemming" hanno delle medie di 9.89% e 8.74%, quindi, vista la poca differenza, la scelta del tipo di testo (tra le due) è più arbitraria. Io ho scelto il tipo "Senza stopwords" per due motivi: la media è più alta; e, Il modello adottato applica un preprocessore e degli embedding pre-addestrati, che dovrebbero essere in grado di sfruttare anche le informazioni che lo stemming andrebbe a rimuovere.

Dopo aver analizzato i vari tipi di testo dei commenti, possiamo processare in modo uguale anche il testo dei parent; in quanto, essendo anche loro commenti si assume che siano molto simili.

1.4 Analisi delle lunghezze di Testo e Parent

Un altro possibile elemento informativo utile potrebbe essere la lunghezza (nel senso di numero di parole) del testo e del parent. Infatti, tale informazione si va a perdere a causa delle operazioni di preprocessing e di padding (operazione, richiesta dal modello, che porta tutte le frasi alla stessa lunghezza). Essa potrebbe essere utile, in quanto, non rappresentano una grossa quantità di dati e servirebbero ad evidenziare contrasti e non tra testo e parent.

Prima di poter inserire tale elemento nel modello è necessaria una breve fase di analisi, che ho realizzato esaminando le distribuzioni delle lunghezze, le distribuzioni del rateo informativo a diverse lunghezze; terminando poi con il calcolo del rateo informativo alle combinazioni delle lunghezze delle due feature.

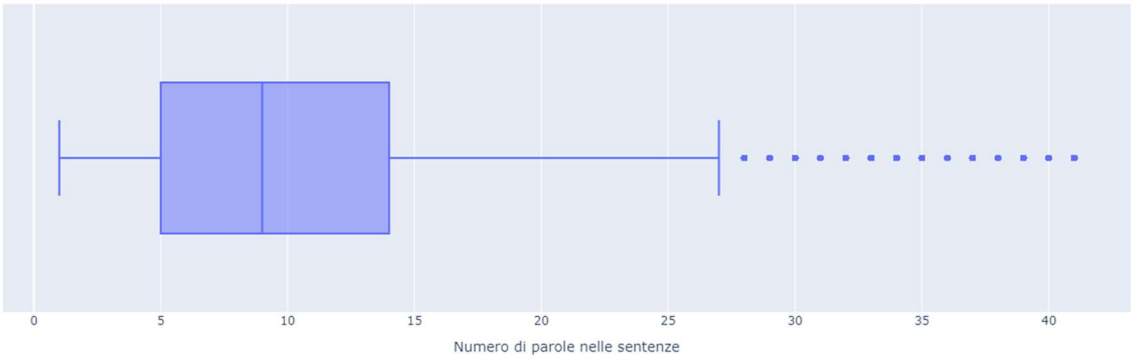


Figura 1.17: Boxplot rappresentante la distribuzione della lunghezza del parent

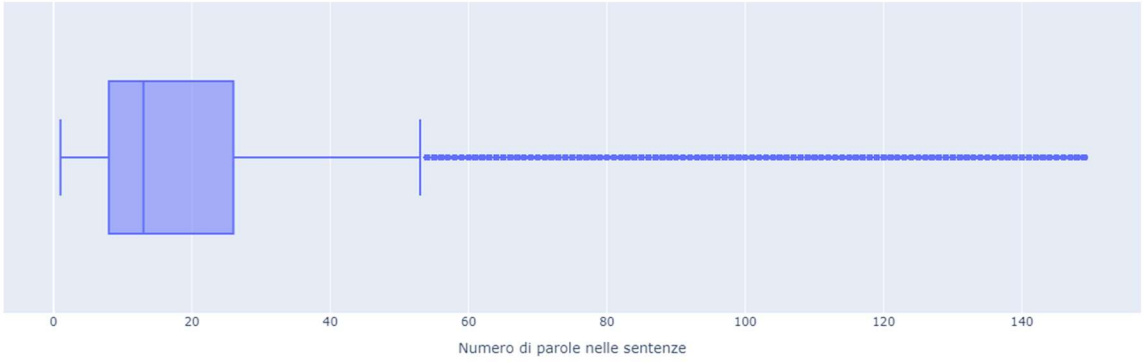


Figura 1.18: Boxplot rappresentate la distribuzione della lunghezza del parent

Come si può notare, da Figura 1.17 e Figura 1.18, il parent è tendenzialmente più lungo del testo; infatti, il parent ha come primo quartile, mediana e secondo quartile rispettivamente: 8, 13, 26; mentre, il testo ha: 5, 9, 14. Inoltre, se andiamo a considerare gli estremi della distribuzione e gli outliers la differenza è ancora più evidente.

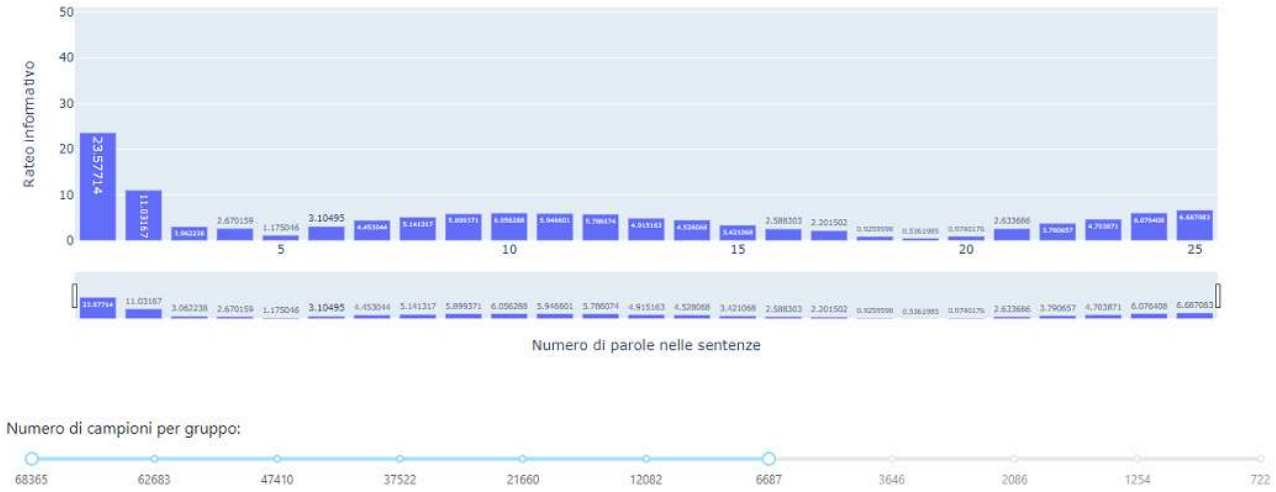


Figura 1.19: Istogramma rappresentate la distribuzione del rateo informativo per la lunghezza del testo, nota: si filtrano solo le lunghezze con una frequenza significativa (>40% della distribuzione)

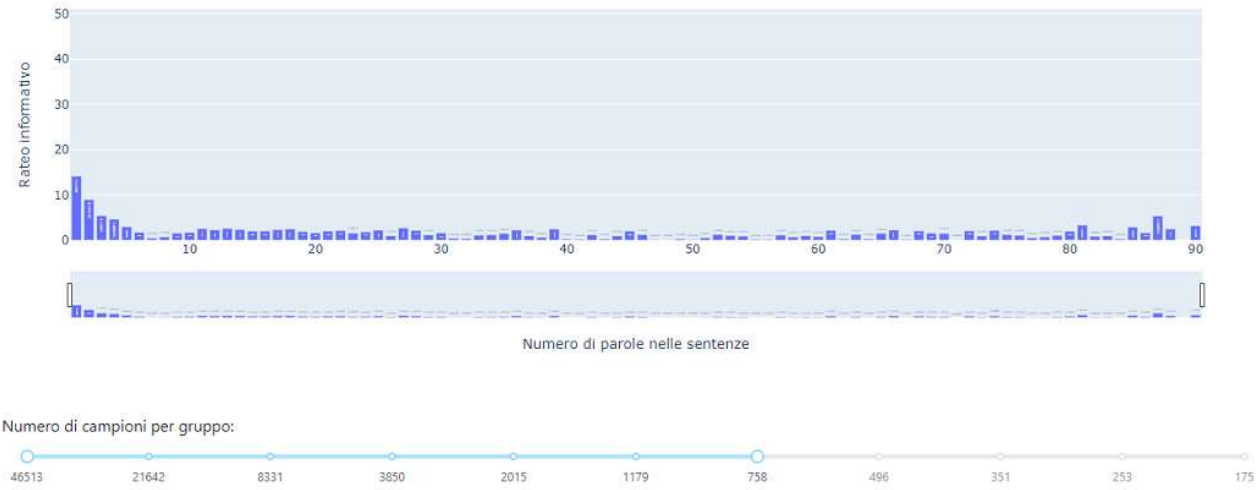


Figura 1.20: Istogramma rappresentante la distribuzione del rateo informativo per la lunghezza del parent. Nota: si filtrano solo le lunghezze con una frequenza significativa (>40% della distribuzione)

Confrontando la Figura 1.19 e la Figura 1.20, notiamo che: il parent ha una lunghezza molto più variabile (come visto precedentemente nella Figura 1.18); la lunghezza del testo ha generalmente un rateo informativo maggiore; in entrambi i casi abbiamo un rateo informativo maggiore con lunghezze basse.

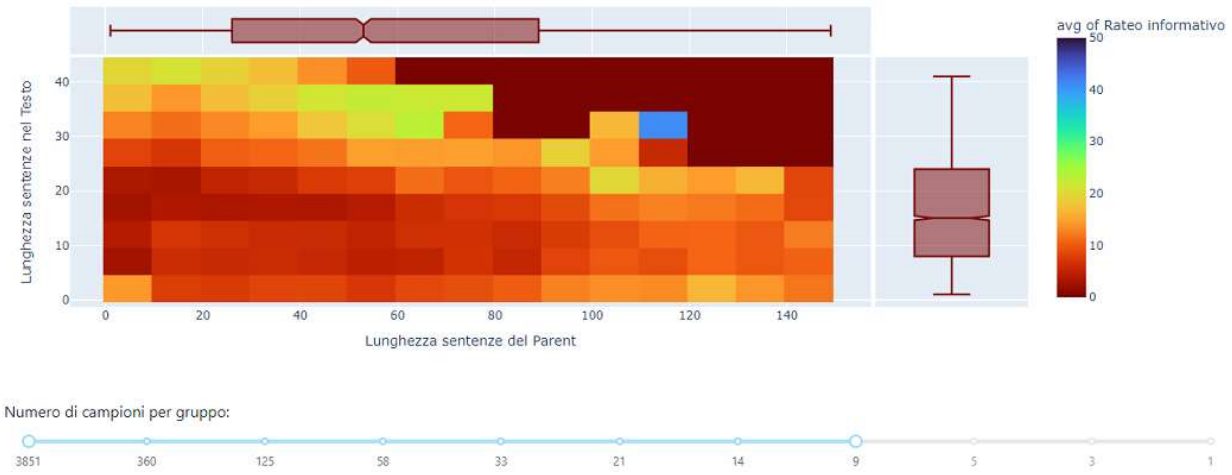


Figura 1.21: Heatmap che rappresenta sulle celle il rateo informativo medio delle lunghezze associate del testo e del parent

Dalla Figura 1.21 notiamo che:

- L'Heatmap è stata filtrata per eliminare le lunghezze con frequenze non significative (>40% della distribuzione).
- È presente (come atteso), nella parte in alto a destra un'area a rateo informativo 0, in quanto a quei valori i campioni sono stati filtrati.
- Il rateo informativo medio non è molto alto; ma, si può individuare un'area con lunghezze tra 40 e 80 per il parent e 30 e 40 per il testo dove il rateo supera il 20%.
- È presente un'area ad alto rateo (40%) che tuttavia è poco significativa, in quanto ha un numero di campioni basso (aumentato il filtro infatti sparisce).

Quindi a seguito dell'analisi, le due lunghezze non hanno un rateo molto alto; tuttavia, non richiedendo molti parametri da parte del modello, e quindi non rischiano di aggiungere troppo rumore, potrebbero comunque essergli utili.

Capitolo 2: Modello

In questo capitolo descriverò il modello utilizzato, illustrando prima, a livello teorico, il modello e le ragioni dietro la mia scelta; successivamente approfondirò l'architettura neurale impiegata; per poi presentare i risultati ottenuti.

2.1 Presentazione breve sui Transformer

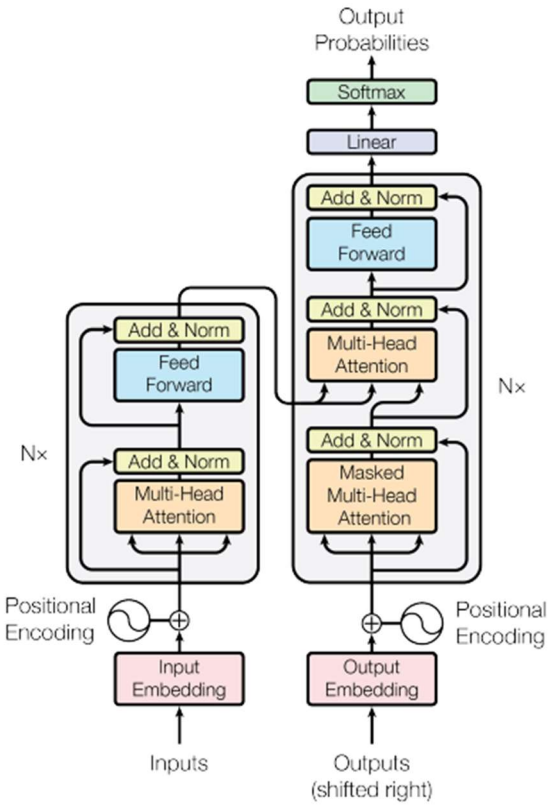


Figura 2.1: Architettura generica di un transformer

I modelli Transformer sono stati introdotti nel 2017 con il paper "Attention Is All You Need" da Vaswani et al. Rappresentano un'architettura di rete neurale che si basa fortemente sull'uso di meccanismi di attenzione. A differenza dei modelli RNN e LSTM (che erano lo standard precedente), i Transformer non dipendono da una struttura sequenziale, ma utilizzano l'attenzione per catturare le relazioni a lungo raggio tra le parole all'interno del testo, questo li rende particolarmente adatti per l'elaborazione di sequenze di testo di lunghezza variabile.

È composto da due componenti fondamentali: l'encoder, che elabora il testo in input e crea una rappresentazione contestuale delle parole; il decoder invece, prende in input tale rappresentazione e genera una sequenza di parole tradotta o generata.

I modelli basati su Transformer sono promettenti per task come quello del sarcasmo a causa delle loro capacità di catturare e modellare le relazioni contestuali tra le parole, anche in testi molto lunghi.

2.2 BERT e TransformerEncoder

Nel mio modello ho adottato due modelli della classe dei transformer (BERT e TransformerEncoder), essi vengono usati per elaborare il testo e il parent simultaneamente, in modo da rilevare legami e contrasti, e sono posti uno dopo l'altro.

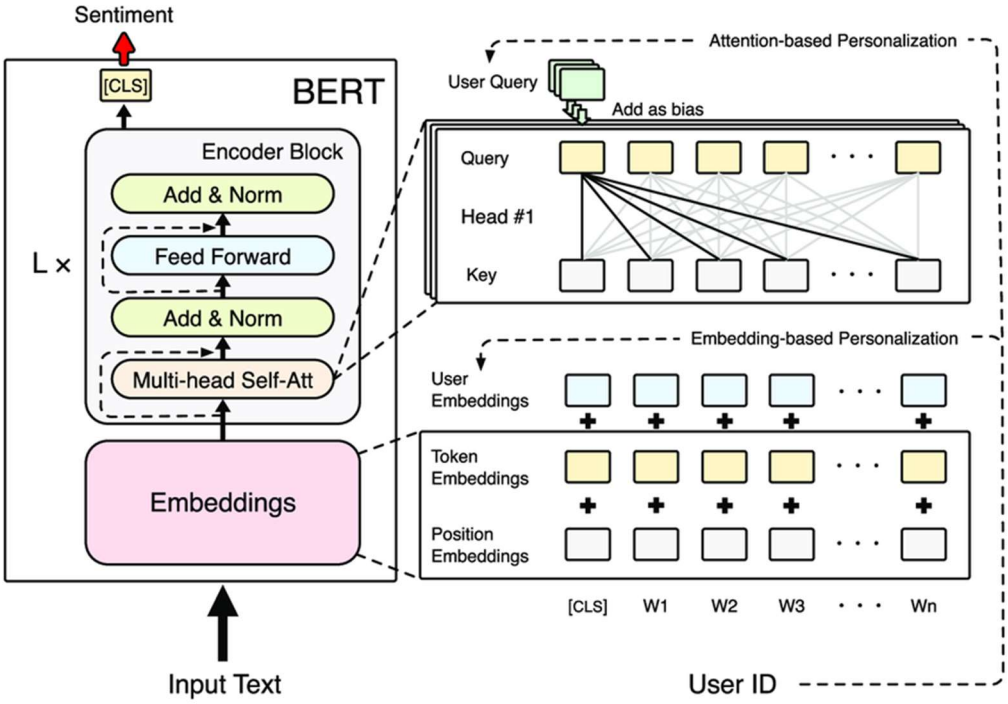


Figura 2.2: Schema illustrativo di BERT

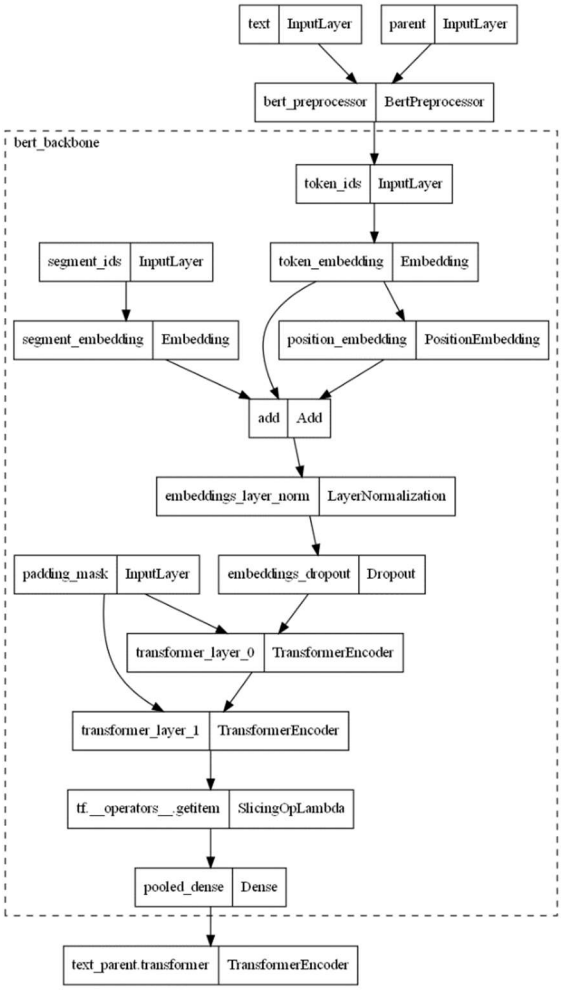
BERT (Bidirectional Encoder Representations from Transformers) è un modello pre-addestrato basato sui Transformer che utilizza una serie di encoder per estrarre la rappresentazione del testo. In particolare, prende in input una o due sentenze (in questo caso testo e parent), e dopo aver effettuato l'embedding (rappresentazione distribuita delle parole) processa le rappresentazioni usando multipli blocchi sequenziali di encoder (come quelli del transformer). Questo modello viene addestrato in due fasi:

- Pre-training: in questa prima fase bisogna pre-addestrare il modello su una grande quantità di dati, senza label, in due modalità:
  - Masked Language Model (MLM): in cui si mascherano delle parole casuali e il modello dovrà ricostruirle dal contesto
  - Next Sentence Prediction (NSP): in cui il modello dovrà prevedere se due frasi sono consecutive o no nel testo originale.
- Fine-tuning: dopo il pre-addestramento si rendono statici (non addestrabili) i parametri del modello inziale, e si aggiungono dei layer che vengono addestrati per usare l'output di BERT al fine del task in questione.

Ho usato BERT perché, essendo pre-addestrato, si possono sfruttare le sue capacità, derivate da enormi quantità di dati e di risorse computazionali, senza addestrarlo da zero. Inoltre, è possibile usarlo sfruttando l'output di una o di entrambe le modalità; nel mio caso, estraendo la dimensione della rappresentazione corrispondente al token CLS (visibile in Figura 2.1), sfrutto la NSP che è più adatta al task del sarcasmo, in quanto, racchiude i legami semantici fra le due frasi.

Anche TransformerEncoder è un modello uguale all'encoder dell'architettura Transformer, che ho utilizzato per il Fine-tuning. L'ho addestrato per prendere in input la rappresentazione in output di BERT, e la rielaborarla per estrarre le informazioni utili al modello per prevedere il sarcasmo.

Figura 2.3: BERT utilizzato



2.1 Il modello usato

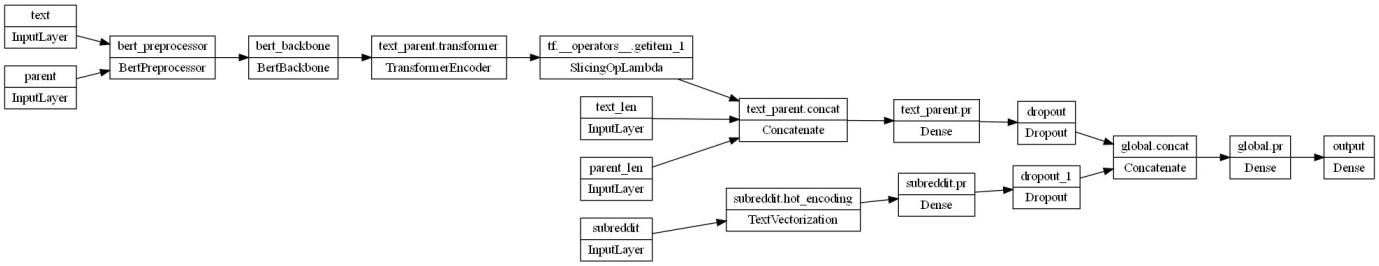
Il modello usato è divisibile in due parti:

- La prima è quella di sfrutta BERT
- La seconda concatena il risultato della prima alle feature subbreddit, text_len e parent_len

La prima parte si basa sull'utilizzo di BERT per processare simultaneamente le due feature più importanti: testo e parent. Esso è composto da: un Preprocessor, che prende in input i testi, e li tokenizza in modo che siano compatibili con il modello; e il Backbone che prende i token delle due sentenze ed estrae le rappresentazioni.

Tale componente è possibile scaricarlo dalla libreria di keras, nel mio caso, ho usato il "bert_tiny_en_uncased" che non supporta caratteri maiuscoli, ed è stato addestrato su wikipedia inglese e su BookCorpus. Di quelli possibili è il più piccolo, infatti ha "solo" 4.39M di parametri, mentre il più grande ha ben 335.14M di parametri.

Figura 2. 4: Rappresentazione del modello usato; in cui si possono vedere i vari layer con i loro nomi



La prima parte, infine, restituisce le rappresentazioni al TransformerEncoder chiamato text_parent.transformer che, composto da 4 teste per un totale di 131K parametri effettua il fine-tuning. Il risultato dell'encoder sarà passato poi alla seconda parte.

La seconda parte elabora separatamete il subreddit, effettuando prima il multi-hot encoding per poi elaborare il risultato con un layer di 25 neuroni (100K parametri), ottenendo un vettore a lunghezza 25. In particolare l'hot-econdig viene così effettuato:

- Nella fase di adapt (che precede il training) crea un vocabolario dei nomi più frequenti;
- La dimensione è impostata a 4000, che è due terzi del numero di elementi unici con più di un testo (come calcolabile dalla Figura 1.22: 14545 * 0.4)
- Per ogni nome in input restituisce un vettore di 4001 elementi, con un solo 1 la cui posizione è la stessa che ha nel dizionario (c'è un dimensione aggiuntiva per i valori esterni)

Inoltre la seconda parte concatena il risultato della prima parte con le lunghezze del testo e del parent, ottenendo un vettore di 130 valori (di cui 2 sono le lunghezze); per poi applicare un layer di 100 neuroni (13K parametri), con cui si ottiene un vettore a lunghezza 100. Successivamente tale risultato viene concatenato con il risultato del subreddit, e poi passato ad un primo layer con 20 neuroni (2520 parametri). Che infine arriva all'ultimo neurone, che con la funzione sigmoid, restituisce la probabilità di appartenere alla classe sarcastica.

2.3 Problemi affrontati durante il training

Questo modello, per quanto faccia leva sul BERT, a causa del numero totale di 246K parametri e del quasi un milione di righe è stato eterno da addestrare. Perciò ho deciso di proposito di concentrarmi su un solo modello, senza effettuare tuning degli iperparametri e scegliendoli al meglio dalle analisi effettuate e da alcuni test performati.

L'addestramento è stato possibile solo grazie a Colab (cloud di Google), che mette a disposizione delle GPU Tesla T4 che sono decine di volte più performati delle top di gamma comuni di Nvidia (RX e GTX). Con le T4 una singola epoca impiegava poco più di un'ora e 30 minuti; mentre con il processore Intel I7-6700k, del mio computer fisso, i tempi erano di svariati giorni.

Capitolo 3: Risultati

In questo capitolo analizzeremo i risultati che ho ottenuto nel training del modello; prima vedremo l'history di training, successivamente i risultati sul dataset di validation ed infine sul dataset di testing (fino ad ora rimasto intoccato).

3.1 History di training

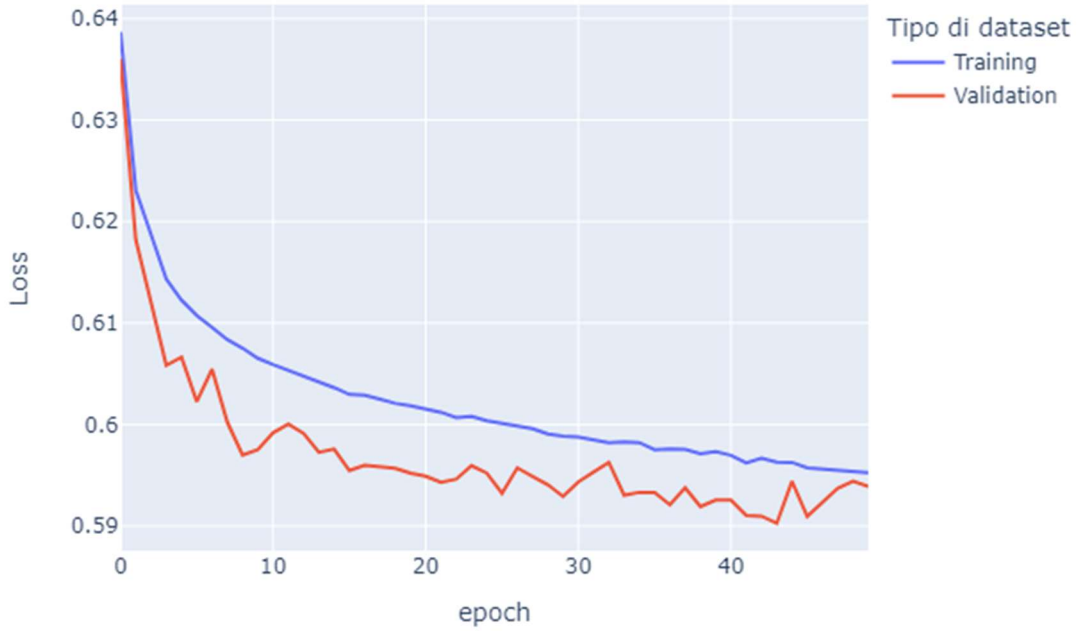


Figura 3.1: Funzione di Loss durante il training, in cui si confronta i valori sui dataset di training e di validation.

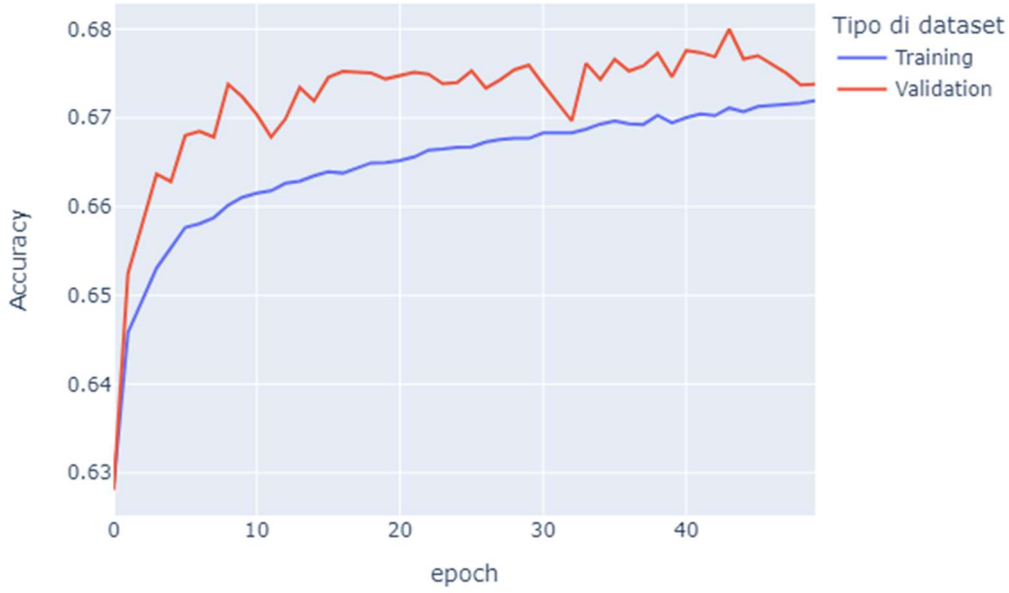


Figura 3.2: Metrica Accuracy durante il training, in cui si confronta i valori sui dataset di training e di validation.

Osservando Figura 3.1 e Figura 3.2, possiamo osservare che:

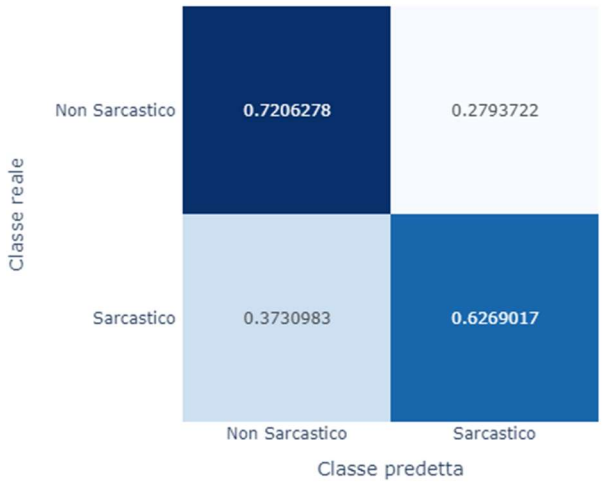
- Il modello dopo training, durato complessivamente 50 epoche, ottiene una precisione del 67.2% (a partire dal 62.8%) sul training, e del 68.0% (a partire da 62.8%) sul validation.
- Entrambe le funzioni di Loss scendono gradualmente:
    - Infatti, il callback Earlystopping impostato, che blocca l'addestramento dopo 5 epoche consecutive senza un miglioramento, non si è mai attivato;
    - Quindi il modello non va sicuramente in overfitting.
- Vista la lenta discesa è probabile che il modello vada in underfitting, ciò è risolvibile:
    - Eliminando i layer di Dropout, che introducono una componente randomica utile a generalizzare.
    - Aumentando i parametri del modello, specificatamente quelli posti nel text_parent.transformer o quelli negli ultimi layer.
    - Proseguendo l'addestramento per più epoche.

3.2 Risultati sul dataset Validation

Ho approfondito i risultati ottenuti sul dataset di validation (esattamente come poi effettuerò per il Test), attraverso i seguenti grafi e calcoli:

- Confusion Matrix: per avere una panoramica delle label predette rispetto a quelle reali; ottenendo i valori di vero e falso positivo e di vero e falso negativo.
- Accuracy, precision, recall e f1-score per label:
    - Accuracy: numero di previsioni corrette / previsioni totali;
    - Precision: True Positives / (True Positives + False Positives), misura le classificazioni positive corrette rispetto al totale delle classificazioni positive ottenute;
    - Recall: True Positives / (True Positives + False Negatives), misura le classificazioni positive corrette rispetto a tutte le istanze positive reali.
    - F1-score: tiene in considerazione le due metriche precedenti, per fornire una misura complessiva;
- Istogramma: rappresentate le probabilità ottenute rispetto alle classi reali.

- Curva ROC: che esamina la relazione tra il tasso di veri positivi (True Positive Rate, TPR) e il tasso di falsi positivi (False Positive Rate, FPR) al variare della soglia di classificazione;



Accuracy: 0.674

Classe	precision	recall	f1-score
Non sarcastico	0.659	0.721	0.689
Sarcastico	0.691	0.627	0.658

Figura 3.3: Confusion Matrix e statistiche per il dataset di validation

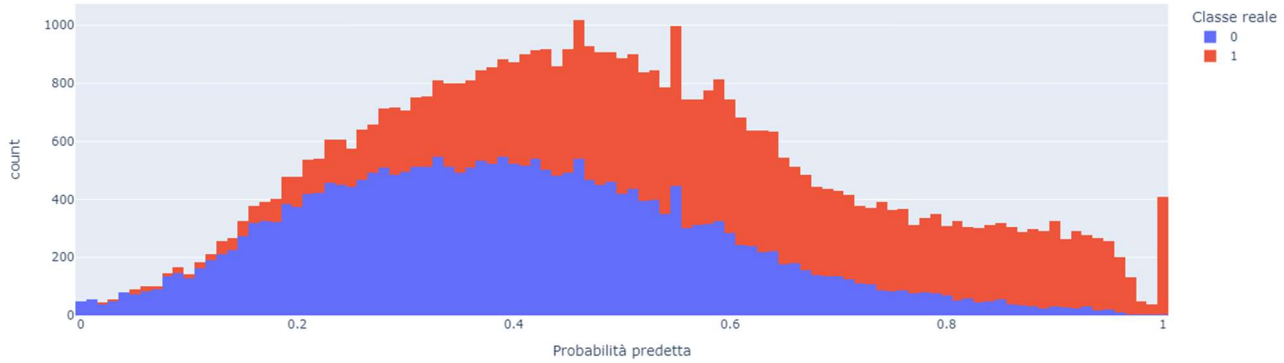


Figura 3.4: Istogramma che confronta le probabilità predette rispetto alle classi reali; per il dataset di validation

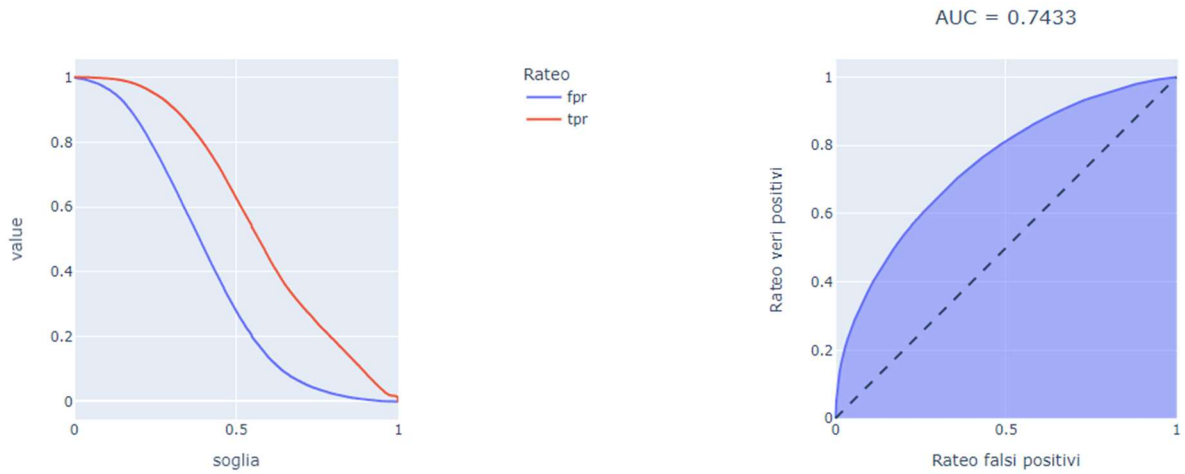


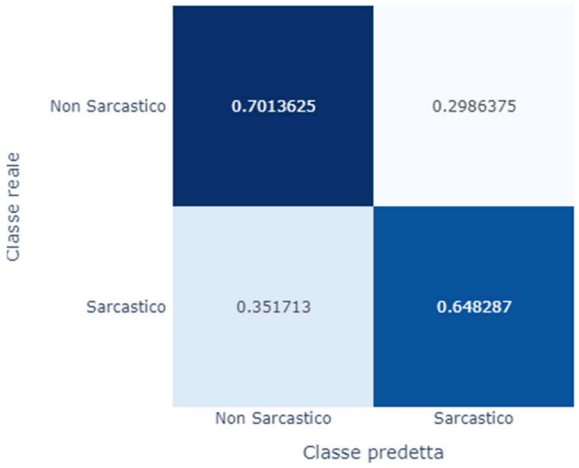
Figura 3.5: ROC, AUC e ratei FPR e TPR a diverse soglie; per il dataset di validation

Analizzando i risultati ottenuti; visualizzabili in Figura 3.3, Figura 3.4 e Figura 3.5, possiamo notare che:

- Il modello prevede decisamente meglio le istanze negative; infatti, il rateo di veri negativi è del 72% contro il 62.7% dei veri positivi; ciò lo conferma anche l'F1-score e la recall per la label negativa.
- Il modello ha un alto tasso di falsi positivi, rispetto a quelli di falsi negativi (37.3% contro 27%); cioè, tende maggiormente a classificare positive istanze negative rispetto al contrario.
- L' AUC di 0.743 indica che il modello ha un buon poter discriminante fra le classi, ma non eccessivo, infatti, si discosta abbastanza dallo 0.5 ma non si avvicina molto al valore di 1.

3.3 Risultati sul dataset di Test

Procedendo ora con l'analisi sul dataset di Test, ho ottenuto:



Accuracy: 0.675

Classe	precision	recall	f1-score
Non sarcastico	0.666	0.701	0.683
Sarcastico	0.685	0.648	0.666

Figura 3.6: Confusion Matrix e statistiche per il dataset di test

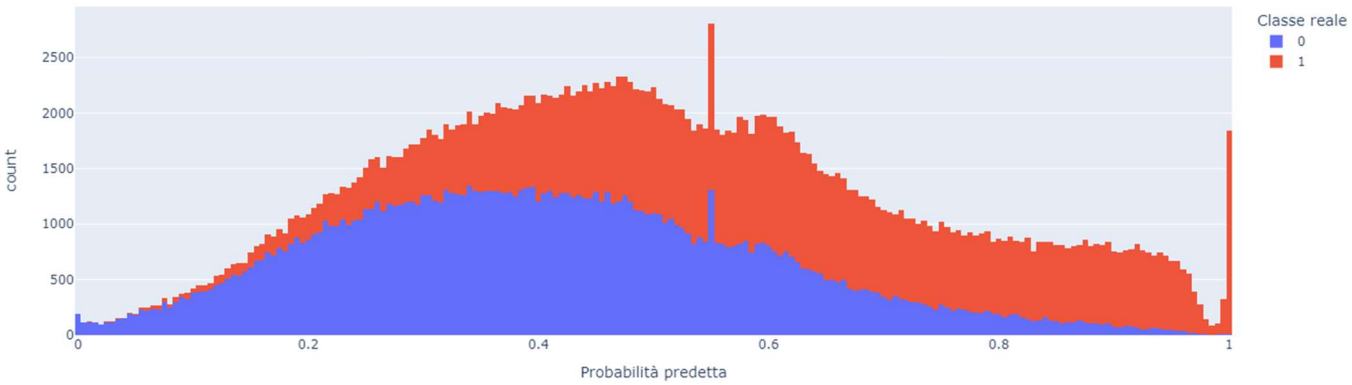


Figura 3.7: Istogramma che confronta le probabilità predette rispetto alle classi reali; per il dataset di test

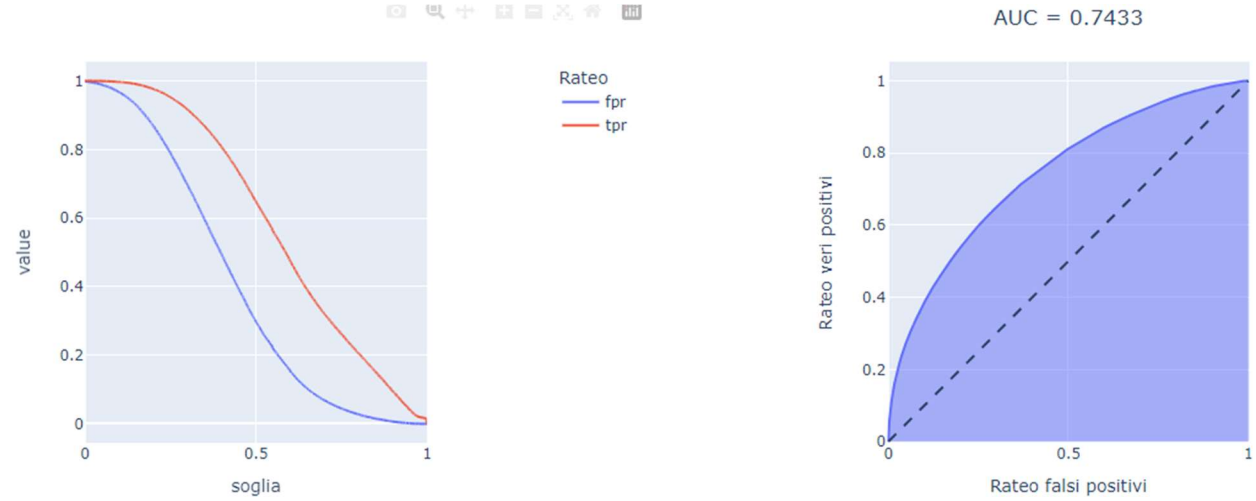


Figura 3.8: ROC, AUC e ratei FPR e TPR a diverse soglie; per il dataset di test

Osservando Figura 3.6, Figura 3.7 e Figura 3.8, notiamo che le considerazioni fatte per i dati di validation non sono cambiate molto, nello specifico:

- Il modello conferma la sua capacità di generalizzazione, in quanto l'AUC non è cambiato (considerando l'approssimazione alle 3 cifre decimali),
- Il rateo di veri negativi è sceso, ma è aumentato quello di veri positivi; quindi, il modello tende a distinguere meglio le istanze positive.
- L'Accuracy è rimasta quasi invariata (da 67.4% a 67.5%).
- Precision, Recall e F1-score sono meno polarizzate fra le due classi, rispetto al dataset precedente.

Conclusione

In questo progetto ho approcciato la rilevazione del sarcasmo, una branca di NLP particolarmente ostica, utilizzando approcci nuovi e più sperimentali (come i Transformer), seppur in una scala ridotta.

Il dataset messo a disposizione, con le sue 5 feature e oltre 1M di istanze, dopo una breve fase di preprocessing, mi ha permesso di effettuare diverse analisi. In cui ho calcolato il rateo informativo per i diversi elementi del contesto, e per i diversi tipi di testo.

In fine ho processato i dataset, in accordo con le analisi effettuate su quello di Training, per poi addestrare un modello basato su BERT e sui TransformerEncoder.

In conclusione le analisi effettuate, nel primo capitolo, sono state uno strumento utile per migliorare la precisione del modello; riducendo anche le risorse e i tempi computazionali, in quanto, mi hanno permesso di escludere dati superflui. Il modello mostra ottime capacità di generalizzazione, ma anche un certo grado di underfitting.

Nel futuro, con migliori risorse computazionali, si potrebbe effettuare tuning degli iperparametri, in modo da individuare il corretto numero di neuroni nei layer interni e il numero di head del TransformerEncoder; oltre, a poter sperimentare con ulteriori input o con BERT a dimensioni maggiori.

Appendice: alcuni dettagli tecnici.

Por poter eseguire da zero il codice è consigliato eseguire il file main.py, messo nel folder del progetto, che esegue prima il file di elaborazione dei dati, poi quello di elaborazione del modello, e infine delle elaborazione ulteriori, necessarie per l'applicazione Dash.

Per poter addestrare da zero il modello, è necessario flaggare a False la costante MODEL_LOAD posta nel file costant.py.

Per avviare l'applicazione Dash bisogna eseguire il file index.py nel folder "app"; tuttavia, nonostante le operazioni di ottimizzazione effettuate, a causa del numero elevato di istanze, essa risulta abbastanza pesante.

L'applicazione Dash è strutturata su 6 pagine:

- Dashboard: è la pagina iniziale e contiene le informazioni sui dati iniziali;
- Sentence Len Analysis: contiene le analisi sulle lunghezze del parent e del testo;
- Context Exploring: contiene le analisi degli elementi del contesto.
- Text Analysis: contiene l'analisi effettuata sui tipi di testo.
- Model Training: contiene le analisi effettuate sul training del modello, e sul dataset di validation.
- Model Testing: al cui interno sono presenti le analisi sul dataset di test e la demo del modello.