Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Corso di Laurea Specialistica in Informatica

Previsione di sentimenti di odio e stereotipi tramite artificial neural network

Studente:

Alessio Mancinelli

Professore:

Prof. Alfredo Milani

Anno Accademico 2019–2020

Introduzione

Il progetto si pone l'obiettivo di proporre dei modelli di artificial neural network che siano in grado di prevedere se il testo inserito da tastiera contiene sentimenti di odio e/o stereotipi. Tutte le ANN saranno addestrate con il dataset ottenuto dal sito Evalita 2020, contenente circa 7000 tweets già etichettati; si tratterà quindi in un problema di apprendimento supervisionato. Con riferimento ai modelli, il progetto si articola in 4 capitoli: il primo capitolo descrive le varie librerie e framework utilizzati per lo svolgimento del progetto, facendo riferimento anche a come è stato possibile garantire la riproducibilità dei risultati. Il secondo capitolo invece delinea l'architettura della soluzione proposta, viene descritto quindi l'intero processo di preprocessing, cleaning e filtering del dataset utilizzato, incluso lo studio effettuato per la scelta delle feature. Il terzo capitolo si concentra nel descrivere i vari esperimenti effettuati con i modelli, facendo riferimento anche alle metriche con le quali sono stati calcolati i risultati. Nel quarto, infine, si espongono i risultati dell'indagine svolta.

Capitolo 1

Framework e librerie utilizzate

Il linguaggio di programmazione utilizzato per lo svolgimento di questo progetto è stato Python. Esso è un linguaggio orientato agli oggetti, ad alto livello e con semantica dinamica, che ha permesso la costruzione dei vari modelli proposti attraverso librerie e framework specifici che descriveremo in questo capitolo.

1.1 Numpy

Numpy è il pacchetto fondamentale per l'elaborazione scientifica in Python che fornisce funzioni matematiche e strutture dati per il calcolo e l'implementazione di vettori e matrici multidimensionali. Molte librerie, come Pandas per esempio, si basano su di esso per l'esecuzione di calcoli.

1.2 Pandas

Pandas è una libreria open source per Python che fornisce strutture dati flessibili, progettate per semplificare la manipolazione di diverse tipologie di formati. Tra le varie strutture che la libreria permette di manipolare troviamo dati in formato tabellare (fogli di calcolo Excel o file aventi formato CSV), serie di dati temporali ordinati e non, e strutture dati aventi indici di riga e di colonna. Quest'ultimo aspetto è anche omissibile, in quanto i dati non devono essere necessariamente indicizzati per essere inseriti in una struttura di Pandas. Le due tipologie di strutture dati che la libreria offre sono chiamate 'Series' e 'Dataframe'. La differenza tra queste è che le 'Series' sono strutture dati unidimensionali, mentre i 'Dataframe' bidimensionali. Tra le caratteristiche fondamentali di Pandas troviamo la possibilità di gestire dati mancanti (rappresentati con 'NaN') e la mutabilità dimensionale, ovvero, righe e colonne possono essere inserite ed eliminate da una struttura in qualunque momento. Troviamo poi l'allineamento automatico dei dati che consiste nel dare la possibilità al programmatore di associare delle etichette a ciascuno di essi, o lasciare che la libreria associ automaticamente i dati durante la computazione, infine troviamo la possibilità di effettuare merge e join tra dataset.

1.3 Scikit-learn

Scikit-learn è una libreria in Python che offre la possibilità di utilizzare svariati algoritmi di apprendimento sia supervisionato che non supervisionato. Essa interagisce con varie librerie, alcune di queste già precedentemente de-

scritte, come Pandas e NumPy. Per il suo utilizzo all'interno del linguaggio di programmazione, bisogna far riferimento alla sintassi 'Sklearn', che permette la costruzione di modelli basati su algoritmi di regressione (sia logistica che lineare), classificazione e clustering. Permette anche di richiamare metodi per la suddivisione dei dati, come la cross validation e per il calcolo delle performance dei modelli, dando la possibilità di costruire la matrice di confusione. Questa libreria è focalizzata sulla modellazione dei dati, per quanto concerne il caricamento e la manipolazione, si fa riferimento alle libreria Pandas e Numpy.

1.4 Keras

Keras è un API di alto livello, scritta in Python, che permette una rapida e semplice prototipazione di ANN, dando la possibilità al programmatore di scegliere numero di livelli, numero di neuroni e funzioni d'attivazione da utilizzare. Essa supporta sia reti convoluzionali che reti ricorrenti, nonchè una combinazione tra le due; il suo funzionamento può avvenire attraverso CPU o GPU. Keras può avere diversi backend; questo perchè la libreria non gestisce operazioni di basso livello, come per esempio il prodotto tra tensori ma, al contrario, si affida ad una libreria di manipolazione specializzata e ben ottimizzata per farlo. Keras gestisce il problema in maniera modulare, ovvero, invece di scegliere una singola libreria e rendere l'implementazione strettamente legata a quest'ultima, permette il collegamento di diversi backend. I backend disponibili sono tre: TensorFlow, Theano e CNTK. Tensorflow e Theano sono framework che permettono la manipolazione di tensori, sviluppati rispettivamente da Google e LISA, mentre CNTK è un

toolkit open source per il deep learning sviluppato da Microsoft. Il backend scelto in questo progetto è stato Tensorflow.

1.5 Riproducibilità dei risultati

I framework e le librerie utilizzate ai fini di garantire la riproducibilità dei risultati sono state 'numpy', 'random', 'tensorflow' e 'os'. I primi tre sono stati utilizzati per impostare il random seed, così da garantire, per esempio, che il valore dei pesi della ANN proposta siano sempre gli stessi ad ogni esecuzione del programma. Inoltre il framework TensorFlow è stato utilizzato per fare in modo che il backend utilizzi un singolo thread. La libreria 'os' invece ha permesso di accedere alle funzionalità del sistema operativo ed è stata utilizzata per impostare il PYTHONHASHSEED a zero, per garantire la ripetibilità degli esperimenti in relazione agli oggetti di str, byte e datetime.

```
random.seed(1234)
np.random.seed(1234)
tf.set_random_seed(1234)
os.environ['PYTHONHASHSEED'] = '0'
session_conf = tf.ConfigProto(
      intra_op_parallelism_threads=1,
      inter_op_parallelism_threads=1)

sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

Figura 1.1: Codice per la riproducibilità dei risultati

Capitolo 2

Architettura della soluzione

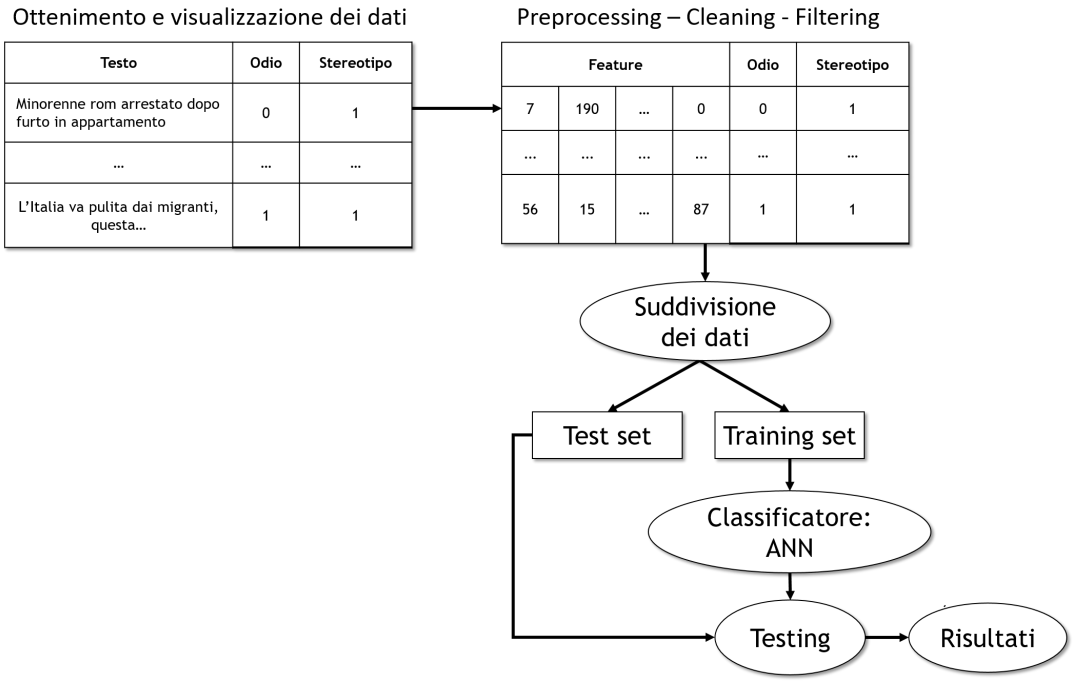

Figura 2.1: Workflow della soluzione

La soluzione adottata si svolge in diverse fasi in accordo al workflow mostrato in figura 2.1, dove i rettangoli rappresentano i dati utilizzati e gli

ovali le azioni svolte; segue una breve descrizione per ogni fase.

- Ottenimento e visualizzazione del dataset dal sito di Evalita 2020.
- Preprocessing - Cleaning - Filtering: Trasformazione dei dati e ottenimento delle Feature.
- Suddivisione dei dati al fine di ottenere il training e test set.
- Classificatore: Prototipazione delle ANN.
- Fase di testing: Applicazione dei modelli costruiti.
- Risultati: Ottenimento e visualizzazione dei risultati.

2.1 Dataset e preprocessing dei dati

Il dataset ottenuto dal sito di Evalita 2020 (visibile in figura 2.2) è stato distribuito in formato TSV e contiene 6849 tweets. Di questi:

- 4072 non contengono sentimenti di odio;
- 2777 contengono sentimenti di odio;
- 3803 non contengono stereotipi
- 3046 contengono stereotipi.

Le colonne disponibili sono quattro, ma ai fini del progetto verranno prese in considerazione solamente la colonna 'text', 'hs' e 'stereotype', non essendo l'id dei tweet un informazione rappresentativa del problema che sta venendo trattato.

	A	B	C	D
1	id	text	hs	stereotype
2	2066	Ã^ terrorismo anche questo, per mettere in uno stato di soggezione le persone e renderle innocue, mentre qualcuno... URL	0	0
3	2045	@user @user infatti finchÃ© ci hanno guadagnato con i campi #rom tutto era ok con #Alemanno #Ipocriti	0	0
4	61	Corriere: Tangenti, Mafia Capitale dimenticataMazzette su buche e campi rom URL #roma	0	0
5	1259	@user ad uno ad uno, perchÃ© quando i migranti israeliti arrivarono in terra di Canaan fecero fuori tutti i Canaaniti.	0	0
6	949	Il divertimento del giorno? Trovare i patrioti italiani che inneggiano contro i rom facendo la spesa alla #Lidl (multinazionale tede	0	0
7	256	Modena: Comune paga la benzina ai nomadi che portano figli a scuola: MODENA â€“ La giunta PDâ€¦ URL	0	0
8	3001	@user @user altro che islam o cristianesimo!!! ...a c dobbiamo sorbire anche "dell'ignorante" : islam Ã¨ uno solo!!	0	1
9	765	@user @user @user grazie stef stavo giusto caricando ho anche messo Che Salvini avallava il finanziare campi rom	0	0
10	1157	... e smettetela di dire che anche gli italiani sono stati migranti, erano trattati male ma non per questo uccidevano innocenti per	0	1
11	268	Minorenne rom arrestato dopo furto in appartamento URL	0	1
12	644	Studentessa cinese morta fermato un nomade di 20anni e denunciato nomade sedicenne tutti del vicino campo nomade SkyTG	0	1
13	2518	Ddl di Calderoli: â€œIl velo islamico e lâ€™apologia della Sharia diventino reatoâ€. SieteÂ dâ€™accordo? URL	0	0

Figura 2.2: Dataset in formato TSV

```
df_cols = ["testo", "odio", "stereotipi"]
df = pd.read_csv('haspeede2_dev_taskAB.tsv', delimiter = "\t", encoding='utf-8', names = df_cols)
df = df.drop(df.index[0])
df["odio"].replace({"0": 0, "1": 1}, inplace=True)
df["stereotipi"].replace({"0": 0, "1": 1}, inplace=True)
```

Figura 2.3: Apertura del dataset in Python

Il dataset è stato aperto all'interno del linguaggio di programmazione tramite il metodo *read_csv* della libreria Pandas, che permetterà la gestione tramite Dataframe. Sono stati poi rimpiazzati tutti i valori delle colonne 'odio' e 'stereotipi' (che rappresentano le label da prevedere) con valori numerici. Le prime cinque righe del dataframe appena creato sono le seguenti:

	testo	odio	stereotipi
2066	È terrorismo anche questo, per mettere in uno ...	0	0
2045	@user @user infatti finché ci hanno guadagnato...	0	0
61	Corriere: Tangenti, Mafia Capitale dimenticata...	0	0
1259	@user ad uno ad uno, perché quando i migranti ...	0	0
949	Il divertimento del giorno? Trovare i patrioti...	0	0

2.2 Pulizia del testo

Prima di poter trasformare il testo in un formato consono all'addestramento del modello, è necessario rimuovere tutte le parole prive di effettivo significato. Ho deciso quindi di eliminare sia tutte le stopwords in lingua italiana, sia

alcune specifiche parole all'interno del testo come "URL" o "@user". Particolare attenzione è stata posta alle emoji, che possono portare informazione utile che il modello potrebbe imparare a riconoscere. Per la loro trasformazione in testo ho utilizzato la libreria 'emoji' che mappa l'emoji analizzata con la sua rispettiva rappresentazione testuale. Per la rimozione delle stopwords ho utilizzato la libreria *nltk* che mette a disposizione dei file predefiniti contenenti articoli e parole prive di significato in lingua italiana. Un altro processo molto utile per la pulizia del testo è lo Stemming (Fig 2.4), ovvero prendere solamente la radice delle parole così da fare in modo che tutte le possibili declinazioni di una parola prendano lo stesso significato.

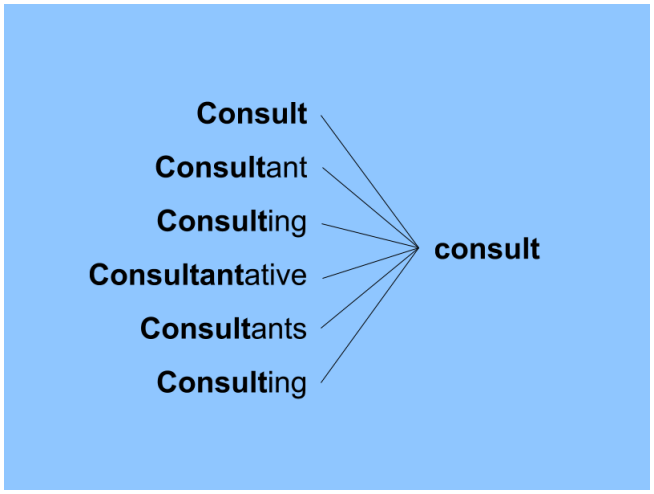

Figura 2.4: Esempio di Stemming

In questo specifico caso però, ho scelto di non utilizzare questa tecnica perchè non ho riscontrato un aumento significativo delle performance dopo l'applicazione. Vengono quindi mostrate in figura 2.5 alcune applicazioni della funzione "clean_text()" implementata in python per la pulizia del testo.

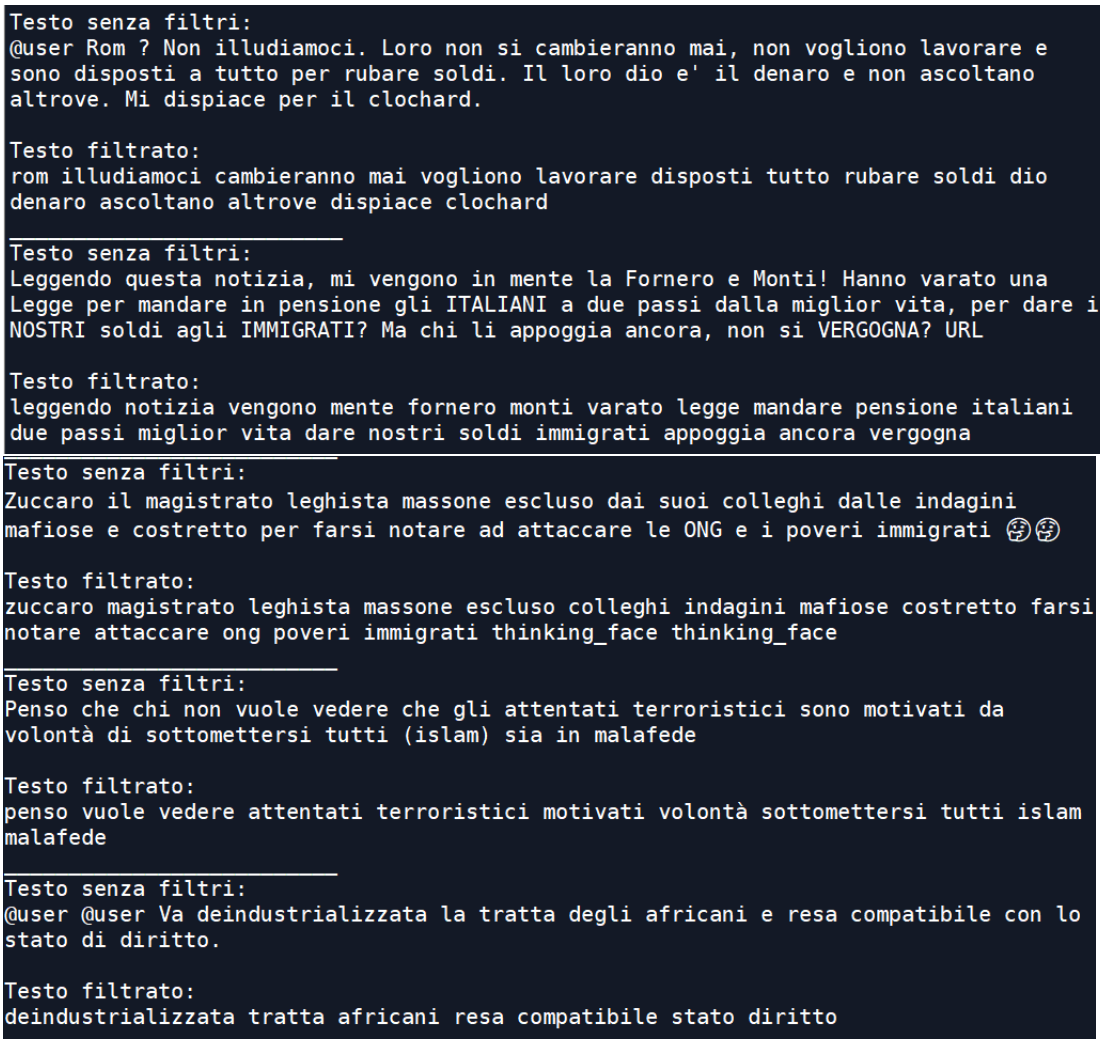

Figura 2.5: Esempi di pulizia del testo

2.3 Ulteriori studi sul dataset

Vediamo adesso alcune frasi appartenenti alle differenti classi presenti nel dataset. Un esempio di frase che contiene sia sentimenti di odio che stereotipi è la seguente:

#londonattack chiedete ai buonisti del cavolo cosa vogliono fare con i islamici perché io se chiedo mi vengono solo insulti sulla bocca

Un esempio di frase che non contiene sentimenti di odio ne stereotipi è la seguente:

@user @user infatti finchè ci hanno guadagnato con i campi #rom tutto era ok con #Alemanno #Ipocriti

Un esempio di frase che non contiene sentimenti di odio, ma invece contiene stereotipi è la seguente:

Certo che in #Tunisia non dev'essere facile indagare su estremismo islamico. Sono tutti estremisti.

In questo caso lo stereotipo sta nel fatto di supporre che tutti i Tunisini siano estremisti. Un esempio di frase che contiene sentimenti di odio e non stereotipi è la seguente:

#canaledisicilia mi hanno rotto i coglioni tutti quanti: politici, scafisti e immigrati. Chiudiamo le frontiere e stop.

Viene adesso mostrato uno studio relativo all'analisi delle 20 parole più utilizzate da chi scrive testi contenenti sentimenti di odio e/o stereotipi, incluse le emoji.

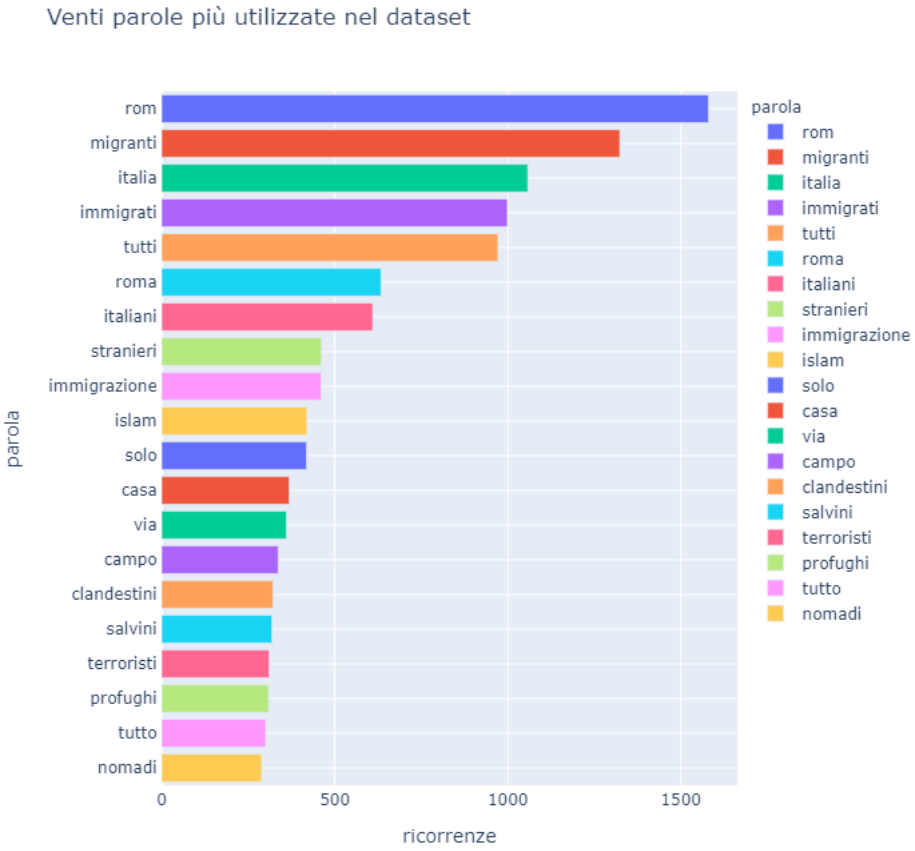

Figura 2.6: Venti parole più utilizzate in generale nel dataset

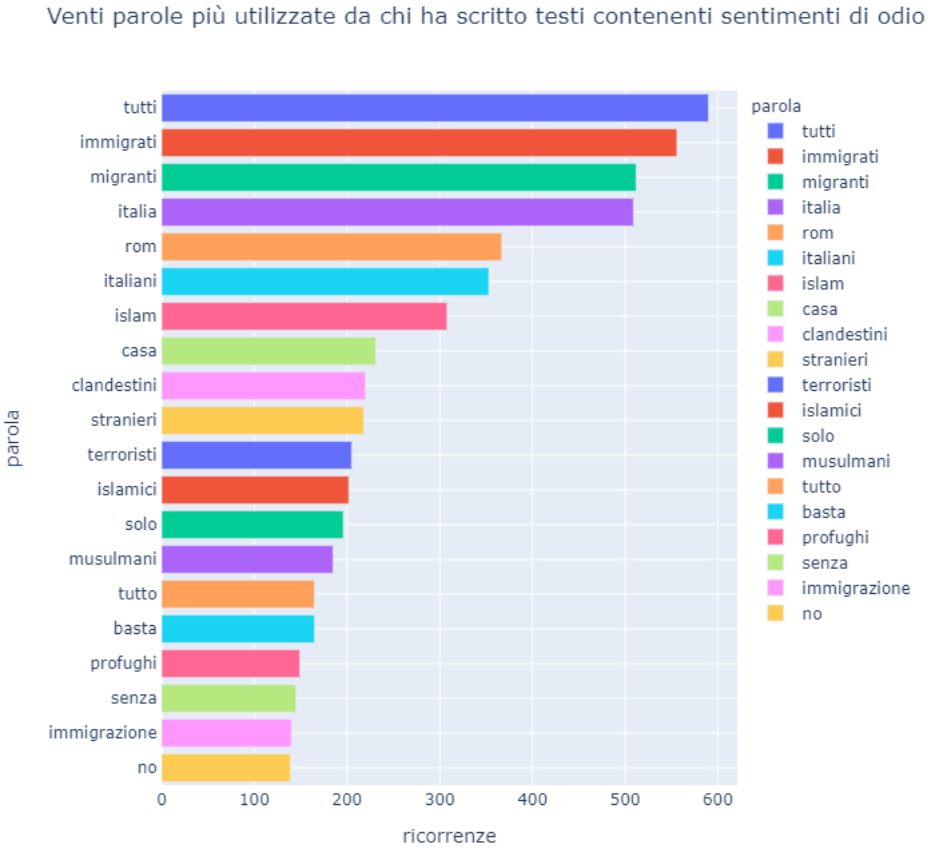

Figura 2.7: Venti parole più utilizzate da chi *ha* scritto testi contenenti sentimenti di odio

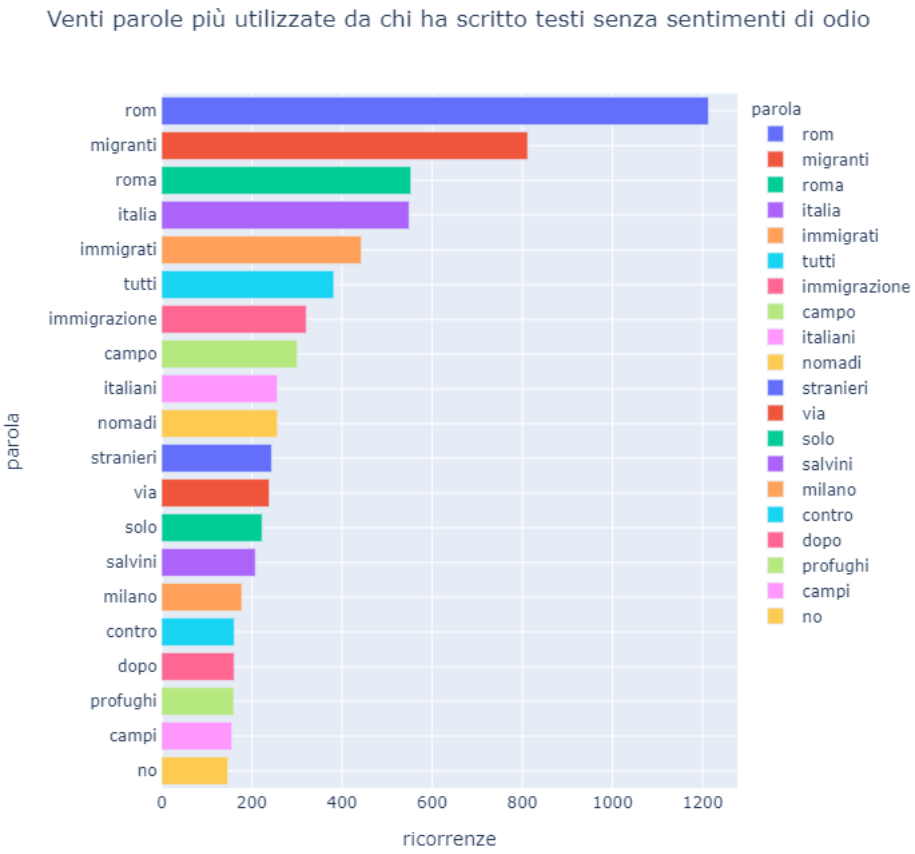

Figura 2.8: Venti parole più utilizzate da chi ha scritto testi *non* contenenti sentimenti di odio

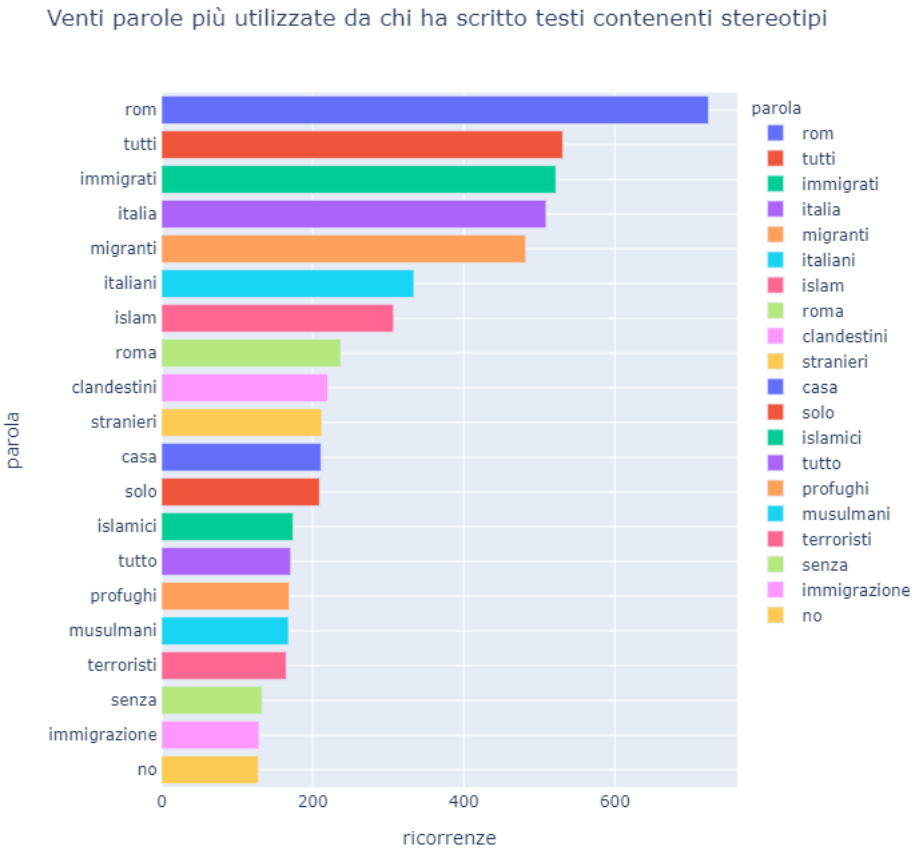

Figura 2.9: Venti parole più utilizzate da chi *ha* scritto testi contenenti stereotipi

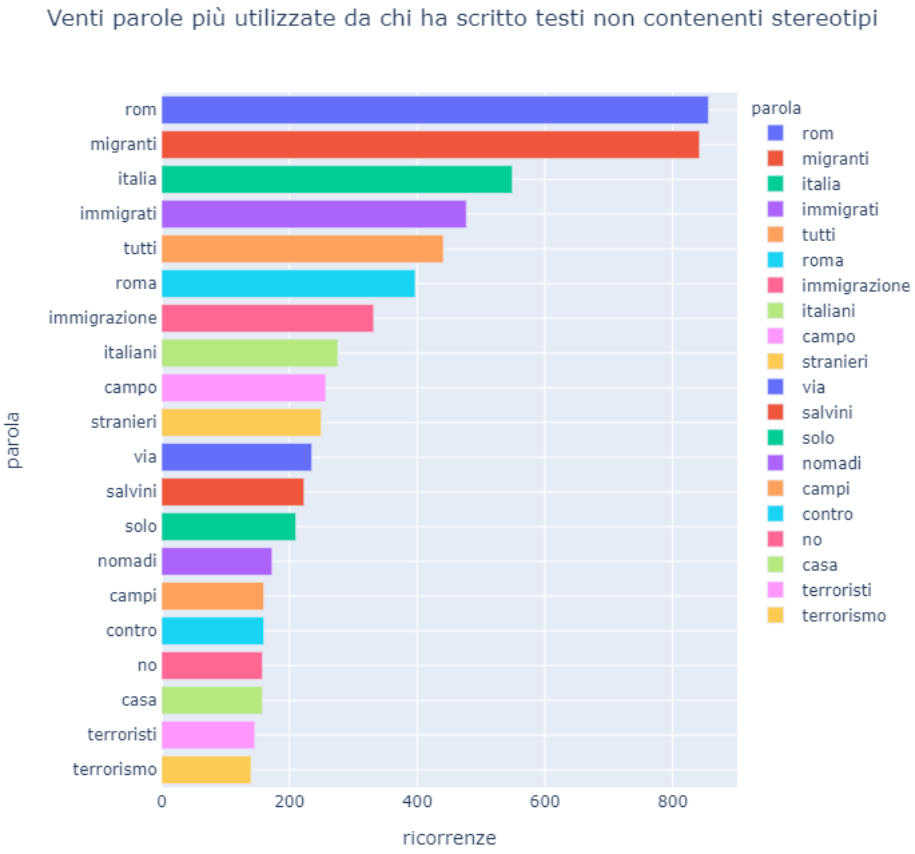

Figura 2.10: Venti parole più utilizzate da chi ha scritto testi *non* contenenti stereotipi

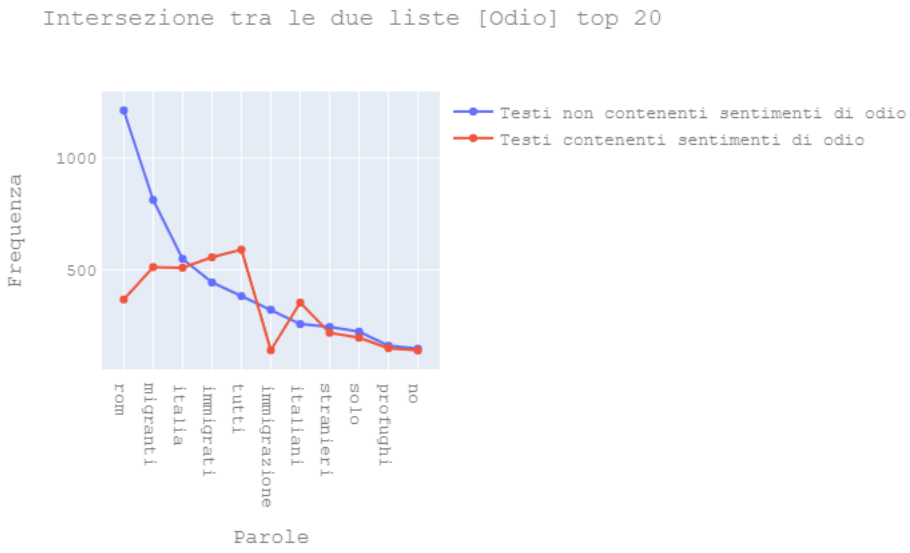

Figura 2.11: Intersezione tra le 20 parole più usate relative ai sentimenti di odio

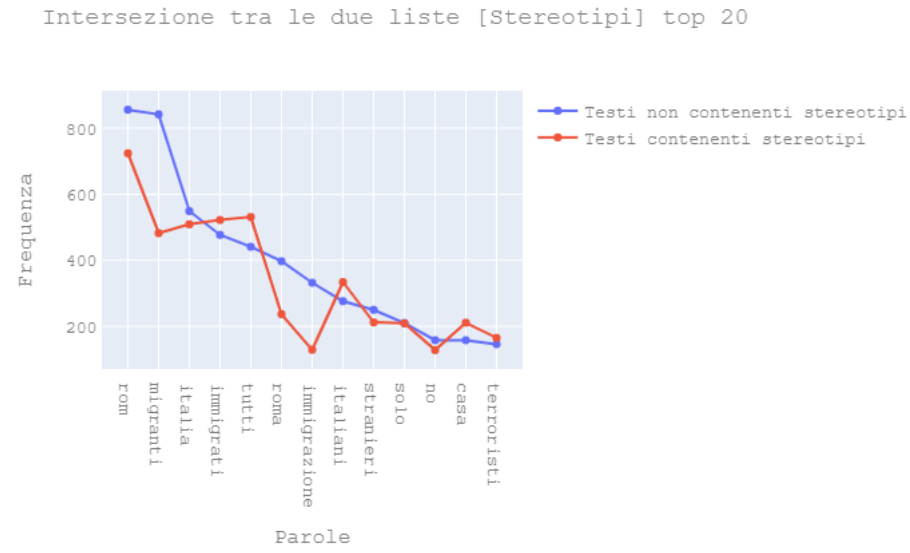

Figura 2.12: Intersezione tra le 20 parole più usate relative agli stereotipi

La parola più utilizzata in tutto il dataset è 'rom', con un numero di ricorrenze pari a 1589. Chi scrive testi contenenti sentimenti di odio utilizza più frequentemente parole come 'tutti', 'immigrati' e 'migranti', rispetto a chi non scrive testi contenenti sentimenti di odio che utilizza più frequentemente parole come 'rom', 'roma' e 'Italia'. Per quanto riguarda gli stereotipi, le parole più usate da chi scrive testi che li contengono sono 'tutti', 'rom' e 'immigrati', mentre per i restanti abbiamo sempre le parole 'rom' e 'migranti'. Come possiamo notare, la parola rom è utilizzata molto spesso da entrambe le classi. In merito all'intersezione tra le venti parole più frequenti utilizzate da chi scrive testi contenenti sentimenti di odio e non notiamo che il numero di elementi presenti è undici, quindi più della metà delle parole vengono utilizzate da entrambe le classi ma con frequenza diversa. La stessa osservazione vale per l'intersezione tra le parole più frequenti di chi scrive testi contenenti stereotipi e non; in questo caso, le parole in comune sono tredici. Andiamo adesso ad analizzare il numero di emoji utilizzate all'interno dei testi. Su circa 7000 tweets, solamente 298 contengono emoji, quindi il numero di occorrenze risulterà molto più basso rispetto a quello delle parole. Come vedremo, l'emoji 'angry' sarà la più utilizzata sia da chi scrive testi contenenti sentimenti di odio sia stereotipi, mentre l'emoji 'face_with_tears_of_joy' viene utilizzata molto frequentemente da tutte le classi.

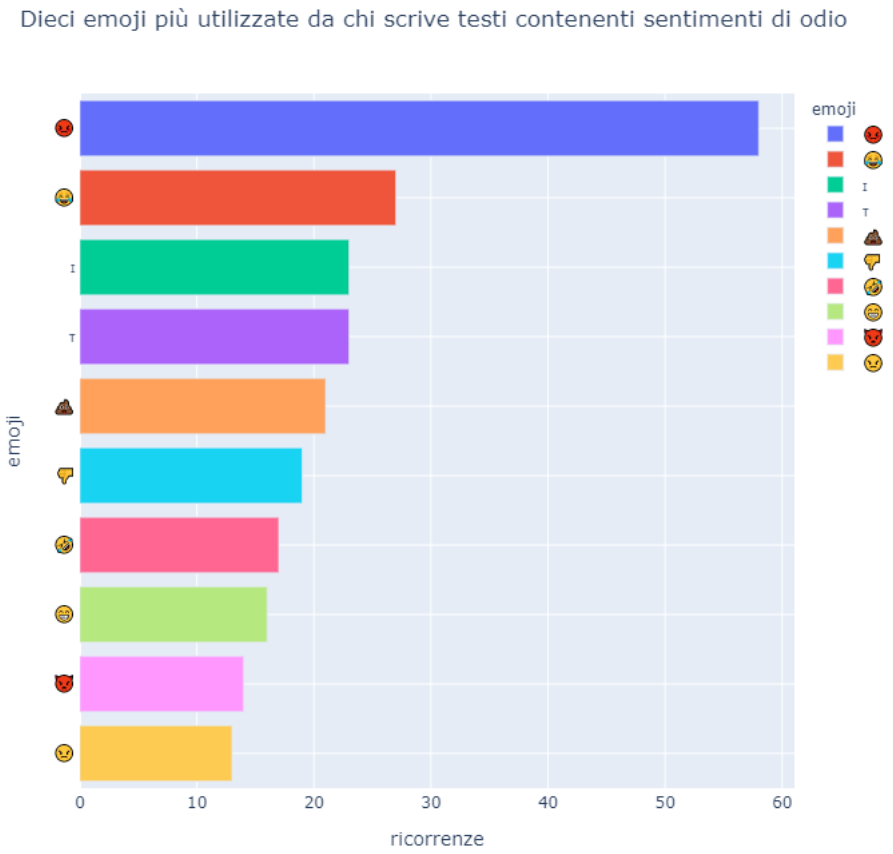

Figura 2.13: Le dieci emoji più utilizzate da chi scrive testi contenenti sentimenti di odio

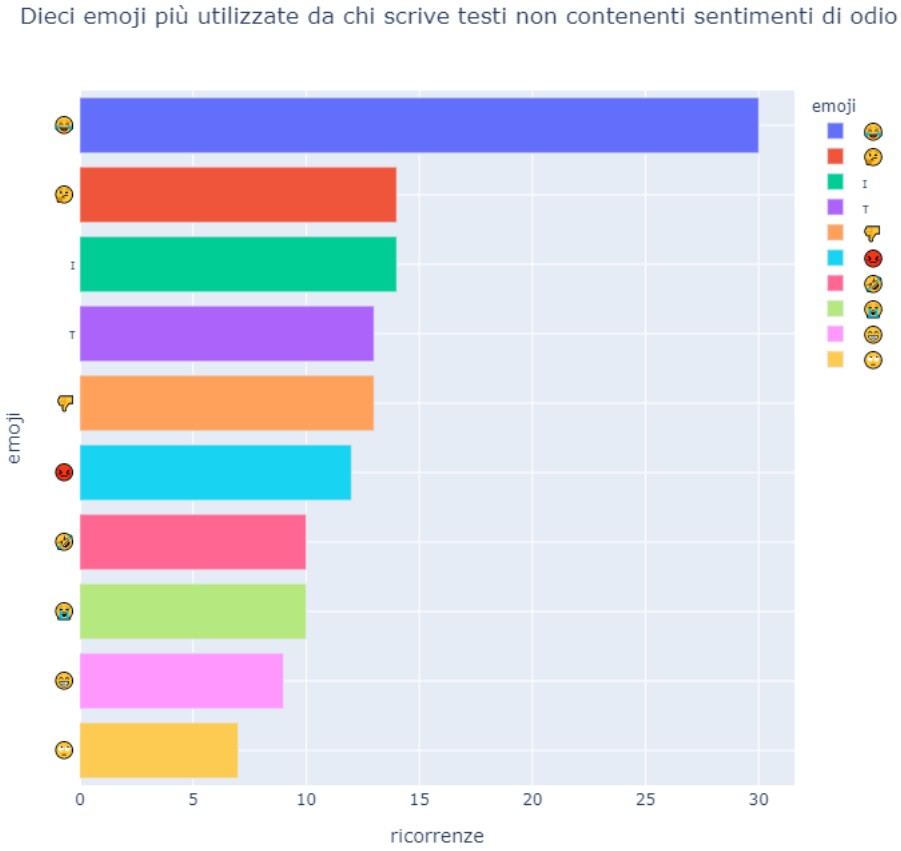

Figura 2.14: Le dieci emoji più utilizzate da chi scrive testi *non* contenenti sentimenti di odio

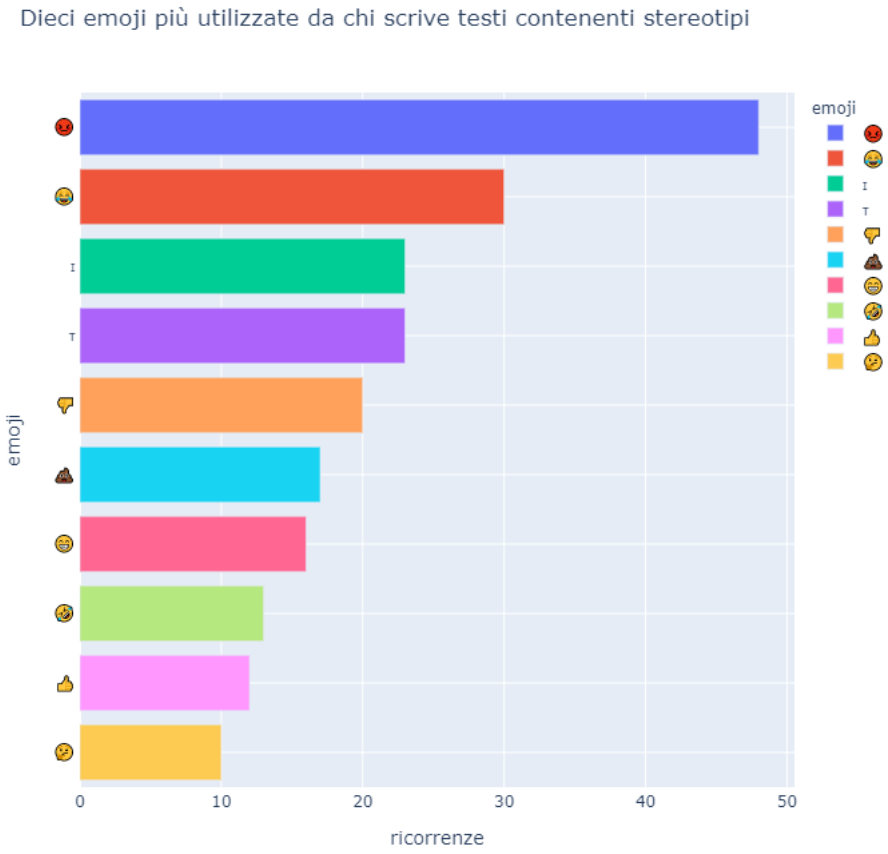

Figura 2.15: Le dieci emoji più utilizzate da chi scrive testi contenenti stereotipi

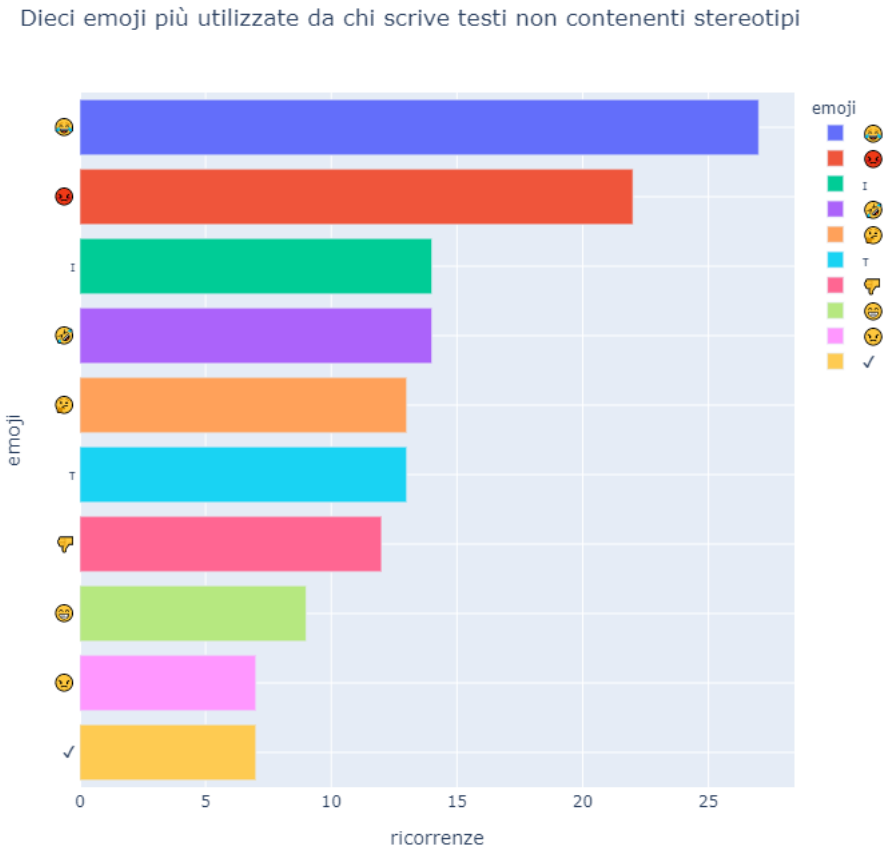

Figura 2.16: Le dieci emoji più utilizzate da chi scrive testi *non* contenenti stereotipi

2.4 Feature Engineering

Per le creazione delle feature ho svolto un primo esperimento sulla lunghezza delle frasi appartenenti all'una o all'altra classe, per verificare se questa misura possa contenere informazioni implicite che il modello potrebbe imparare a riconoscere. I risultati ottenuti sono che chi scrive testi non contenenti sentimenti di odio utilizza in media 136.46 caratteri per frase, mentre chi scrive testi contenenti sentimenti di odio ne utilizza 165.72. Per quanto riguarda gli stereotipi abbiamo 138.03 caratteri per frase per la classe 0 (assenza di stereotipi) e 161.18 caratteri per chi scrive testi contenenti stereotipi. Come possiamo notare dai valori, chi scrive testi contenenti sentimenti di odio o stereotipi è più propenso a scrivere molti caratteri all'interno delle frasi. Controlliamo adesso il numero di parole all'interno del testo. Per chi scrive testi contenenti sentimenti di odio abbiamo in media 21.61 parole per frase, mentre 25.95 parole per la classe opposta. Per quanto riguarda gli stereotipi abbiamo 21.82 parole per chi scrive testi non contenenti stereotipi e 25.31 per i restanti. Anche in questo caso, viene dimostrato che chi scrive testi contenenti sentimenti di odio o stereotipi è incline ad inserire più parole all'interno della frase. Sulla base di queste misure, le feature prese in considerazione per ogni tweet sono state:

1. Numero di parole
2. Numero di lettere
3. Lunghezza media delle parole ottenuta tramite la divisione tra il numero di lettere ed il numero di parole nella frase.
4. Numero di stopwords in lingua italiana.

5. Numero di parole incluse nelle 20 più usate di tutto il dataset
6. Numero di parole incluse nelle 10 più usate da chi ha scritto testi contenenti sentimenti di odio
7. Numero di parole incluse nelle 10 più usate da chi ha scritto testi contenenti stereotipi
8. Numero di parole incluse nelle 10 più usate da chi *non* ha scritto testi contenenti sentimenti di odio
9. Numero di parole incluse nelle 10 più usate da chi *non* ha scritto testi contenenti stereotipi
10. Numero di punctuation
11. Numero di punti esclamativi (!)
12. Numero di punti interrogativi (?)
13. Numero di parole completamente maiuscole
14. Numero di parole che iniziano con la lettera maiuscola
15. Numero di riferimenti ad altri utenti (@user)
16. Numero di hashtag inseriti (#)
17. Numero di emoji totali all'interno della frase
18. Numero totale di emoji incluse nelle 5 più usate da chi ha scritto testi contenenti sentimenti di odio

19. Numero totale di emoji incluse nelle 5 più usate da chi *non* ha scritto testi contenenti sentimenti di odio

20. Numero totale di emoji incluse nelle 5 più usate da chi ha scritto testi contenenti stereotipi

21. Numero totale di emoji incluse nelle 5 più usate da chi *non* ha scritto testi non contenenti stereotipi

Le feature calcolate saranno fornite al modello in un livello di input apposito, così da essere concatenate insieme al testo filtrato in formato numerico.

2.5 Embeddings

Con il termine Word Embeddings viene indicato un insieme di strumenti, modelli del linguaggio e tecniche di apprendimento all'interno dell'area del Natural Language Processing che permettono di rappresentare parole e frasi di testi scritti attraverso vettori di numeri reali. L'approccio più semplice per la rappresentazione delle parole è quello di utilizzare i cosiddetti 'one-hot embeddings', attraverso cioè vettori di grandi dimensioni (tante quante la dimensione del dizionario) con componenti tutte uguali a zero, tranne per quella relativa all'indice della parola rappresentata. Il problema di questa rappresentazione è che non fornisce informazioni sulle relazioni tra le diverse parole, e quindi non permette l'estrazione di un valore di similarità. Il Word Embeddings invece, è in grado di codificare le informazioni semantiche e sintattiche delle parole, dove l'informazione semantica ha a che fare con il significato, mentre quella sintattica ha a che fare con il ruolo all'interno della struttura del testo. Tramite questa rappresentazione è possibile sviluppare

modelli linguistici e statistici che permettono, a partire da un corpus di documenti iniziale, di addestrare tramite reti neurali i vettori stessi. La maggior parte di questi modelli cercano di ottimizzare una funzione di loss andando a minimizzare la differenza tra i valori della predizione e quelli reali.

2.5.1 Creazione del vocabolario

Per la creazione del vocabolario ho utilizzato la classe Tokenizer della libreria Keras, che ha consentito di vettorializzare un corpus di testo trasformando ciascuna frase in una sequenza di numeri interi (ogni numero intero è l'indice di un token in un dizionario). La grandezza del vocabolario in questo caso è di 17777 parole. Ai fini di poter dare come input i dati ottenuti ad una rete neurale, è necessario che la lunghezza di tutte le frasi sia la stessa. Ho quindi iterato tutte le frasi all'interno del dataset (gestito in questo caso tramite una lista contenente tutto il testo scritto dagli utenti) tenendo traccia della lunghezza massima trovata. Questo valore è stato utilizzato per l'aggiunta di Zero padding in post a tutte le frasi.

```
from sklearn.utils import shuffle
df = shuffle(df)
df = df.drop(columns=['text_backup'])
df = df.drop(columns=['stereotipi'])
# Metti tutto quello che trovi nella colonna 'testo' nella lista chiamata lista_testo
lista_testo = df["testo"].fillna('').to_list()
# prendo tutti i valori della mia lista e li casto a stringa
lista_testo = [str(i) for i in lista_testo]
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lista_testo)
vocab_size = len(tokenizer.word_index) + 1
print("Le parole all'interno del vocabolario sono: ", vocab_size)
lista_testo_tokenizer = tokenizer.texts_to_sequences(lista_testo)
max_len = max(len(x) for x in lista_testo_tokenizer) # è 76
df['testo_token'] = tokenizer.texts_to_sequences(df['testo'])
print("La lunghezza prima il post padding è: ", len(df['testo_token'].iloc[1]))
df['testo_token_padding'] = pad_sequences(df['testo_token'], padding = "post", maxlen = max_len).tolist()
print("La lunghezza dopo il post padding è: ", len(df['testo_token_padding'].iloc[1]))
```

Figura 2.17: Creazione del vocabolario e aggiunta di zero padding

```
Frase selezionata:
[6, 21, 364, 3591, 3590, 986, 4698]
La lunghezza prima del Post Padding è:  7

[   6   21  364 3591 3590  986 4698    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0]
La lunghezza dopo il Post Padding è:  76
```

Figura 2.18: Esempio di applicazione di zero padding in post

Come possiamo notare in figura 2.18, la lunghezza della frase selezionata prima del post padding è 7, mentre dopo l'applicazione è stata trasformata in un array di 76 elementi che rappresenta la lunghezza massima delle frasi in tutto il dataset. I valori ottenuti andranno a rappresentare la ventiduesima feature, e saranno forniti come input del livello Embedding di Keras.

2.6 Elaborazione e Splitting dei dati

Dopo aver effettuato la pulizia del testo e creato le ventidue feature, ho notato come all'interno del dataset fossero presenti prima tutti i testi non contenenti sentimenti di odio (con classe 0) e poi tutti quelli con classe 1. Questo potrebbe essere un problema quando si daranno come input al modello, poichè imparerebbe a riconoscere sempre prima una classe a dispetto dell'altra. La soluzione è stata applicare uno shuffle al dataframe così da fare in modo che le righe siano ben mischiate tra di loro.

```
# Filtro ed ottengo solamente le colonne da dare in input al modello inclusa la colonna odio
df = df[['testo_token_padding','Feature_1','Feature_2','Feature_3','Feature_4',
         'Feature_5','Feature_6','Feature_7','Feature_8','Feature_9','Feature_10',
         'Feature_11','Feature_12', 'Feature_13','Feature_14','Feature_15',
         'Feature_16','Feature_17', 'Feature_18', 'Feature_19', 'Feature_20','Feature_21',
         'odio']]

#Ottengo i valori dal dataframe
X = df.iloc[:,0:22].values
Y = df.iloc[:,22].values

#Effettuo lo splitting, 80% al training e 20% al testing
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)


# Il primo elemento della lista rappresenta la frase alla quale applicare l'embedding
X_train_embedding = np.array([item[0] for item in X_train])
# Tutti i restanti elementi rappresentano le feature calcolate sulla frase in questione
X_train_feature = np.array([item[1:] for item in X_train])

#Si applica la stessa procedura per il test set.
X_test_embedding = np.array([item[0] for item in X_test])
X_test_feature = np.array([item[1:] for item in X_test])
```

Figura 2.19: Suddivisione dei dati relativi ai sentimenti di odio

Il passo successivo è stato quello di ordinare in maniera opportuna le 22 feature calcolate e la colonna 'odio' (Risp. 'Stereotipi') da prevedere, per poi andare ad ottenere i valori direttamente dal dataframe tramite il metodo *values*. Ho poi assegnato l'80% dei dati al training set ed il restante 20% al test set. Fondamentale è stato il fatto di suddividere anche gli input da fornire al modello, in quanto in prima posizione degli array X_train e X_test è presente l'input da fornire al livello Embedding di Keras, mentre nelle restanti posizioni (esclusa la ventiduesima colonna relativa alla label 'odio' (Risp. 'Stereotipi')) sono presenti le feature ricavate direttamente dal testo, che avranno bisogno di un livello di input apposito con un quantitativo di neuroni pari al loro numero.

Capitolo 3

Esperimenti

3.1 Modello Artificial Neural Network (ANN)

Un approccio alla costruzione di una ANN consiste nell'effettuare svariati esperimenti cambiando di volta in volta la selezione delle variabili e l'apprendimento dei parametri mediante alcune scelte:

- Quali variabili di input verranno utilizzate
- Come saranno rappresentati i risultati
- Quanti strati nascosti deve contenere la rete
- Quanti nodi devono essere presenti in ogni strato nascosto

Le reti proposte sono tutte 'Fully Connected Networks'; ogni neurone è collegato con tutti quelli presenti nel livello successivo. I modelli sono caratterizzati da due livelli di input:

- Input per il livello Embedding:

- Input per le Feature

Il livello Embedding sarà inizializzato con pesi casuali e imparerà un embedding per tutte le parole passate come vettore numerico. Questo livello necessita di tre parametri obbligatori:

- input_dim: Dimensione del vocabolario (numero di parole univoche all'interno del testo), rappresentata dalla variabile vocab_size nel progetto.

- output_dim: Dimensione dello spazio vettoriale in cui verranno incorporate le parole; ogni parola sarà rappresentata da un vettore di questa dimensione.

- input_length: Dimensione dell'input; in questo caso sarà 76 dato che la frase più lunga in tutto il dataset è composta da quel numero di parole.

Durante l'addestramento della rete neurale verranno modificati anche i pesi del livello Embedding se si imposta il parametro trainable a True. Il risultato di questo livello è un vettore a due dimensioni che contiene un vettore per ogni parola nel testo. Dato che si vuole collegare una ANN direttamente al risultato di questo layer, è necessario utilizzare un livello di tipo Flatten() che permetterà di compattare la matrice 2D ottenuta ad un vettore 1D. Il secondo livello di input invece, sarà caratterizzato da 21 neuroni, che rappresentano le ventuno feature prese in input per ogni record. I livelli di input saranno poi concatenati l'uno con l'altro e verranno poi aggiunti consecutivamente tre livelli di tipo Dense(); nello specifico, due livelli nascosti aventi rispettivamente 64 e 32 neuroni entrambi con funzione d'attivazione 'Relu' ed un livello di output avente 2 neuroni con funzione d'attivazione 'softmax'.

Per quanto concerne l'addestramento del modello, sono state effettuate 3 epoche con un batch_size pari a 128. L'ottimizzatore utilizzato nel processo è stato Adam, come loss function la categorical_crossentropy e come metrica per il calcolo delle performance Accuracy.

```
from keras.utils import to_categorical
y_train = to_categorical(y_train, 2)
y_test_cat = to_categorical(y_test, 2)
embedding_dim = 76

#Inizio la definizione del modello
input_testo = Input(shape=(max_len,))

x = Embedding(vocab_size, embedding_dim, input_length = max_len, trainable = True)(input_testo)

input_feature = Input(shape=(X_train_feature.shape[1],))

model_final = Concatenate()([x, input_feature])
model_final = Dense(64, activation='relu', bias_initializer='zeros')(model_final)
model_final = Dense(32, activation = "relu", bias_initializer='zeros') (model_final)
model_final = Dense(2, activation='softmax', bias_initializer='zeros')(model_final)
model_final = Model([input_testo,input_feature], model_final)
model_final.compile(loss="categorical_crossentropy", optimizer = 'adam', metrics = ["accuracy"])

print(model_final.summary())
history = model_final.fit(x=[X_train_embedding,X_train_feature], y = np.array(y_train),
                          batch_size = 128, epochs=3, verbose = 1,
                          validation_split=0.2)
```

Figura 3.1: Definizione del modello dedicato alla previsione dei sentimenti di odio

Dopo aver sperimentato con il modello sopra citato sia la previsione di stereotipi che di sentimenti di odio, ho voluto testare altri modelli addestrati solamente con uno dei due input precedentemente descritti per annotare la variazione delle performance con e senza la presenza di feature aggiuntive. I risultati ottenuti saranno mostrati nel dettaglio nella parte finale della relazione, e mostrano come il modello dedicato alla previsione di stereotipi sia più performante quando addestrato solamente con il testo in formato numerico, senza prendere in considerazione le feature del testo stesso. Inoltre, ho apportato leggere modifiche al numero di hidden layer presenti nel modello, andando a rimuovere il livello nascosto contenente 64 neuroni e svolgendo 12

epoche di allenamento. Riassumendo, le performance migliori per la previsione dei sentimenti di odio sono state ottenute tramite il modello che prende in input sia il testo in formato numerico sia le feature calcolate, mentre per quanto riguarda la previsione di stereotipi, il modello che si è dimostrato più performante è stato quello che ha come input solamente il testo in formato numerico ottenuto dopo l'applicazione del metodo *texts_to_sequences* di Keras.

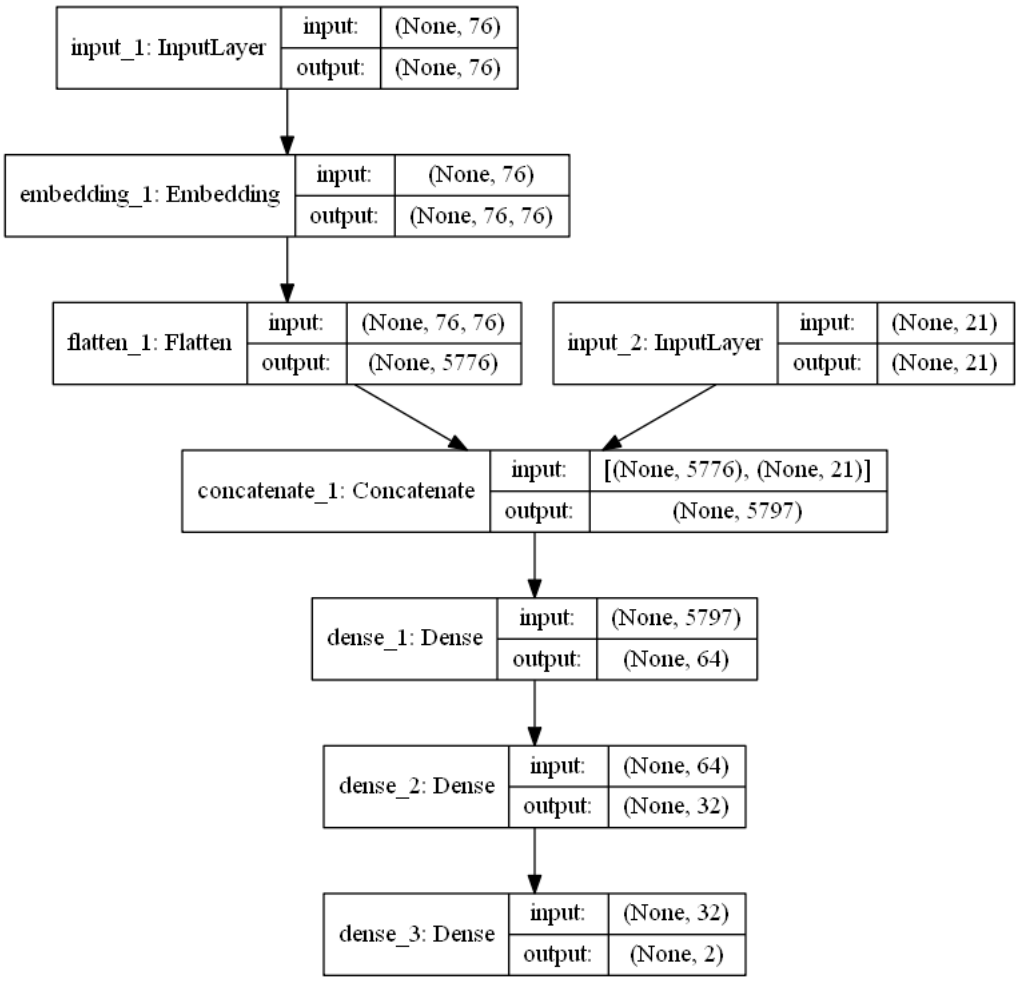

Figura 3.2: Struttura del modello ANN per la previsione dei sentimenti di odio

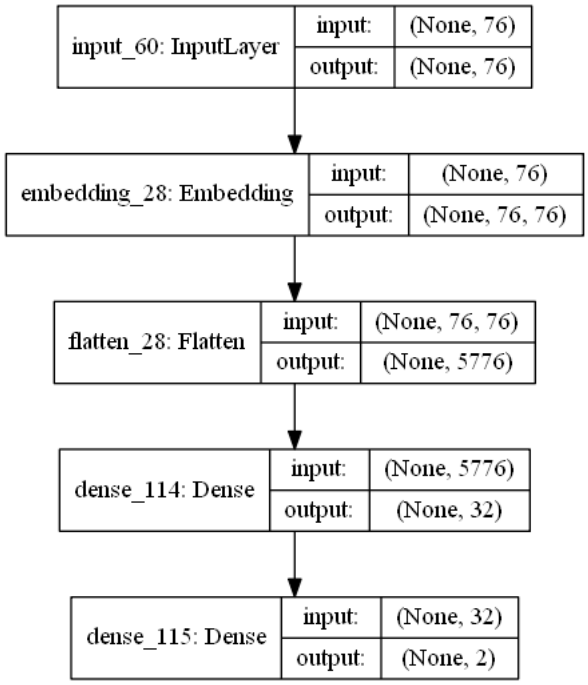

Figura 3.3: Struttura del modello ANN per la previsione di stereotipi

3.2 Salvataggio ed utilizzo dei modelli creati

Al fine di utilizzare i modelli addestrati per la previsione dell'input ottenuto da tastiera ho salvato prima la struttura del modello in formato *json*, e successivamente il valore dei pesi degli archi in formato *h5*. Questo permetterà di caricare i modelli già addestrati su un nuovo file, evitando di eseguire l'addestramento ad ogni esecuzione del programma.

```
{"class_name": "Model", "config": {"name": "model_58", "layers": [{"name": "input_116", "class_name": "InputLayer",
"config": {"batch_input_shape": [null, 76], "dtype": "float32", "sparse": false, "name": "input_116"}, "inbound_nodes":
[]}, {"name": "embedding_59", "class_name": "Embedding", "config": {"name": "embedding_59", "trainable": true,
"batch_input_shape": [null, 76], "dtype": "float32", "input_dim": 17777, "output_dim": 76, "embeddings_initializer":
{"class_name": "RandomUniform", "config": {"minval": -0.05, "maxval": 0.05, "seed": null}}, "embeddings_regularizer":
null, "activity_regularizer": null, "embeddings_constraint": null, "mask_zero": false, "input_length": 76},
"inbound_nodes": [[["input_116", 0, 0, {}]]]}, {"name": "flatten_56", "class_name": "Flatten", "config": {"name":
"flatten_56", "trainable": true, "data_format": "channels_last"}, "inbound_nodes": [[["embedding_59", 0, 0, {}]]]},
{"name": "input_117", "class_name": "InputLayer", "config": {"batch_input_shape": [null, 20], "dtype": "float32",
"sparse": false, "name": "input_117"}, "inbound_nodes": []}, {"name": "concatenate_58", "class_name": "Concatenate",
"config": {"name": "concatenate_58", "trainable": true, "axis": -1}, "inbound_nodes": [[["flatten_56", 0, 0, {}],
["input_117", 0, 0, {}]]]}, {"name": "dense_172", "class_name": "Dense", "config": {"name": "dense_172", "trainable":
true, "units": 64, "activation": "relu", "use_bias": true, "kernel_initializer": {"class_name": "VarianceScaling",
"config": {"scale": 1.0, "mode": "fan_avg", "distribution": "uniform", "seed": null}}, "bias_initializer": {"class_name":
"Zeros", "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null,
"kernel_constraint": null, "bias_constraint": null}, "inbound_nodes": [[["concatenate_58", 0, 0, {}]]]}, {"name":
"dense_173", "class_name": "Dense", "config": {"name": "dense_173", "trainable": true, "units": 32, "activation": "relu",
"use_bias": true, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan_avg",
"distribution": "uniform", "seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}},
"kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null,
"bias_constraint": null}, "inbound_nodes": [[["dense_172", 0, 0, {}]]]}, {"name": "dense_174", "class_name": "Dense",
"config": {"name": "dense_174", "trainable": true, "units": 2, "activation": "softmax", "use_bias": true,
"kernel_initializer": {"class_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan_avg", "distribution":
"uniform", "seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_regularizer": null,
"bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null, "bias_constraint": null},
"inbound_nodes": [[["dense_173", 0, 0, {}]]]}], "input_layers": [["input_116", 0, 0], ["input_117", 0, 0]],
"output_layers": [["dense_174", 0, 0]]}, "keras_version": "2.2.4", "backend": "tensorflow"}
```


Figura 3.4: Modello dedicato alla previsione di sentimenti di odio in formato json

```
model_json = model.to_json()
with open("Modello_Stereotipi_Salvato/modello_stereotipi.json", "w+") as json_file:
    json_file.write(model_json)

model.save_weights("Modello_Stereotipi_Salvato/modello_stereotipi.h5")
print("Il modello Stereotipi è stato salvato correttamente")
```

Figura 3.5: Salvataggio del modello dedicato agli stereotipi

```
# Carico il modello addestrato per riconoscere sentimenti di odio
json_odio = open('Modello_Odio_Salvato/modello_odio.json', 'r')
modello_json_odio = json_odio.read()
json_odio.close()
loaded_model_odio = model_from_json(modello_json_odio)
loaded_model_odio.load_weights("Modello_Odio_Salvato/modello_odio.h5")
print("Il modello 'Odio' è stato caricato correttamente")

#Carico il modello addestrato per riconoscere gli stereotipi
json_stereotipi = open('Modello_Stereotipi_Salvato/modello_stereotipi.json', 'r')
modello_json_stereotipi = json_stereotipi.read()
json_stereotipi.close()
loaded_model_stereotipi = model_from_json(modello_json_stereotipi)
loaded_model_stereotipi.load_weights("Modello_Stereotipi_Salvato/modello_stereotipi.h5")
print("Il modello 'Stereotipi' è stato caricato correttamente")
```

Figura 3.6: Caricamento dei modelli pre-allenati

Una volta aver caricato i modelli pre-allenati, viene richiesto all'utente di inserire una frase, la quale sarà filtrata tramite la funzione *clean_text* e saranno calcolate tutte le ventuno feature precedentemente descritte. A questo punto, dato che i modelli lavorano con due input diversi, al modello dedicato alla previsione dei sentimenti di odio saranno sottoposte in input sia il testo in formato numerico sia le ventuno feature, mentre il modello dedicato alla previsione di stereotipi prenderà in input solamente il testo in formato numerico. Vengono infine mostrati dei test effettuati tramite l'inserimento di una frase da tastiera e le rispettive predizioni fornite dai modelli.

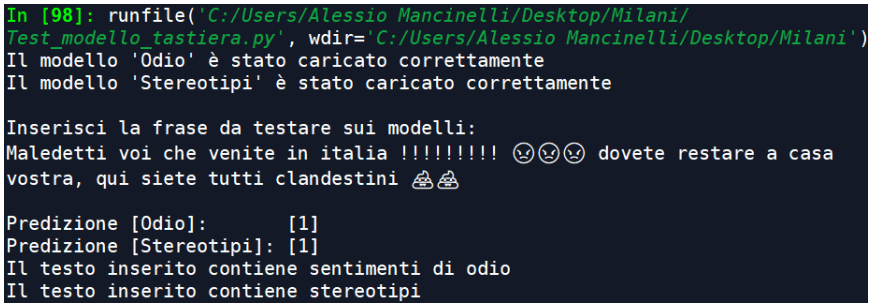

Figura 3.7: Prima previsione del modello su input passato da tastiera

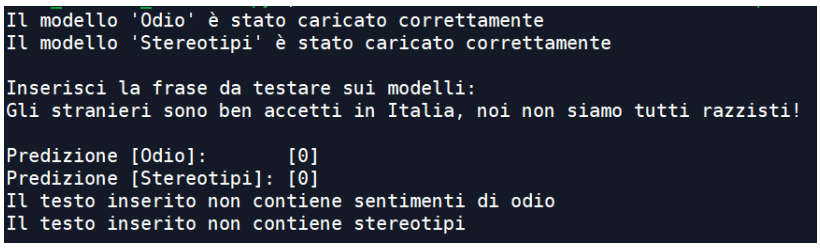

Figura 3.8: Seconda previsione del modello su input passato da tastiera

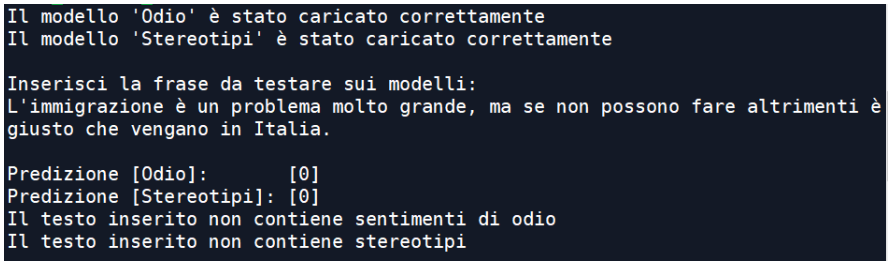

Figura 3.9: Terza previsione del modello su input passato da tastiera

3.3 Metriche di valutazione

La valutazione dei modelli proposti è stata effettuata in base alla matrice di confusione.

	Previsione=0	Previsione=1
ValoreDesiderato=0	TP	FN
ValoreDesiderato=1	FP	TN

Come si può notare dalla tabella il valore che il modello prevederà è posto nelle colonne, mentre il valore desiderato, e quindi reale, è rappresentato dalle righe. Questo permette di descrivere una matrice di confusione a due classi (classificatore binario) {0} e {1}. Le performance vengono calcolate in base al valore dei termini interni alla tabella: TP, FN, FP e TN; segue quindi una breve descrizione di questi.

- True Positive (TP) : Tutti i campioni che il modello ha previsto con esito 0, e il valore desiderato era 0.
- True Negative (TN) : Tutti i campioni che il modello ha previsto con esito 1, e il valore desiderato era 1.

- False Negative (FP) : Tutti i campioni che il modello ha previsto 0, ma il valore desiderato era 1.
- False Positive (FN) : Tutti i campioni che il modello ha previsto 1, ma il valore desiderato era 0.

Tutti questi parametri permettono di calcolare delle metriche di performance come:

$$Accuracy = \frac{TP + TN}{TN + FP + FN + TP}$$

L'accuracy può essere vista come rapporto tra tutti i dati stimati correttamente (TP+FN) rispetto al totale dei campioni considerati (TN+FP+FN +TP), ovvero la percentuale di classificazioni corrette.

$$Precision = \frac{TP}{TP + FP}$$

La precision può essere definita come l'abilità del classificatore nel prevedere l'assegnamento della classe 0 (Risp. 1), ovvero il numero di tutti i campioni previsti correttamente con classe 0 (Risp. 1) (TP) diviso il numero totale (TP+FP) delle previsioni effettuate con classe 0 (Risp. 1).

$$Recall = \frac{TP}{TP + FN}$$

La recall è l'abilità del classificatore nel prevedere l'assegnamento della classe 0 (Risp. 1) (TP) rispetto al totale dei campioni che avrebbe potuto prevedere sempre con classe 0 (Risp. 1) (TP+FN).

$$F - score = \frac{2 * Recall * Precision}{Recall + Precision}$$

L'F-score esprime la media armonica tra la Precision e il Recall.

Un altro valore che si riporterà all'interno delle tabelle dei risultati è il Support, ovvero il numero di campioni analizzati per ogni classe.

Capitolo 4

Valutazione dei risultati

4.1 Previsione dei sentimenti di odio

I risultati del modello dedicato alla previsione dei sentimenti di odio che prende in input sia le feature che il testo in formato numerico sono:

	Precision	Recall	F-score	Support
0	78%	79%	78%	793
1	71%	69%	70%	577

Accuracy: 75% su 1370 campioni.

Valori della matrice di confusione:

627	166
178	399

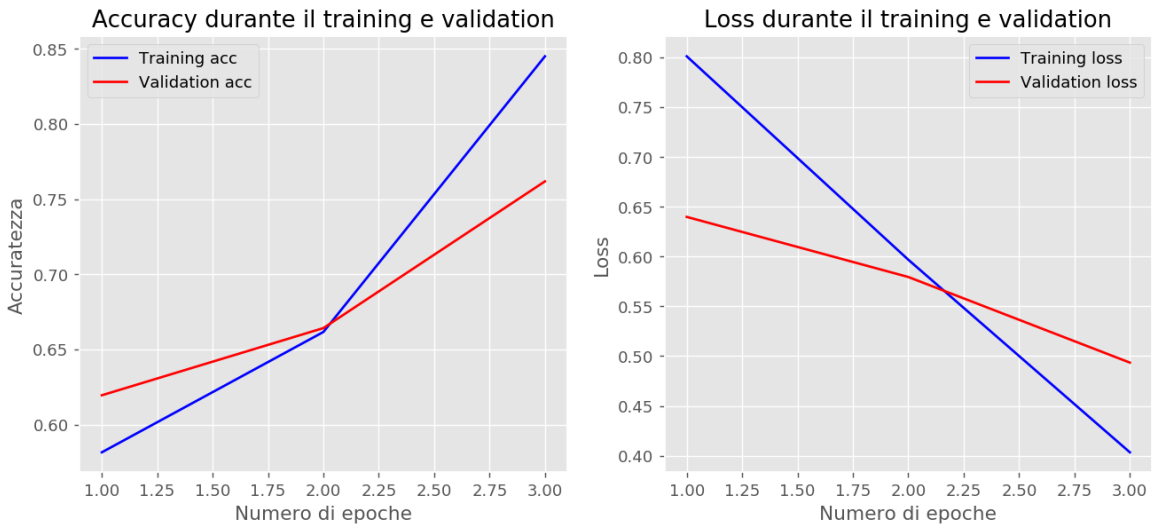

Figura 4.1: Accuracy e Loss durante l'addestramento del primo modello dedicato alla previsione dei sentimenti di odio

I risultati del modello dedicato alla previsione dei sentimenti di odio che prende in input solamente le feature calcolate direttamente dal testo sono:

	Precision	Recall	F-score	Support
0	66%	77%	71%	793
1	59%	47%	52%	577

Accuracy: 64% su 1370 campioni.

Valori della matrice di confusione:

607	186
307	270

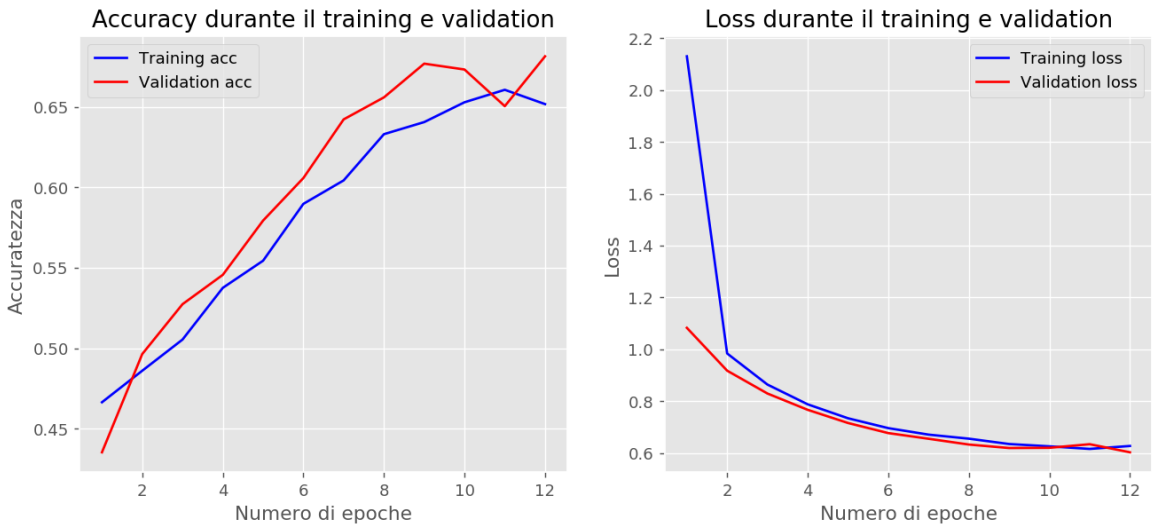

Figura 4.2: Accuracy e Loss durante l'addestramento del secondo modello dedicato alla previsione dei sentimenti di odio

I risultati del modello dedicato alla previsione dei sentimenti di odio che prende in input solamente il testo in formato numerico sono:

	Precision	Recall	F-score	Support
0	76%	82%	79%	793
1	72%	64%	68%	577

Accuracy: 74% su 1370 campioni.

Valori della matrice di confusione:

647	146
208	369

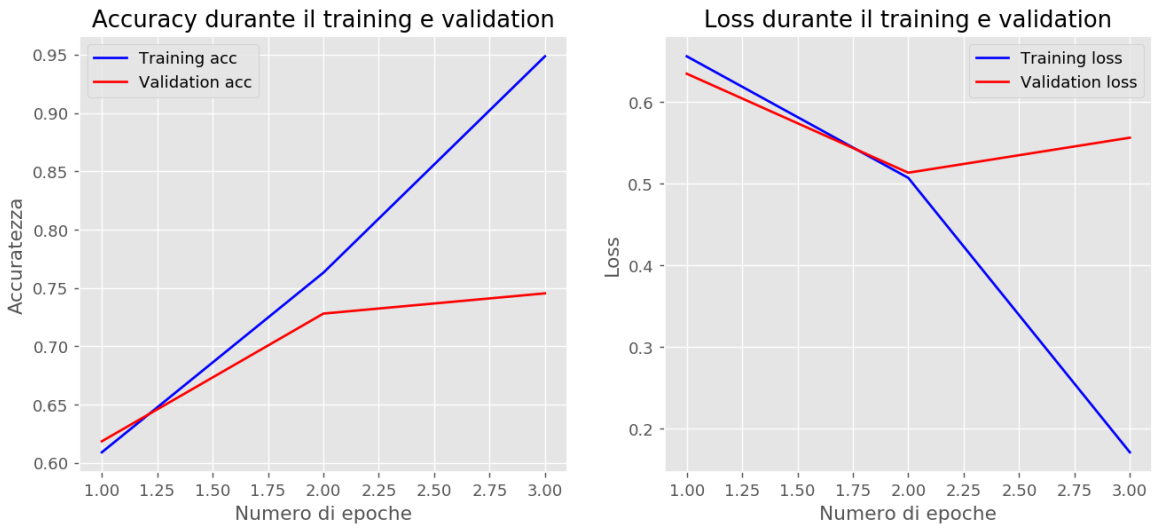

Figura 4.3: Accuracy e Loss durante l'addestramento del terzo modello dedicato alla previsione dei sentimenti di odio

4.2 Previsione di stereotipi

I risultati del modello dedicato alla previsione di stereotipi che prende in input sia le feature che il testo in formato numerico sono:

	Precision	Recall	F-score	Support
0	69%	72%	70%	753
1	64%	61%	62%	617

Accuracy: 67% su 1370 campioni.

Valori della matrice di confusione:

541	212
242	375

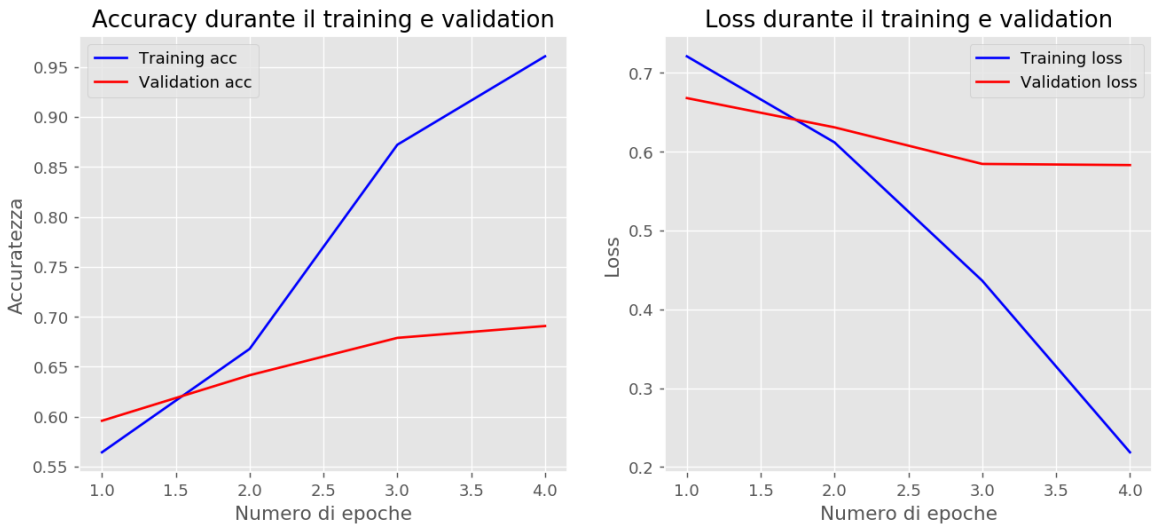

Figura 4.4: Accuracy e Loss durante l'addestramento del terzo modello dedicato alla previsione di stereotipi

I risultati del modello dedicato alla previsione di stereotipi che prende in input solo le feature calcolate dal testo sono:

	Precision	Recall	F-score	Support
0	58%	92%	71%	753
1	65%	18%	28%	617

Accuracy: 59% su 1370 campioni.

Valori della matrice di confusione:

694	59
509	108

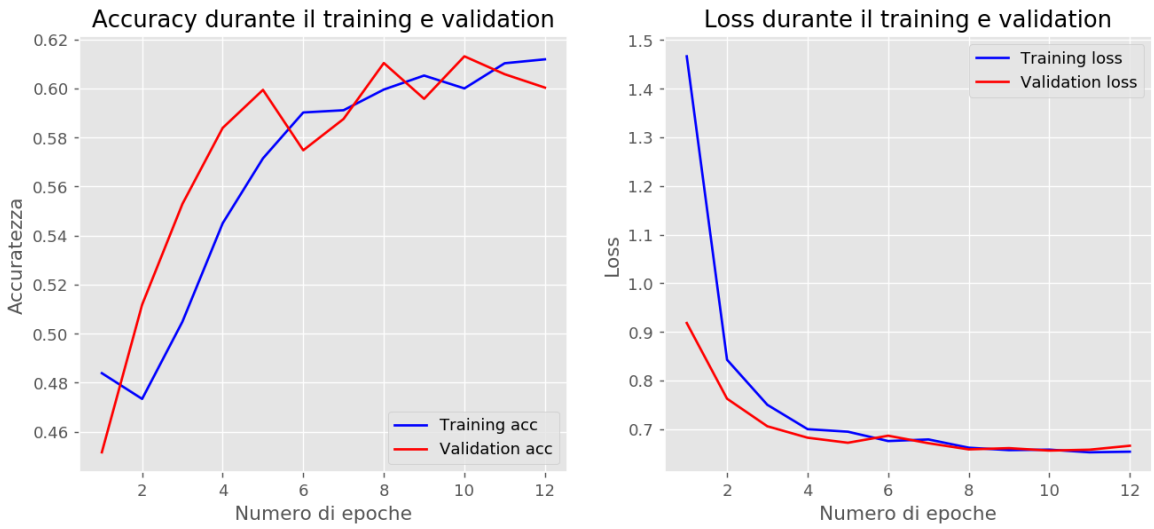

Figura 4.5: Accuracy e Loss durante l'addestramento del secondo modello dedicato alla previsione di stereotipi

I risultati del modello dedicato alla previsione di stereotipi che prende in input solo il testo in formato numerico sono:

	Precision	Recall	F-score	Support
0	70%	77%	73%	753
1	68%	59%	63%	617

Accuracy: 69% su 1370 campioni.

Valori della matrice di confusione:

578	175
253	364

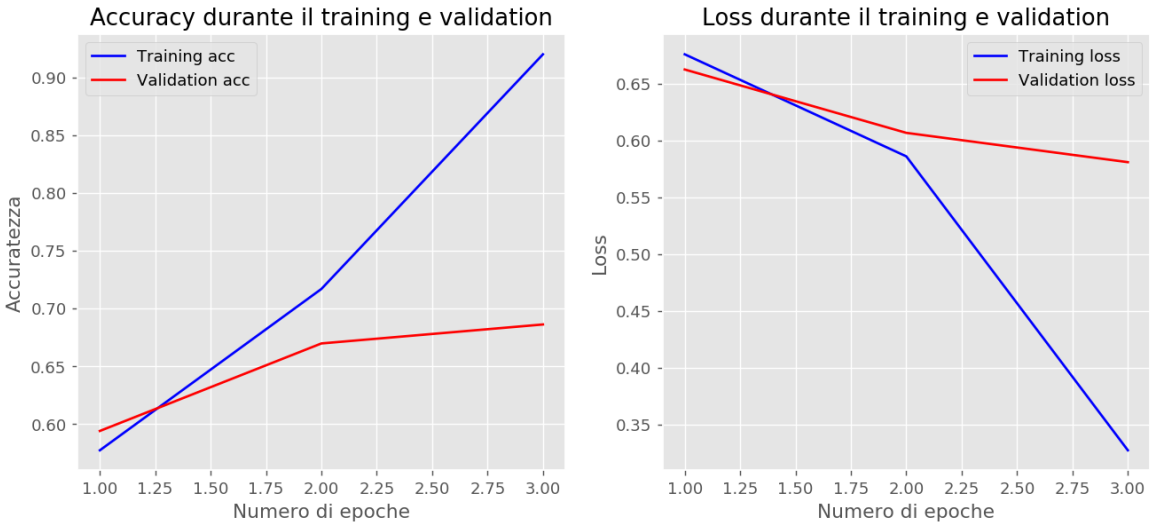

Figura 4.6: Accuracy e Loss del terzo modello dedicato alla previsione di stereotipi

Conclusioni

L'obiettivo di questo progetto era mettere in relazione la sintassi ed il modo di scrittura delle persone sul social network Twitter con la presenza di sentimenti di odio e/o stereotipi. Tutti i modelli hanno mostrato discrete prestazioni nella risoluzione del problema in oggetto, specialmente il modello dedicato ai sentimenti di odio che grazie all'aggiunta delle feature ha subito un leggero miglioramento di tutte le performance. La stessa cosa però non è accaduta con il modello dedicato alla previsione di stereotipi, che fornisce performance migliori quando addestrato solamente con il testo in formato numerico. Il salvataggio dei modelli in formato json ha poi permesso il loro caricamento in un differente file, così da evitare l'addestramento ogni qualvolta si volesse testare la previsione di una frase inserita da tastiera.