

UNIVERSITÀ DEGLI STUDI DI PERUGIA
Dipartimento di Matematica e Informatica

CORSO DI LAUREA SPECIALISTICA IN INFORMATICA



Progetto di Intelligent Systems

Studenti:

Alex Dominici

Professore:

Prof. Alfredo Milani

Anno Accademico 2019–2020

Introduzione

Il problema preso in considerazione per il progetto riguarda lo svolgimento di un task appartenente alla challenge di Evalita 2020. Data l'enorme quantità di contenuti generati dagli utenti sul Web, e in particolare sui social media, il problema di individuare e quindi limitare la diffusione del discorso di odio contro le donne sta rapidamente diventando fondamentale soprattutto per l'impatto sociale del fenomeno. L'attività AMI ha quindi lo scopo di identificare automaticamente i contenuti misogini di Twitter in italiano e quindi si tratta di un problema di apprendimento supervisionato di tipo NLP (Natural Language Processing).

Diamo adesso una breve definizione di misogino e di aggressivo, il primo viene preso in considerazione se una frase contiene parole che screditano le donne in qualsiasi modo o approccio. Diversi esempi di questo odio contro le donne vengono manifestati attraverso l'esclusione sociale, la discriminazione sessuale, l'ostilità e l'oggettificazione. Mentre per quanto riguarda la definizione di aggressivo questa viene strettamente legata alla presenza di misoginia, dunque solo in caso di essa potrà esserci un comportamento aggressivo o meno. Si ha un comportamento aggressivo quando si usano parole particolarmente offensive verso un determinato individuo.

Misogino	Aggressivo
Parole che screditano le donne	Parole particolarmente offensive
Esclusione sociale	Comportamento violento
Discriminazione sessuale	Affermare proprie opinioni come fatti
Ostilità	Presunzione

Figura 1: Definizioni misogino e aggressivo

I capitoli successivi mostrano come raggiungere questo obiettivo, dove prima di tutto è stato recuperato il dataset di training contenenti i 5000 tweets etichettati come misogini e non misogini, dei quali è stata fatta una profonda analisi e pulizia. Successivamente sono stati progettati diversi modelli supervisionati, allenandoli con tali tweets. Infine una volta che il modello addestrato ha raggiunto dei buoni risultati viene implementata una interfaccia utente che consenta di sottoporre e classificare un testo qualsiasi digitato dall'utente.

Capitolo 1

Linguaggio e Librerie utilizzate

Il linguaggio di programmazione per lo svolgimento del progetto è stato Python, il quale è un linguaggio di alto livello e orientato a oggetti, tipicamente utilizzato a sviluppare applicazioni distribuite, scripting e sistemi di testing. In questo progetto è stato proprio utilizzato per lo sviluppo dei modelli di apprendimento attraverso le seguenti librerie.

1.1 Pandas

Libreria che consente di utilizzare diversi strumenti per l'analisi e la gestione dei dati scritti in python. Essa è open source e viene fornita con molteplici strutture dati, come Series e Dataframes, che ci aiutano non solo a rappresentare i dati in modo efficiente, ma anche a manipolarli in vari modi. Inoltre permette di recuperare facilmente i dati da diverse fonti come database SQL, CSV O file JSON ed è in grado di organizzare ed etichettare i dati tramite dei metodi intelligenti di allineamento e indicizzazione che ne consentono una semplice lettura.

Quando i dati dopo che sono stati recuperati e memorizzati, vengono fornite diverse funzioni per pulire il set di dati, recuperare i valori mancanti, analizzarli, trasformarli e un set di operazioni statistiche per eseguire semplici analisi.

Pandas è dunque una libreria molto conosciuta ed usata per il recupero e per il pre-processing dei dati in modo da usarli successivamente in altre librerie come Scikit-learn o Tensorflow.

1.2 NumPy

Numpy ovvero Numeric Python, risulta essere il pacchetto fondamentale per il calcolo scientifico con Python ed è una delle più grandi librerie di calcolo matematico e scientifico. Una delle funzionalità più importanti di NumPy è la sua interfaccia Array. Questa interfaccia può essere utilizzata per fornire strutture dati per il calcolo e l'implementazione di vettori e matrici multi-dimensionali. Inoltre questa libreria fornisce diverse funzioni matematiche.

1.3 Scikit-learn

Anche questa è una libreria open source scritta in Python ed è in grado di interagire con altre librerie come Pandas, NumPy e Keras. Essa principalmente permette di utilizzare diversi algoritmi (ad esempio di classificazione, di regressione o di clustering) che consentono di costruire modelli per l'apprendimento sia supervisionato che non. Inoltre concede la possibilità di costruire la matrice di confusione attraverso funzioni e metodi per la suddivisione dei dati e per il calcolo delle performance dei modelli.

1.4 Keras

Libreria scritta in Python che fornisce un modo semplice per esprimere l'apprendimento automatico e la costruzione di reti neurali.

Oltre alla facilità di apprendimento e di costruzione dei modelli, essa ha il vantaggio di essere adottata da un gran numero di utenti, di essere integrata con cinque diversi motori di back-end (come TensorFlow, CNTK, Theano, MXNet e PlaidML), di avere un supporto per GPU multiple e la capacità di training distribuito.

Keras dunque non esegue le proprie operazioni di basso livello, come ad esempio i prodotti tra tensori ma si affida ad una libreria specializzata per farlo, quindi viene permesso il collegamento a diversi back-end, come citato precedentemente, in modo da gestire il problema modularmente e impedendo che l'implementazione si leghi ad una singola libreria. Infine essa fornisce alcune utilità per l'elaborazione di set di dati, per la compilazione di modelli, per la valutazione dei risultati e per la visualizzazione di grafici.

Capitolo 2

Procedimento risolutivo

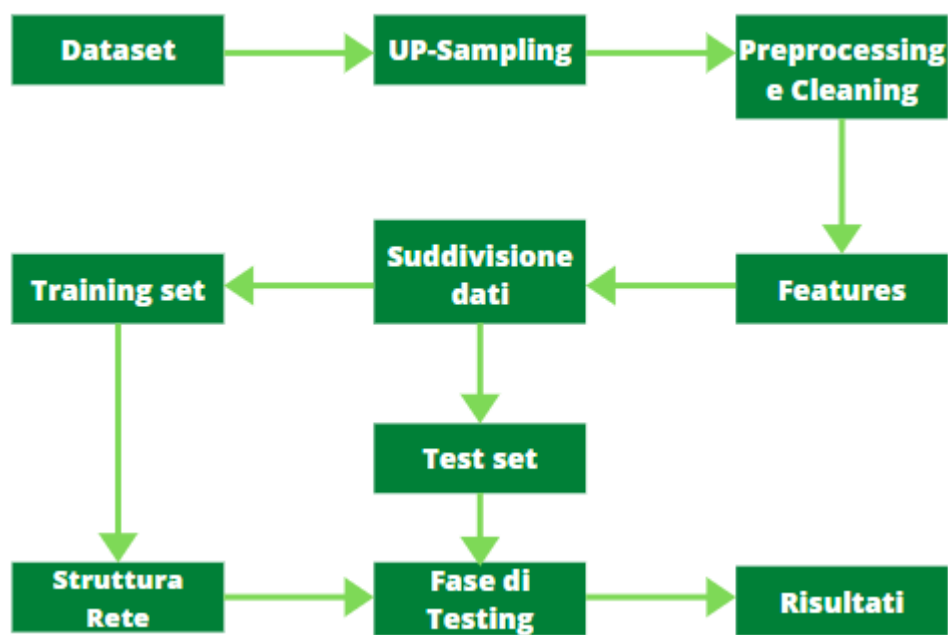


Figura 2.1: Fasi principali della soluzione

In questo capitolo verranno illustrate le fasi principali per raggiungere l'obiettivo proposto, partendo dall'ottenimento del Dataset dal sito di Evali-

ta, passando per l'Up-Sampling dei dati e facendone un pre-processing e un cleaning accurato in modo tale da poter arrivare ad ottenere delle features e il testo in formato numerico. Vengono poi splittati i dati in test e training set, in modo tale che quest'ultimi possano essere usati per allenare i modelli creati per classificare le frasi. Tramite il test set invece è stata svolta la fase di testing valutandone anche i relativi risultati.

2.1 Visualizzazione del dataset

Il dataset recuperato dal sito di Evalita è stato distribuito in formato TSV e per poterlo visualizzare e lavorarci è stata utilizzata la citata libreria Pandas. Come mostrato in Figura 3.2 il dataset contiene 5000 tweets in lingua italiana e questi vengono etichettati in base alla misoginia e l'aggressività.

	id	text	misogynous	aggressiveness
0	1	@KassemAmin4 @Laylasexgdr Fatti trovare te lo...	1	1
1	2	@meb Tu dovresti ricominciare dai semafori a f...	1	1
2	3	Amore,sei presentabile? Xchè così via Skype ti...	1	1
3	4	@Il_nulla Salvo poi mandare la culona a Mosca,...	1	0
4	5	@GiorgiaMeloni @FratellidItalia Vediamo Gentil...	1	1
...
4995	4996	non ci posso credere sono queste le cose che m...	0	0
4996	4997	sono così incazzata! dovete smetterla di consi...	0	0
4997	4998	SENTITE PORCA PUTTANA TAE NON PUÒ POSTARE SUL ...	0	0
4998	4999	ma vaffanculo senti che cazzo me li fai te i c...	0	0
4999	5000	MA CHE CAZZO FATE VI SIETE MANGIATE IL CERVELL...	0	0

5000 rows × 4 columns

Figura 2.2: Dataset di training

Inoltre ai fini del progetto sono state mantenute solo le colonne 'text',

'mosogynous' e 'aggressiveness', dato che l'Id dei tweets non risulta essere utile per la classificazione.

2.2 Up-sampling delle etichette

Vediamo adesso come sono state ripartite queste classi/etichette all'interno del dataset appena ottenuto.

Per quanto riguarda la classe misogina abbiamo 2337 tweet etichettati come tali mentre 2663 etichettati come non misogina. Invece per quanto riguarda la classe aggressività abbiamo uno sbilanciamento poichè 3217 tweets sono etichettati come non aggressivi mentre solo 1783 sono definiti aggressivi.

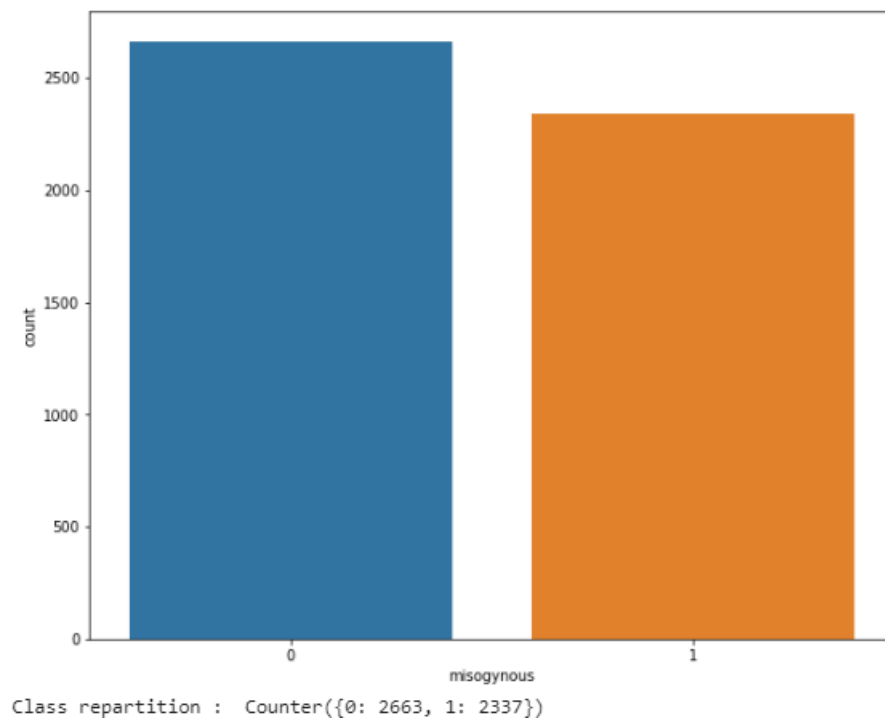


Figura 2.3: Ripartizione calsse misogina

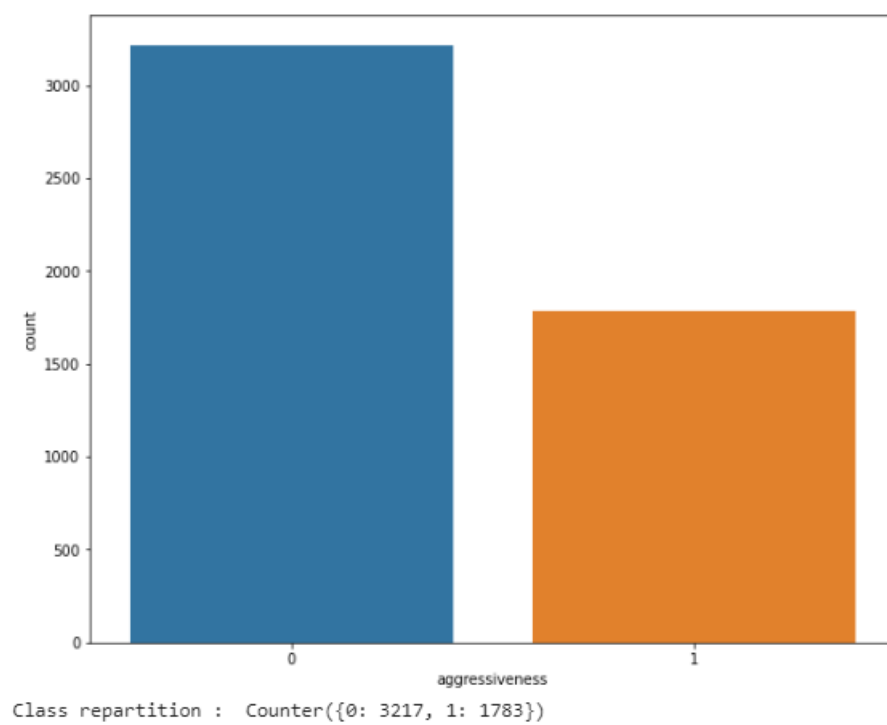


Figura 2.4: Ripartizione calsse aggressività

Prima di arrivare in questa fase di Up-sampling, sono stati realizzati dei modelli allenati sempre con lo stesso numero di tweets. Tuttavia vedendo che le prestazioni non risultavano ottimali per la previsione dell'aggressività ho ritenuto necessario eseguire questa operazione.

Dunque al fine di bilanciare il dataset è stato eseguito l'up-sampling relativo ai testi etichettati come "aggressiveness" in modo tale da poter incrementare il loro numero. Tutta via l'upsampling viene fatto fino ad un massimo del 20% dei dati già esistenti, poichè se maggiore influenzerebbe troppo l'intero dataset.

Per poter eseguire questo up-sampling è stata utilizzata la funzione "Resample" fornita dalla libreria, già citata, "Scikit-learn" la quale prendendo in input i testi etichettati come aggressivi restituisce una sequenza di copie ricampionate. Il numero di samples generati viene indicato come parametro nella funzione, in modo da poter individuare il valore corretto senza influenzare negativamente i dati.

Dopo l'upsampling il numero di tweets etichettati come aggressivi è pari a 2200 mentre quelli non aggressivi sono rimasti sempre 3217.

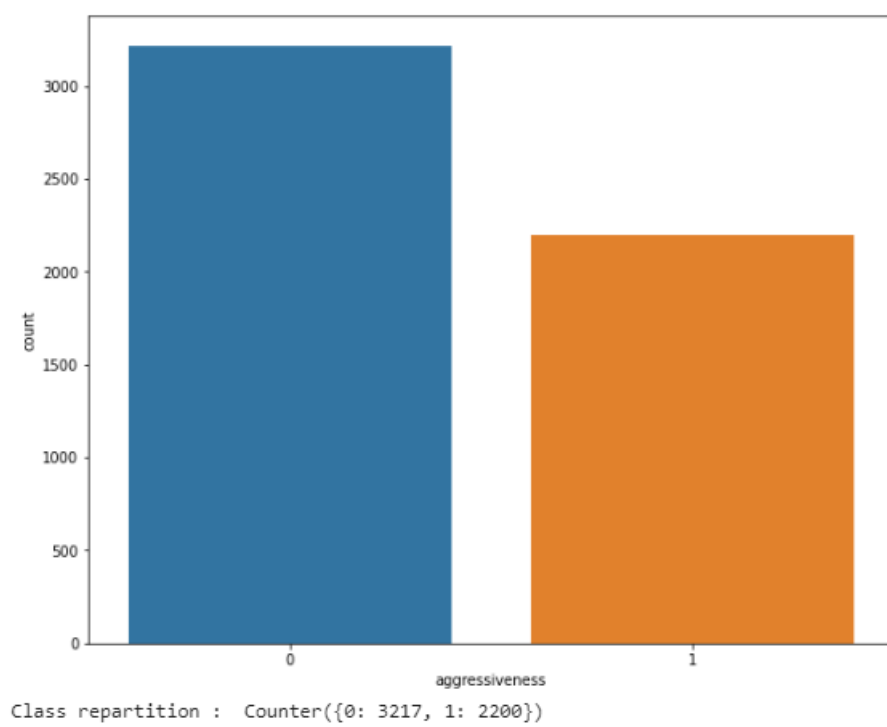


Figura 2.5: Ripartizione calsse aggressività dopo aver eseguito l'up-sampling

2.3 Pre-processing e cleaning dei dati

Per poter fare una buona classificazione, prima di trasformare il testo in formato numerico, ho eseguito un pre-processing per ogni tweet del data set, in modo da filtrarlo e ripulirlo dalle stopwords, da link "http" e tag "@", i quali non sono utili ai fini della classificazione. Il pre-processing dei dati è quindi il processo di conversione di quest'ultimi in qualcosa che un computer possa capire.

Per la rimozione delle stopwords, cioè parole comunemente usate ma che devono essere ignorate, viene adottata la libreria NLTK la quale fornisce file in diverse lingue contenenti per l'appunto tali parole da filtrare.

Inoltre ho rimosso tutti i caratteri speciali e sostituito le emoji con il relativo nome che le rappresenta. Per poter svolgere quest'ultima operazione ho sfruttato la libreria "Emot", la quale riconosce e mappa le eventuali emoji del testo in modo da aiutare il modello a riconoscerle e ad effettuare la classificazione.

L'ultimo punto del pre-processing riguarda lo Stemming, il quale risulta essere solitamente utile per la pulizia del testo. Esso è un processo di riduzione di una parola alla sua radice di parole, tuttavia questo processo non è stato più applicato poichè le prestazioni dei modelli peggioravano con la sua presenza.

Vediamo adesso alcuni esempi di come sono diventati i tweets dopo il loro pre-processing: Tuttavia, dopo la pulizia, alcune righe del dataframe sono diventate vuote, dunque eliminate per evitare di influenzare il futuro modello negativamente.

```

Testo originale:
@francyn9 @Cam4_IT Quanto sei porca Francyn ♥ Io ti prenderei più forte, bella tettona ♥ baci

Testo ripulito:
porca francyn heart_suit prenderei forte bella tettona heart_suit baci

-----

Testo originale:
@vittoriarisi Buongiorno tettona mia bella idea per Stasera SABATO 14 APRILE -- alla grande Fiera
Erotica sarò al Locale ALIBI 🌟?? a CONEGLIANO (TV) assolutamente da non mancare

Testo ripulito:
buongiorno tettona bella idea stasera sabato 14 aprile grande fiera erotica locale alibi sparkles
conegliano tv assolutamente mancare

-----

Testo originale:
Bella Culona!♥? https://t.co/BQI3VkfEmc

Testo ripulito:
bella culona red_heart

```

Figura 2.6: Esempio pulizia tweets

2.4 Ottenimento delle features

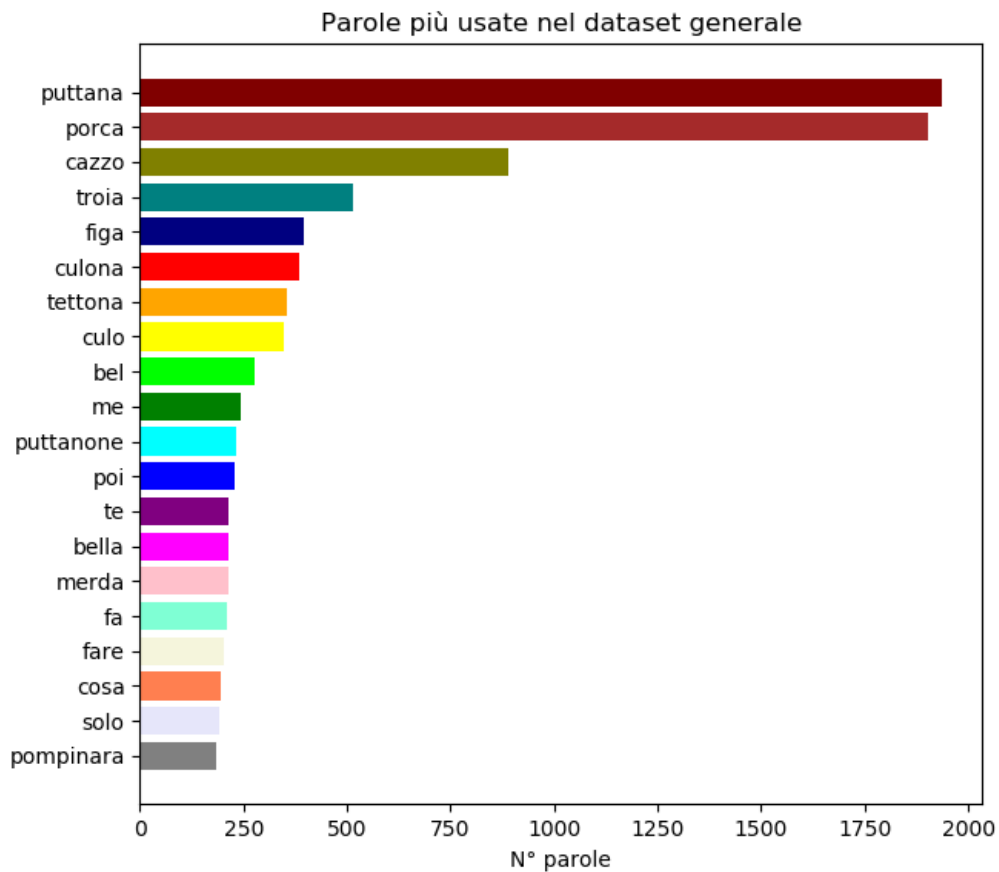
Un ulteriore studio nel dataset è stato rivolto alla raccolta delle caratteristiche più o meno importanti, che forniscono informazioni aggiuntive per il training dei modelli.

Le prime features identificate sono state il numero di punti esclamativi e di tag utilizzati in ogni tweet, dopo di che sono state aggiunte altre 13 caratteristiche che ho ritenuto utili ai fini dell'apprendimento, tra queste abbiamo:

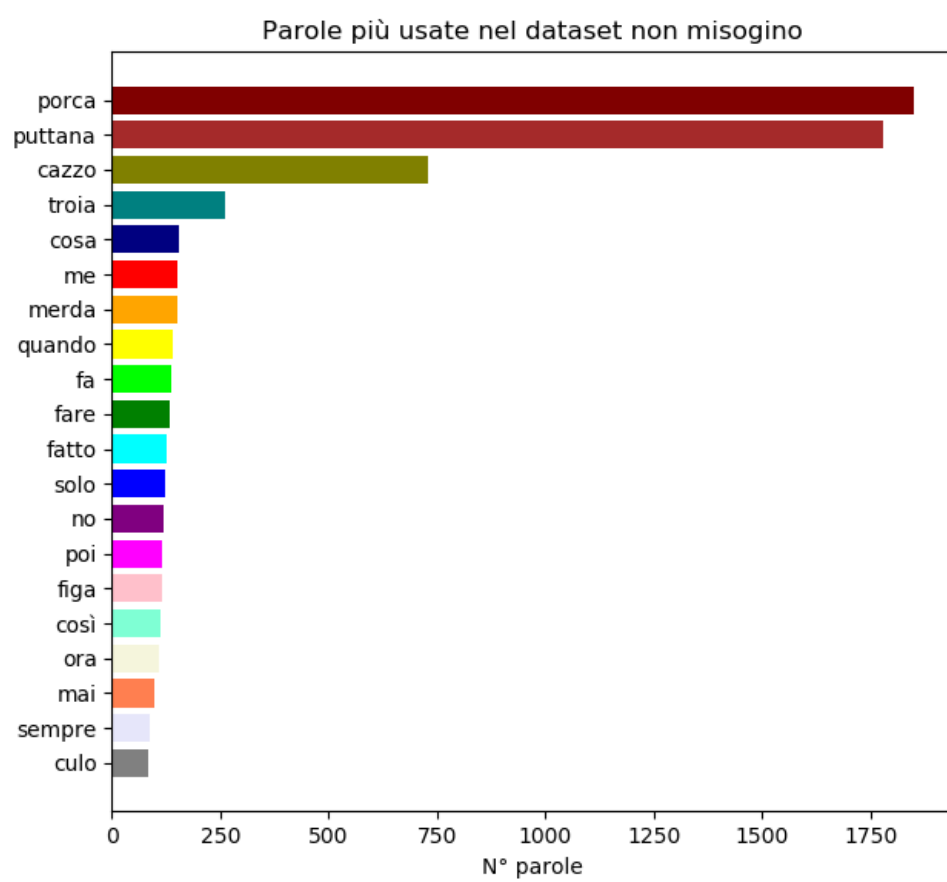
- Il numero di punti interrogativi.
- Il numero di hashtag.
- Il numero di parole utilizzate.
- Il numero di caratteri utilizzati.

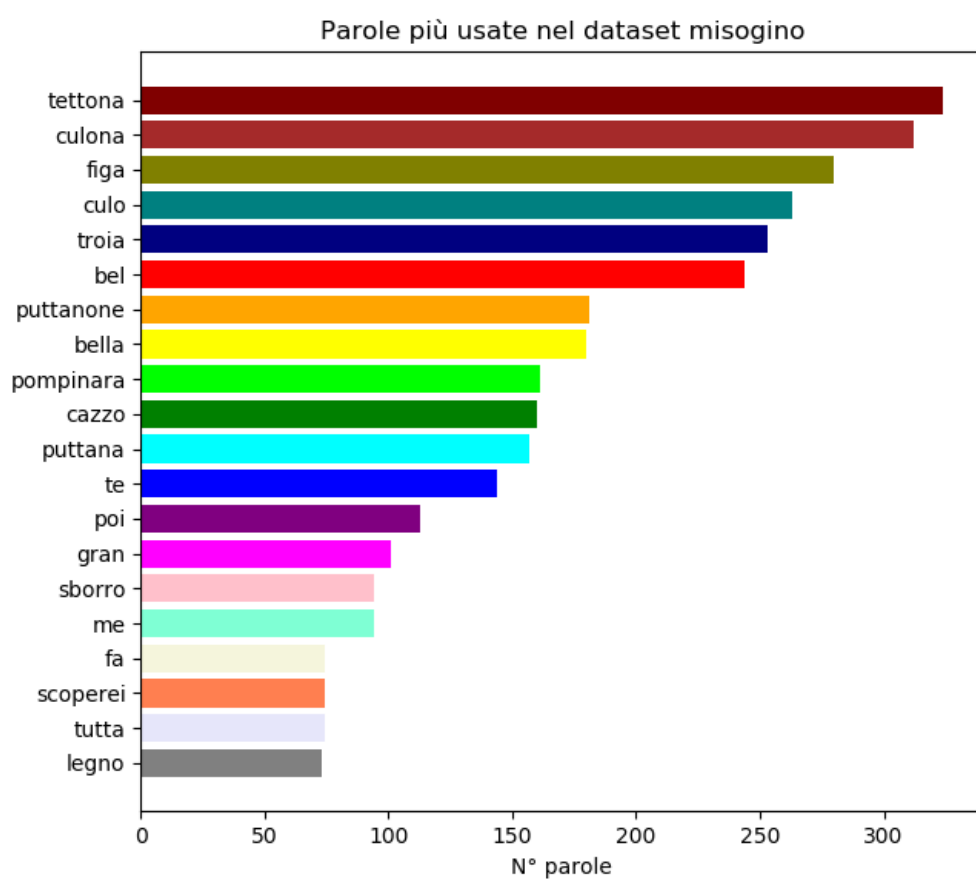
- Il numero di nomi femminili in modo da evidenziare un contesto femminile o maschile.
- Il numero di risate.
- Il numero di insulti.

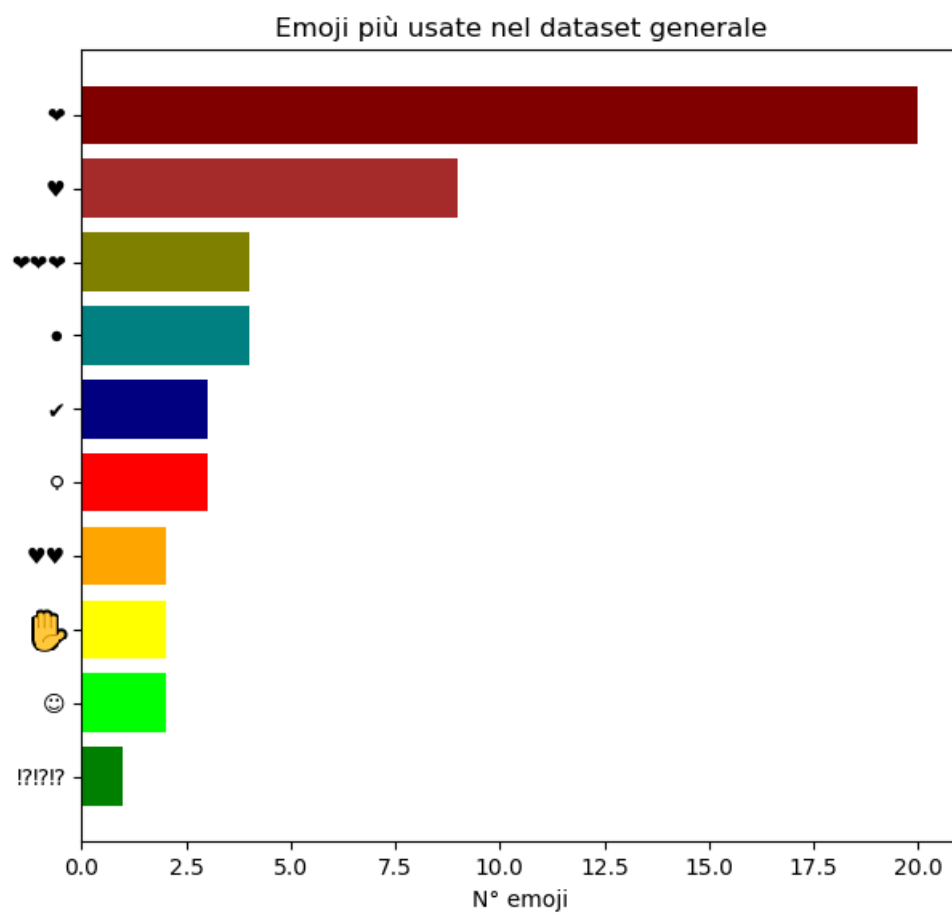
A questo punto per definire le successive caratteristiche ho trovato necessario evidenziare le parole e le emoji più usate nel dataset totale:

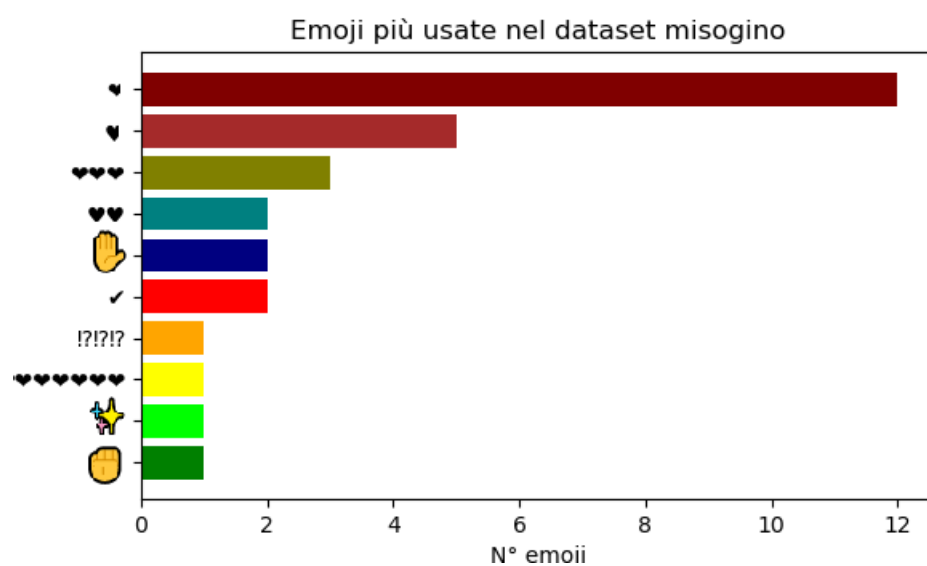
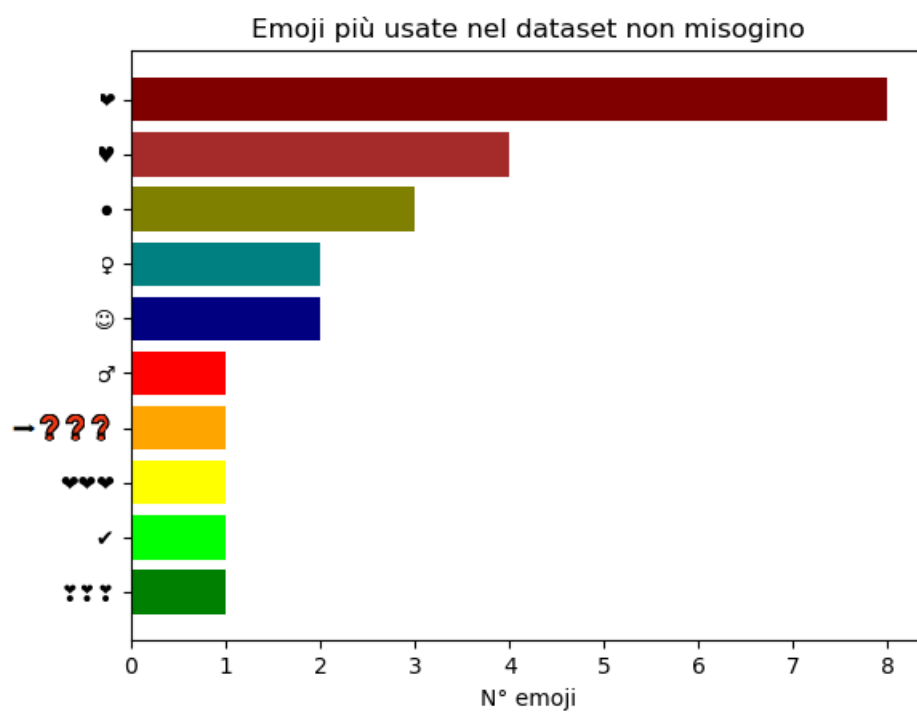


Le seguenti caratteristiche sono state calcolate anche per quanto riguarda la classe "aggressiveness". Da queste immagini si capisce che per quanto









riguarda le parole più utilizzate, come ci si aspettava si hanno insulti rivolti al genere femminile. Tuttavia la parola più presente, nonostante sia un insulto, viene utilizzata in contesti non misogini e quindi intuitivamente come imprecazioni non necessariamente rivolte alle donne. Questo ci permette di identificare dunque le parole più utilizzate contro le donne come ad esempio "Tettona", "Culona" o "Figa".

Lo stesso studio è stato poi applicato alle emoji, anche se nel dataset solo 43 tweets di 5000 contengono emoji, dunque non rappresenterà una caratteristica troppo rilevante. Sebbene non siano un numero elevato questo ci mostra che l'emoji come il "cuore" viene utilizzata con la stessa frequenza sia per contenuti misogini che non, mentre la "mano aperta" e il "pugno chiuso" vengono evidenziati come simboli presenti in contenuti misogini. Al contrario quelli indicanti il sesso e il cerchio nero vengono rilevati per contenuti non misogini. In conclusione dopo aver evidenziato queste ulteriori caratteristiche, ho calcolato per ogni tweet nel dataset:

- Il numero di parole tra le 10 più usate in tutto il dataset.
- Il numero di parole tra le 10 più usate nel dataset misogino.
- Il numero di parole tra le 10 più usate nel dataset non misogino.
- Il numero di emoji tra le 5 più usate in tutto il dataset.
- Il numero di emoji tra le 5 più usate nel dataset misoginoe.
- Il numero di emoji tra le 5 più usate nel dataset non misoginoe.

2.5 Embedding delle parole

Per poter effettivamente ottenere delle informazioni usabili dal testo è necessario eseguire una trasformazione in formato numerico in modo tale da passarle poi ai modelli.

Il word embedding è un tipo di rappresentazione distribuita che consente alle parole che hanno stesso significato di avere una rappresentazione simile. Dunque vengono creati vettori di numeri reali in uno spazio vettoriale predefinito, questi rappresentano delle parole o delle frasi attraverso l'utilizzo di strumenti e tecniche di apprendimento.

L'aspetto più interessante di questo approccio è che a differenza di metodi come "one-hot embeddings" che rappresentano solo le parole attraverso vettori di grandi dimensioni, questi altri codificano il significato e il ruolo della parola nel vettore mantenendo così le informazioni sulle relazioni delle diverse parole. Dunque tramite il word embedding possiamo sviluppare modelli che addestrino i vettori stessi tramite le reti neurali partendo da un insieme di testi selezionati e organizzati per facilitare le analisi.

Dunque per eseguire la trasformazione di ogni tweet ripulito in un vettore di numeri interi è necessario creare ed allenare un vocabolario con tutte le parole del dataset, le quali vengono associate ad un'indice cioè ad un numero. Per fare ciò viene utilizzata la classe `Tokenizer` di Keras ed utilizzato il metodo `"fit_on_texts"`.

```
In [14]: print(tokenizer.word_index)
{'puttana': 1, 'porca': 2, 'cazzo': 3, 'troia': 4, 'culona': 5,
'figa': 6, 'culo': 7, 'tettona': 8, 'bel': 9, 'puttanone': 10,
'me': 11, 'poi': 12, 'bella': 13, 'te': 14, 'fa': 15, 'merda': 16,
'pompinara': 17, 'fare': 18, 'cosa': 19, 'solo': 20, 'quando': 21,
'fatto': 22, 'ora': 23, 'così': 24, 'no': 25, 'sempre': 26, 'rt':
27, 'essere': 28, 'lurida': 29, 'legno': 30, 'quel': 31, 'bene':
32, 'mai': 33, 'mamma': 34, 'anni': 35, 'gran': 36, 'dire': 37,
'sborro': 38, 'casa': 39, 'due': 40, 'prima': 41, 'cagna': 42,
'tutta': 43, 'proprio': 44, 'detto': 45, 'asia': 46, 'zitta': 47,
'ancora': 48, 'vuoi': 49, 'argento': 50, 'culone': 51, 'dopo': 52,
'male': 53, 'so': 54, 'già': 55, 'donna': 56, 'bocca': 57, 'va':
58, 'pure': 59, 'scoperei': 60, 'foto': 61, 'senza': 62, 'voglio':
63, 'cose': 64, 'stupro': 65, 'ogni': 66, 'vita': 67, 'pezzo': 68,
'raga': 69, 'basta': 70, 'ce': 71, 'meglio': 72, 'madre': 73,
```

Figura 2.7: Esempio creazione vocabolario

Una volta ottenuto il vocabolario viene calcolata la sua dimensione che è pari a 11457 parole e viene ricercata la frase con massima lunghezza che è di 34 parole.

```
Dimensione del vocabolario: 11457 parole
Frase più lunga: 34 parole
```

Figura 2.8: Dimensione vocabolario e frase con massima lunghezza

Questo viene fatto poichè per poter dare in input i dati ottenuti ad una rete neurale le frasi devono essere tutte della stessa lunghezza, quindi per riportarle tutte alla stessa lunghezza uso la tecnica del `post_zero_padding`. Quando trovo una frase che è più piccola aggiungo degli zeri tramite la fun-

zione `pad_sequences` specificando dove si trovano le frasi, dove aggiungere gli 0 e la lunghezza massima che sarà di 34 parole

```
Frase in testo:
culona grande paracelsus falsa ignorante far paura

Frase in testo numerico:
[5, 103, 4896, 1366, 1152, 142, 483]
Lunghezza frase prima del Post_0_padding: 7

Frase in testo numerico con Post_0_padding :
[ 5 103 4896 1366 1152 142 483 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]
Lunghezza frase dopo il Post_0_padding: 34
```

Figura 2.9: Esempio Embedding di una frase del dataset

Come possiamo vedere in Figura 3.9 è stata eseguita la trasformazione della frase in testo numerico. La sua lunghezza prima del padding risulta essere pari a 7 parole, successivamente con l'aggiunta del padding viene raggiunta la lunghezza massima di 34 parole.

2.6 Suddivisione dei dati in training e test set

L'ultimo punto da affrontare prima di passare i nostri dati alla rete neurale è quello della suddivisione in training e test set. Per fare ciò è stato essenziale riordinare il testo in formato numerico, le 15 features e le etichette "misogini" (lo stesso è stato fatto per il modello dedicato alla previsione di frasi aggressive), in modo da recuperare i valori direttamente dal dataframe tramite la funzione "values", così da avere da una parte le caratteristiche e dall'altra le etichette. A questo punto utilizzando la funzione "train_test_split" abbiamo assegnato l'80% dei dati al training set e il restante 20% al test set, impostando il parametro "RANDOM_STATE", il quale permette di eseguire uno

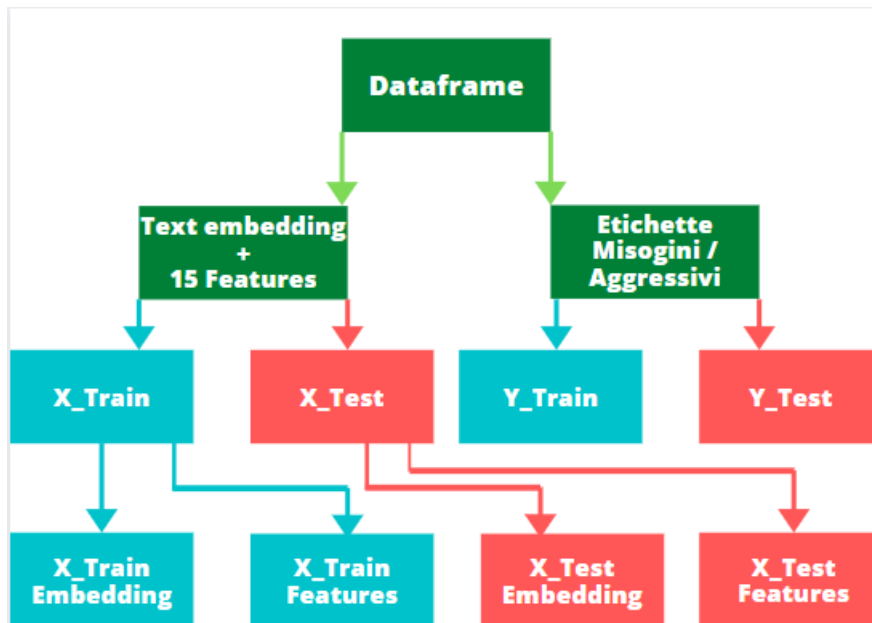


Figura 2.10: Suddivisione dei dati in traing e test set

shuffle dei dati e di evitare che vengano passati come input al modello prima tutti i dati di una classe e poi tutti i dati di un'altra, imparando così a riconoscere sempre prima una e poi l'altra.

Infine è necessario suddividere ulteriormente i dati da fornire come input al futuro modello in "Embedding" e "Features" poichè i primi verranno passati al livello di Embedding di Keras mentre i secondi verranno passati ad uno specifico livello di Input avente un numero di neuroni pari al numero delle features calcolate.

2.7 Struttura della reti

Sono stati prodotti diversi modelli nella costruzione della Artificial Neural Network finale, cambiando di volta la selezione delle variabili e l'apprendimento dei parametri in base a quali input verranno utilizzati, al numero di hidden layers che ha la rete e al numero di nodi presenti in ogni hidden layers.

I modelli creati risultano essere tutti Fully Connected Networks, cioè che ogni neurone è collegato con tutti quelli presenti nel layer successivo. Inoltre per quanto riguarda i primi due modelli sono caratterizzati dall'avere un solo livello di input dedicato al Embedding del testo, mentre con il terzo modello si aggiunge un livello di input dedicato alle Features calcolate in precedenza.

Il layer di Embedding viene utilizzato specificando come paramtri la dimensione dell'input ovvero quella del vocabolario, la dimensione dell'output ovvero quella dello spazio vettoriale in cui verranno incorporate le parole, ognuna di esse verrà rappresentata da un vettore avente questa dimensione. Infine la lunghezza delle nostre frasi che in questo caso sarà 34 ovvero la frase più lunga che abbiamo nel dataset. I pesi verranno inizializzati casualmente e verrà imparato un embedding per tutte le parole passate come vettore numerico. L'output di questo layer sarà l'ottenimento di una matrice, con valori normalizzati tra 1 e -1, che contiene un vettore per ogni parola nel testo.

2.7.1 Primo modello per previsioni misogini

Come è possibile vedere dall'immagine , viene sviluppato il modello sequenziale offerto da Keras e per poter applicare quanto detto vengono specificate 50 hidden features per parola nel livello di Embedding, generando così una matrice per ogni frase. Dato che si vuole collegare la rete direttamente al

Model: "sequential_14"

Layer (type)	Output Shape	Param #
embedding_14 (Embedding)	(None, 34, 50)	351000
flatten_14 (Flatten)	(None, 1700)	0
dense_36 (Dense)	(None, 32)	54432
dense_37 (Dense)	(None, 2)	66
Total params: 405,498		
Trainable params: 405,498		
Non-trainable params: 0		

Figura 2.11: Struttura primo modello per previsioni misogini

risultato di questo livello, è necessario utilizzare un layer di tipo Flatten() che permetterà di compattare tale matrice 2D in un vettore 1D.

Dopo aver eseguito questa trasformazione dei dati possiamo passare al livello MLP, formato dai due ulteriori livelli di tipo Dense(). Il primo è "hidden" ed ha 32 neuroni con funzione di attivazione "Relu" mentre il secondo è di "Output" ed ha 2 neuroni con funzione di attivazione "Softmax".

```
Epoch 1/5
125/125 [=====] - 1s 6ms/step - loss: 0.6932 - accuracy: 0.4358 - val_loss: 0.6931 - val_accuracy: 0.3890
Epoch 2/5
125/125 [=====] - 1s 5ms/step - loss: 0.6932 - accuracy: 0.4655 - val_loss: 0.6931 - val_accuracy: 0.4600
Epoch 3/5
125/125 [=====] - 1s 5ms/step - loss: 0.6931 - accuracy: 0.4770 - val_loss: 0.6931 - val_accuracy: 0.5070
Epoch 4/5
125/125 [=====] - 1s 5ms/step - loss: 0.6931 - accuracy: 0.4652 - val_loss: 0.6931 - val_accuracy: 0.4840
Epoch 5/5
125/125 [=====] - 1s 5ms/step - loss: 0.6931 - accuracy: 0.4640 - val_loss: 0.6931 - val_accuracy: 0.4060
```

Figura 2.12: Addestramento primo modello per previsioni misogini

L'allenamento del primo modello vede il ripetersi di 5 epoche con un batch_size pari a 32. L'ottimizzatore utilizzato è stato Adam, come funzione

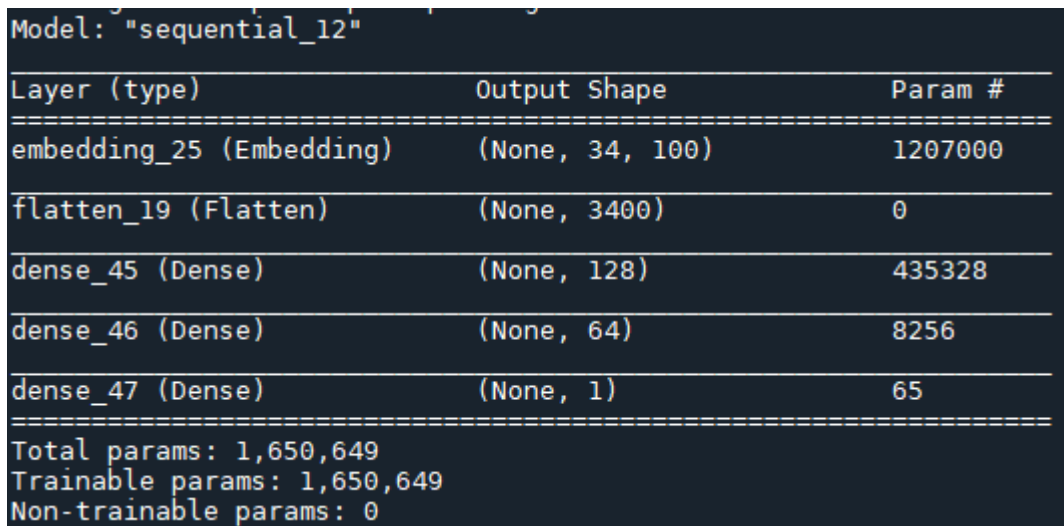
di loss si è usata la "binary_crossentropy" e come metrica per il calcolo delle performance si è utilizzata l'Accuracy.

```
32/32 [=====] - 0s 2ms/step - loss: 0.6931 - accuracy: 0.4060
```

Figura 2.13: Accuratezza primo modello per previsioni misogini

2.7.2 Secondo modello per previsioni misogini

Tuttavia le prestazioni del primo modello non sono soddisfacenti, dunque sono state provate diverse modifiche fino a raggiungere questo secondo modello. In questo abbiamo aumentato a 100 il numero di hidden features, è stato aggiunto un terzo livello Dense() con 128 neuroni e nell'ultimo livello viene utilizzata come funzione di attivazione la "Sigmoid"



```
Model: "sequential_12"
```

Layer (type)	Output Shape	Param #
embedding_25 (Embedding)	(None, 34, 100)	1207000
flatten_19 (Flatten)	(None, 3400)	0
dense_45 (Dense)	(None, 128)	435328
dense_46 (Dense)	(None, 64)	8256
dense_47 (Dense)	(None, 1)	65

=====
Total params: 1,650,649
Trainable params: 1,650,649
Non-trainable params: 0

Figura 2.14: Struttura secondo modello per previsioni misogini

L'allenamento del secondo modello vede il ripetersi di 5 epoche con un batch_size pari a 32. L'ottimizzatore utilizzato è stato Adam, come funzione

di loss si è usata la "binary_crossentropy" e come metrica per il calcolo delle performance si è utilizzata l'Accuracy.

```
Train on 4000 samples, validate on 1000 samples
Epoch 1/5
4000/4000 [=====] - 5s 1ms/sample - loss: 0.4120 - accuracy: 0.8087 - val_loss:
0.3078 - val_accuracy: 0.8630
Epoch 2/5
4000/4000 [=====] - 4s 922us/sample - loss: 0.1189 - accuracy: 0.9563 - val_loss:
0.4150 - val_accuracy: 0.8450
Epoch 3/5
4000/4000 [=====] - 4s 979us/sample - loss: 0.0299 - accuracy: 0.9905 - val_loss:
0.4453 - val_accuracy: 0.8350
Epoch 4/5
4000/4000 [=====] - 9s 2ms/sample - loss: 0.0110 - accuracy: 0.9965 - val_loss:
0.5194 - val_accuracy: 0.8420
Epoch 5/5
4000/4000 [=====] - 4s 1ms/sample - loss: 0.0079 - accuracy: 0.9970 - val_loss:
0.5556 - val_accuracy: 0.8350
1000/1000 [=====] - 0s 83us/sample - loss: 0.5556 - accuracy: 0.8350
```

Figura 2.15: Addestramento secondo modello per previsioni misogini

In questo secondo modello si è raggiunta una buona accuratezza, arrivando ad avere l'83%.

2.7.3 Terzo modello per previsioni misogini

In questo terzo modello, come detto in precedenza, viene aggiunto un secondo layer di input, il quale è caratterizzato dal numero di neuroni pari al numero di features calcolate e prese in input per ogni frase. A questo punto i due livelli di input, cioè quello dove viene inserito l'embedding della frase e quello dove vengono inserite le features, sono concatenati insieme per poi passare l'output ai livelli di tipo Dense() come in precedenza.

Per quanto riguarda questo terzo modello sono stati inserite prima solo due features che mantenevano il numero di punti esclamativi e il numero di tag degli utenti. L'allenamento, anche in questo caso vede il ripetersi di 5 epoche con un batch_size pari a 32. L'ottimizzatore utilizzato è stato Adam, come funzione di loss si è usata la "binary_crossentropy" e come metrica per il calcolo delle performance si è utilizzata l'Accuracy.

Model: "model_25"

Layer (type)	Output Shape	Param #	Connected to
input_72 (InputLayer)	(None, 34)	0	
embedding_25 (Embedding)	(None, 34, 100)	1207000	input_72[0][0]
flatten_26 (Flatten)	(None, 3400)	0	embedding_25[0][0]
input_73 (InputLayer)	(None, 2)	0	
concatenate_30 (Concatenate)	(None, 3402)	0	flatten_26[0][0] input_73[0][0]
dense_70 (Dense)	(None, 128)	435584	concatenate_30[0][0]
dense_71 (Dense)	(None, 64)	8256	dense_70[0][0]
dense_72 (Dense)	(None, 1)	65	dense_71[0][0]

Total params: 1,650,905
 Trainable params: 1,650,905
 Non-trainable params: 0

Figura 2.16: Struttura terzo modello per previsioni misogini

```

Train on 4000 samples, validate on 1000 samples
Epoch 1/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.4084 - accuracy: 0.8090 - val_loss: 0.3000 -
val_accuracy: 0.8750
Epoch 2/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.1247 - accuracy: 0.9540 - val_loss: 0.3620 -
val_accuracy: 0.8500
Epoch 3/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0298 - accuracy: 0.9895 - val_loss: 0.4460 -
val_accuracy: 0.8440
Epoch 4/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0117 - accuracy: 0.9967 - val_loss: 0.5280 -
val_accuracy: 0.8420
Epoch 5/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0070 - accuracy: 0.9975 - val_loss: 0.5711 -
val_accuracy: 0.8410

```

Figura 2.17: Allenamento terzo modello per previsioni misogini

Come è possibile vedere da quest'ultimo allenamento, l'accuratezza del modello è rimasta pressoché invariata rispetto al precedente modello. A questo punto vengono aggiunte tutte e 15 le features calcolate precedentemente.

```
Model: "model_28"
```

Layer (type)	Output Shape	Param #	Connected to
input_78 (InputLayer)	(None, 34)	0	
embedding_28 (Embedding)	(None, 34, 100)	1207000	input_78[0][0]
flatten_29 (Flatten)	(None, 3400)	0	embedding_28[0][0]
input_79 (InputLayer)	(None, 15)	0	
concatenate_33 (Concatenate)	(None, 3415)	0	flatten_29[0][0] input_79[0][0]
dense_79 (Dense)	(None, 128)	437248	concatenate_33[0][0]
dense_80 (Dense)	(None, 64)	8256	dense_79[0][0]
dense_81 (Dense)	(None, 1)	65	dense_80[0][0]

```

Total params: 1,652,569
Trainable params: 1,652,569
Non-trainable params: 0

```

Figura 2.18: Struttura terzo modello con 15 features

```

Train on 4000 samples, validate on 1000 samples
Epoch 1/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.4488 - accuracy: 0.7933 - val_loss:
0.3583 - val_accuracy: 0.8580
Epoch 2/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.1645 - accuracy: 0.9375 - val_loss:
0.3291 - val_accuracy: 0.8660
Epoch 3/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0514 - accuracy: 0.9852 - val_loss:
0.4166 - val_accuracy: 0.8480
Epoch 4/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0188 - accuracy: 0.9950 - val_loss:
0.4252 - val_accuracy: 0.8490
Epoch 5/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0178 - accuracy: 0.9958 - val_loss:
0.4587 - val_accuracy: 0.8610

```

Figura 2.19: Allenamento terzo modello con 15 features

Possiamo notare che l'accuratezza del modello con l'aggiunta delle 15 features è aumentata fino a raggiungere l'86%

2.7.4 Modello per previsioni aggressive

Gli stessi modelli visti per la previsione di contenuti misogini sono stati provati anche per la previsione di contenuti aggressivi. Tuttavia quello che ha garantito prestazioni migliori è stato il terzo modello passandogli tutte e 15 le features calcolate rispetto alla "Aggressiveness".

```
Train on 4000 samples, validate on 1000 samples
Epoch 1/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.5315 - accuracy: 0.7226 -
val_loss: 0.4177 - val_accuracy: 0.7895
Epoch 2/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.2634 - accuracy: 0.8781 -
val_loss: 0.4659 - val_accuracy: 0.7990
Epoch 3/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.1110 - accuracy: 0.9609 -
val_loss: 0.5716 - val_accuracy: 0.7865
Epoch 4/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0627 - accuracy: 0.9789 -
val_loss: 0.6141 - val_accuracy: 0.7725
Epoch 5/5
4000/4000 [=====] - 5s 1ms/step - loss: 0.0462 - accuracy: 0.9831 -
val_loss: 0.7884 - val_accuracy: 0.7480
```

Figura 2.20: Allenamento modello per previsioni aggressive

Come detto in precedenza, il dataset inizialmente era sbilanciato, dunque ho ipotizzato che l'accuratezza del modello fosse inferiore a causa di questo problema. Dopo aver effettuato l'up-sampling e aver così bilanciato il dataset, è stato rieseguito l'allenamento dello stesso modello.

```
Train on 4333 samples, validate on 1084 samples
Epoch 1/5
4333/4333 [=====] - 5s 1ms/step - loss: 0.5892 - accuracy: 0.7090 -
val_loss: 0.4056 - val_accuracy: 0.7961
Epoch 2/5
4333/4333 [=====] - 5s 1ms/step - loss: 0.2268 - accuracy: 0.9064 -
val_loss: 0.3011 - val_accuracy: 0.8722
Epoch 3/5
4333/4333 [=====] - 5s 1ms/step - loss: 0.0915 - accuracy: 0.9685 -
val_loss: 0.3379 - val_accuracy: 0.8962
Epoch 4/5
4333/4333 [=====] - 5s 1ms/step - loss: 0.0544 - accuracy: 0.9815 -
val_loss: 0.3408 - val_accuracy: 0.8865
Epoch 5/5
4333/4333 [=====] - 5s 1ms/step - loss: 0.0422 - accuracy: 0.9844 -
val_loss: 0.3567 - val_accuracy: 0.8805
```

Figura 2.21: Allenamento modello per previsioni aggressive post Upsampling

Possiamo subito notare che l'accuratezza del modello per la previsione di contenuti aggressivi è passata da un 75% ad un 86%, grazie al bilanciamento del dataset.

Capitolo 3

Test e valutazione dei risultati

3.1 Salvataggio e caricamento modelli

A questo punto per poter implementare una interfaccia utente che consenta di sottoporre e classificare un testo qualsiasi digitato dall'utente, è stato necessario salvare prima la struttura dei due modelli finali in formato json, e successivamente il valore dei pesi degli archi in formato h5. Questo permetterà di caricare tali modelli già addestrati su un nuovo file, evitando di eseguire l'addestramento ad ogni esecuzione del programma.

```
# serialize model to JSON
model_json = model_final.to_json()
with open("Model/model_misogino.json", "w+") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model_final.save_weights("Model_h5/model_misogino_h5.h5")
print("Modello misogino salvato correttamente")
```

Figura 3.1: Salvataggio modello per predizione contenuti misogini

```
{
  "class_name": "Model",
  "config": {
    "name": "model_1",
    "layers": [
      {
        "name": "input_1",
        "class_name": "InputLayer",
        "config": {
          "batch_input_shape": [null, 34],
          "dtype": "float32",
          "sparse": false,
          "name": "input_1",
          "inbound_nodes": []
        }
      },
      {
        "name": "embedding_1",
        "class_name": "Embedding",
        "config": {
          "name": "embedding_1",
          "trainable": true,
          "batch_input_shape": [null, 34],
          "dtype": "float32",
          "input_dim": 11384,
          "output_dim": 100,
          "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": 34,
          "inbound_nodes": [
            [
              [
                "input_1",
                0,
                {}
              ]
            ]
          ],
          "class_name": "Flatten",
          "config": {
            "name": "flatten_1",
            "trainable": true,
            "dtype": "float32",
            "data_format": "channels_last",
            "inbound_nodes": [
              [
                [
                  "embedding_1",
                  0,
                  {}
                ]
              ]
            ],
            "class_name": "InputLayer",
            "config": {
              "batch_input_shape": [null, 15],
              "dtype": "float32",
              "sparse": false,
              "name": "input_2",
              "inbound_nodes": [],
              "class_name": "Concatenate",
              "config": {
                "name": "concatenate_1",
                "trainable": true,
                "dtype": "float32",
                "axis": -1,
                "inbound_nodes": [
                  [
                    [
                      "flatten_1",
                      0,
                      {}
                    ],
                    [
                      "input_2",
                      0,
                      {}
                    ]
                  ]
                ],
                "class_name": "Dense",
                "config": {
                  "name": "dense_1",
                  "trainable": true,
                  "dtype": "float32",
                  "units": 128,
                  "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_1",
                        0,
                        {}
                      ],
                      [
                        "dense_1",
                        0,
                        {}
                      ]
                    ]
                  ],
                  "class_name": "Dense",
                  "config": {
                    "name": "dense_2",
                    "trainable": true,
                    "dtype": "float32",
                    "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": [
                      [
                        [
                          "dense_1",
                          0,
                          {}
                        ],
                          [
                            "dense_2",
                            0,
                            {}
                          ]
                        ]
                      ],
                      "class_name": "Dense",
                      "config": {
                        "name": "dense_3",
                        "trainable": true,
                        "dtype": "float32",
                        "units": 2,
                        "activation": "sigmoid",
                        "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_2",
                              0,
                              {}
                            ]
                          ]
                        ],
                        "input_layers": [
                          [
                            "input_1",
                            0,
                            {}
                          ],
                          [
                            "input_2",
                            0,
                            {}
                          ]
                        ],
                        "output_layers": [
                          [
                            "dense_3",
                            0,
                            {}
                          ]
                        ],
                        "keras_version": "2.3.1",
                        "backend": "tensorflow"
                      }
                    ]
                  ]
                }
              ]
            ]
          ]
        }
      ]
    ]
  }
}
```

Figura 3.2: Modello per predizione contenuti misogini in json

```
json_file = open('Model/model_misogino.json', 'r')
modello_misogino_json = json_file.read()
json_file.close()
modello_misogino = tf.keras.models.model_from_json(modello_misogino_json)
# load weights into new model
modello_misogino.load_weights("Model_h5/model_misogino_h5.h5")
print("Modello per il riconoscimento di contenuti misogini caricato")

json_file2 = open('Model/model2_aggressività.json', 'r')
modello_aggressivo_json = json_file2.read()
json_file2.close()
modello_aggressivo = tf.keras.models.model_from_json(modello_aggressivo_json)
# load weights into new model
modello_aggressivo.load_weights("Model_h5/model2_h5_aggressività.h5")
print("Modello per il riconoscimento di contenuti aggressivi caricato")
```

Figura 3.3: Caricamento modelli

Infine, dopo aver caricato i modelli pre-allenati, viene richiesto all'utente di inserire una frase per poterla classificare in base alle due classi. Quello che viene fatto alla frase inserita dall'utente è la pulizia del testo tramite il cleaning applicato ai tweets del dataset, successivamente vengono calcolate tutte le 15 features descritte precedentemente. A questo punto, alla frase ripulita rimane solo da applicare la trasformazione in formato numerico, dopodiché avremo sia l'input per l'embedding sia l'input per le features da passare ai modelli caricati precedentemente.

```
predizione_m = modello_misogino.predict([valori_embedding,valori_features])
predizione_non_misogino= predizione_m[0][0]*100
predizione_misogino= predizione_m[0][1]*100
predizione_finale_misogino = np.argmax(predizione_m, axis=1)

predizione_a = modello_aggressivo.predict([valori_embedding,valori_features])
predizione_non_aggressivo= predizione_a[0][0]*100
predizione_aggressivo= predizione_a[0][1]*100
predizione_finale_aggressivo = np.argmax(predizione_a, axis=1)

if predizione_finale_misogino[0] == 0:
    #if predizione_finale2[0] == 0:
        print("Il testo fornito non è misogino al ", "%.2f" % predizione_non_misogino,"%")
else:
    if predizione_finale_aggressivo[0] == 0:
        print("Il testo fornito è misogino al ", "%.2f" % predizione_misogino,
              "% e non è aggressivo al ", "%.2f" % predizione_non_aggressivo,"%")
    else:
        print("Il testo fornito è misogino al ", "%.2f" % predizione_misogino,
              "% ed è aggressivo al ", "%.2f" % predizione_aggressivo,"%")
```

Figura 3.4: Codice predizione frase

Vediamo adesso alcuni esempi di test eseguiti tramite l'inserimento di una frase da tastiera e le relative predizioni dei modelli.

```
Modello per il riconoscimento di contenuti misogini caricato
Modello per il riconoscimento di contenuti aggressivi caricato

Inserire la frase da classificare:
ma quanto vi credete simpatici quando fate battute del tipo donna schiava,
torna in cucina?
Il testo fornito non è misogino al 51.72 %
```

Figura 3.5: Primo test dove la frase viene riconosciuta come non misogina

```
Modello per il riconoscimento di contenuti misogini caricato
Modello per il riconoscimento di contenuti aggressivi caricato

Inserire la frase da classificare:
Donna vai cucina che è il tuo posto!!!
Il testo fornito è misogino al 99.50 % e non è aggressivo al 59.66 %
```

Figura 3.6: Secondo test dove la frase viene riconosciuta come misogina e non aggressiva

```
Modello per il riconoscimento di contenuti misogini caricato
Modello per il riconoscimento di contenuti aggressivi caricato

Inserire la frase da classificare:
Troia vai in cucina che è il tuo posto
Il testo fornito è misogino al 99.92 % ed è aggressivo al 64.00 %
```

Figura 3.7: Terzo test dove la frase viene riconosciuta come misogina e aggressiva

3.2 Valutazione dei modelli prodotti

Per poter valutare i modelli prodotti è stata utilizzata la matrice di confusione.

	Predicted	
	Positive	Negative
Actual True	TP	FN
Actual False	FP	TN

Questa immagine mostra un esempio di matrice binaria, proprio come nel nostro caso, dove ogni colonna rappresenta i valori predetti, mentre ogni riga rappresenta i valori reali. L'elemento sulla riga i e sulla colonna j è il numero di casi in cui il classificatore ha classificato la classe "vera" i come classe j . Attraverso questa matrice è osservabile se vi è "confusione" nella classificazione di diverse classi.

Tutti questi valori, (TP, FN, FP, TN) consentono di calcolare delle metriche di performance:

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

La precisione è la capacità di un classificatore di non assegnare un'istanza positiva che è in realtà negativa. Per ogni classe è definito come il rapporto tra veri positivi e la somma di veri e falsi positivi. Detto in un altro modo, "per tutte le istanze classificate come positive, quale percentuale era corretta?".

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

La recall è l'abilità di un classificatore di trovare tutte le istanze positive. Per ogni classe è definito come il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi negativi. Detto in altri termini, “per tutte le istanze che erano effettivamente positive, quale percentuale è stata classificata correttamente?”.

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

L'F-score è una media armonica ponderata delle metriche Precision e Recall in modo tale che il punteggio migliore sia 1 e il peggiore sia 0.

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

L'accuracy indica l'accuratezza del modello come dice il nome. Pertanto, la migliore accuratezza è 1, mentre la peggiore è 0. Infine un altro valore che viene riportato nelle tabelle dei risultati è il Support, cioè il numero di campioni analizzati per ogni classe.

3.2.1 Analisi modello previsioni misogini

I risultati ottenuti dal modello che prevede frasi misogini e che prende in input sia il testo in formato numerico che le features sono:

- Precision: 0= 83% 1= 86%
- Recall: 0= 88% 1= 81%
- F-score: 0= 85% 1= 83%
- Support: 0= 520 1= 480
- Accuracy: 86% su 1000 samples

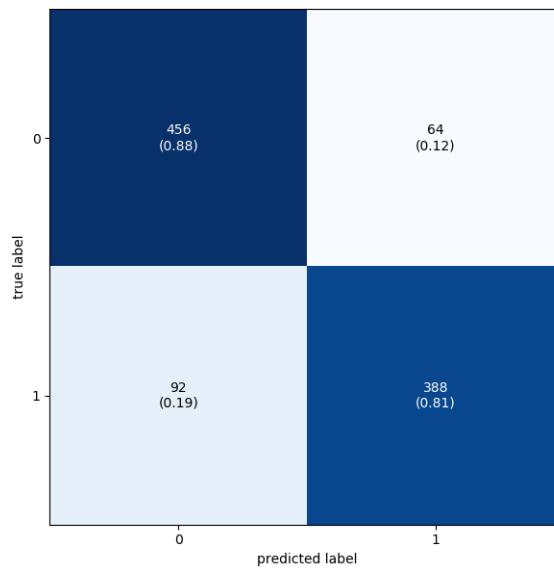


Figura 3.8: Matrice di confusione misogini

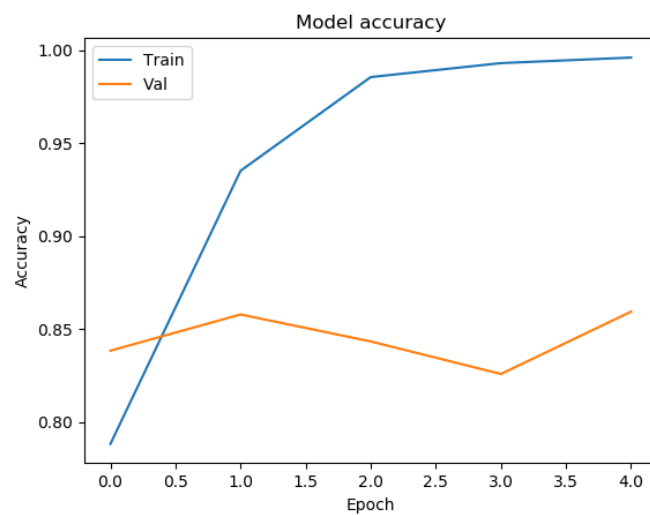


Figura 3.9: Accuracy durante l'addestramento del modello misogini

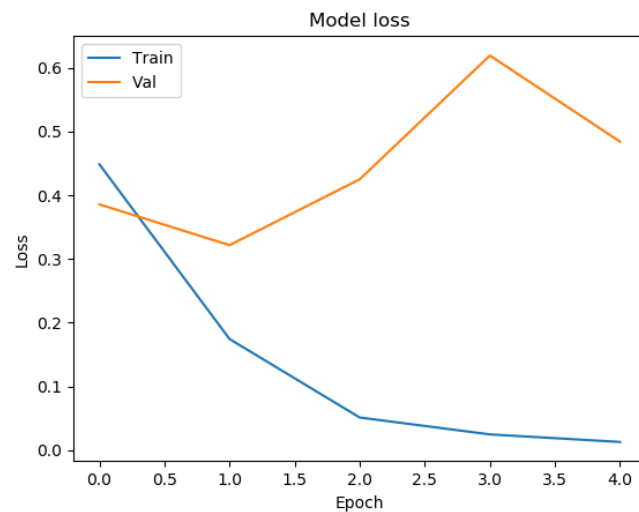


Figura 3.10: Loss durante l'addestramento del modello misogini

3.2.2 Analisi modello previsioni aggressive prima dell'Up-sampling

I risultati ottenuti dal modello che prevede frasi aggressive prima dell'up-sampling e che prende in input sia il testo in formato numerico che le features sono:

- Precision: 0= 79% 1= 76%
- Recall: 0= 88% 1= 62%
- F-score: 0= 83% 1= 68%
- Support: 0= 618 1= 382
- Accuracy: 75% su 1000 samples

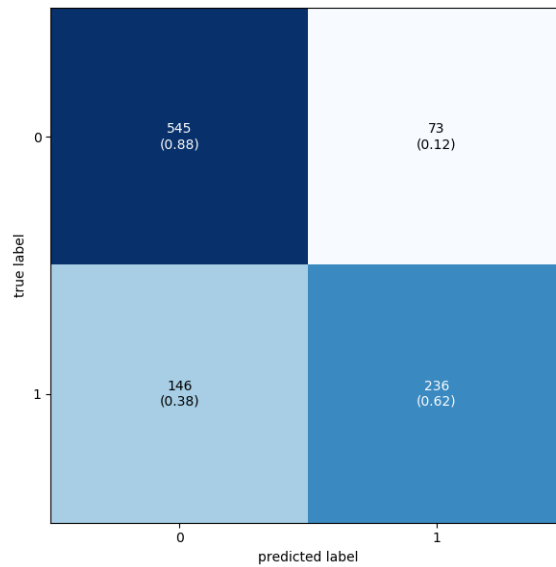


Figura 3.11: Matrice di confusione aggerssive prima up-sampling

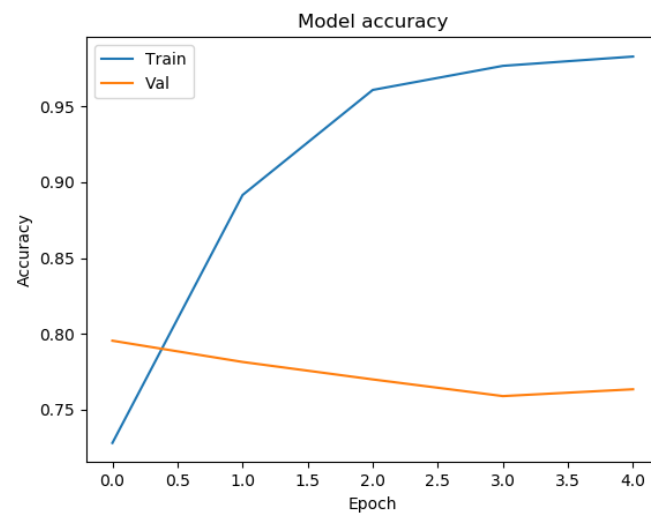


Figura 3.12: Accuracy durante l'addestramento del modello aggressive prima up-sampling

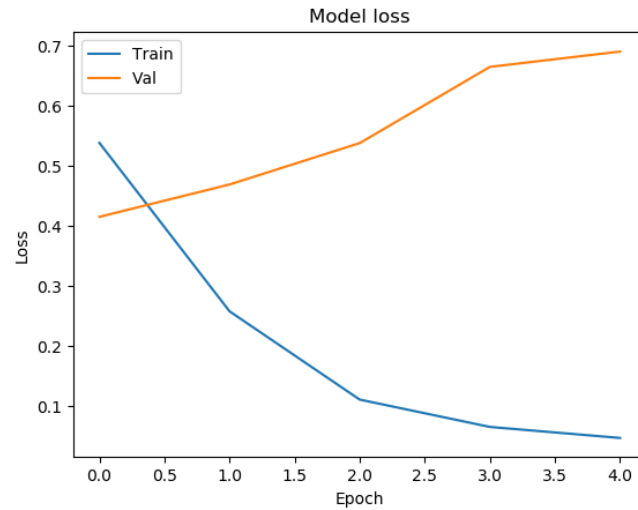


Figura 3.13: Loss durante l'addestramento del modello aggressive prima up-sampling

3.2.3 Analisi modello previsioni aggressive dopo dell'Up-sampling

I risultati ottenuti dal modello che prevede frasi aggressive dopo dell'up-sampling e che prende in input sia il testo in formato numerico che le features sono:

- Precision: 0= 91% 1= 83%
- Recall: 0= 88% 1= 88%
- F-score: 0= 89% 1= 85%
- Support: 0= 638 1= 446
- Accuracy: 86% su 1100 samples

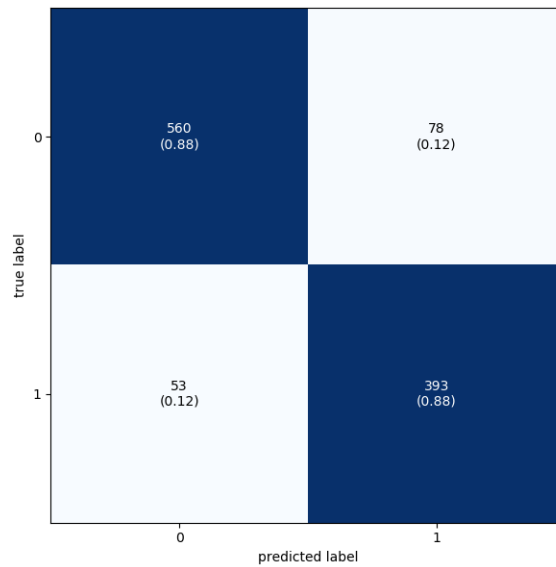


Figura 3.14: Matrice di confusione aggressive dopo up-sampling

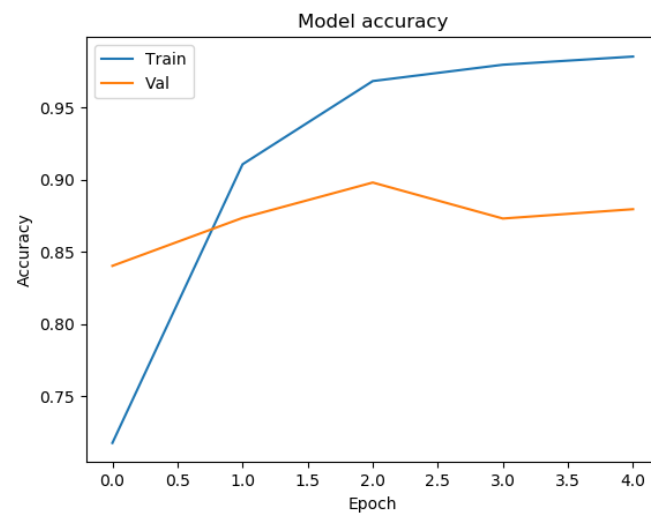


Figura 3.15: Accuracy durante l'addestramento del modello aggressive dopo up-sampling

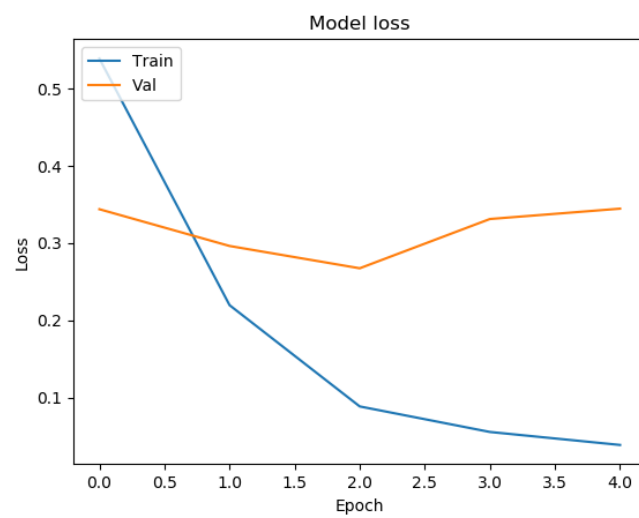


Figura 3.16: Loss durante l'addestramento del modello aggressive dopo up-sampling

Conclusioni

L'obiettivo di questo progetto è stato quello di riuscire a carpire e identificare i contenuti misogini delle frasi e dei tweets in modo tale da creare un modello in grado di classificare automaticamente le frasi scritte dagli utenti.

Tutti i modelli prodotti hanno mostrato delle buone prestazioni, tuttavia una svolta al problema è stata l'aggiunta dell'upsampling dei dati in modo da riequilibrare tutto il dataset. Inoltre l'aggiunta delle features ha permesso di incrementare leggermente le performance in modo da raggiungere buone prestazioni sia per il modello che prevede contenuti misogini (86% di accuratezza) sia per quello che prevede contenuti aggressivi (86% di accuratezza).

Infine il salvataggio dei modelli in formato json e dei pesi in h5, ha permesso il loro caricamento in un differente file, così da evitare l'addestramento ogni volta che si volesse eseguire la previsione di una frase inserita dagli utenti.